UNIVERSITE PARIS 8 – VINCENNES-SAINT-DENIS

U.F.R Arts, Philosophie, Esthétique

THESE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE PARIS 8

Discipline : Esthétique, Science et Technologie des Arts

spécialité Images Numériques

présentée et soutenue publiquement par

Antoine ZANUTTINI

Du photoréalisme au rendu expressif en image 3D temps réel dans le jeu vidéo : programmation graphique pour la profondeur de champ, la matière, la réflexion, les fluides et les contours.

le 14 novembre 2012

Directeur de thèse : Marie-Hélène TRAMUS

JURY

M. Jerome BANAL, Directeur technique	DONTNOD Entertainment
Mme Chu-Yin CHEN, Professeur	Université Paris 8
M. Oskar GUILBERT, Directeur général	DONTNOD Entertainment
M. Fawzi NASHASHIBI, Chercheur HDR	INRIA – responsable équipe-projet IMARA
Mme Marie-Hélène TRAMUS, Professeur	Université Paris 8

UNIVERSITE PARIS 8 – VINCENNES-SAINT-DENIS

U.F.R Arts, Philosophie, Esthétique

THESE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE PARIS 8

Discipline : Esthétique, Science et Technologie des Arts

spécialité Images Numériques

présentée et soutenue publiquement par

Antoine ZANUTTINI

Du photoréalisme au rendu expressif en image 3D temps réel dans le jeu vidéo : programmation graphique pour la profondeur de champ, la matière, la réflexion, les fluides et les contours.

le 14 novembre 2012

Directeur de thèse : Marie-Hélène TRAMUS

JURY

M. Jerome BANAL, Directeur technique	DONTNOD Entertainment
Mme Chu-Yin CHEN, Professeur	Université Paris 8
M. Oskar GUILBERT, Directeur général	DONTNOD Entertainment
M. Fawzi NASHASHIBI, Chercheur HDR	INRIA – responsable équipe-projet IMARA
Mme Marie-Hélène TRAMUS, Professeur	Université Paris 8

Résumé

Du photoréalisme au rendu expressif en image 3D temps réel dans le jeu vidéo : programmation graphique pour la profondeur de champ, la matière, la réflexion, les fluides et les contours.

Cette étude vise à sortir de l'esthétique standardisée des jeux vidéo par l'apport de nouvelles techniques de représentation pour l'image numérique temps réel.

Le rendu dit « photoréaliste » manque souvent de contrôle et de flexibilité pour l'artiste qui cherche à aller au-delà de la fidélité au réel. La crédibilité et l'immersion passent alors par une stylisation de l'image pour proposer un visuel convaincant et esthétique. Le rendu dit « expressif » va plus loin en prenant en compte le regard personnel de l'artiste tout en se basant sur des caractéristiques et des phénomènes issus du réel pour les détourner. Nous montrerons que le photoréalisme et le rendu expressif se rejoignent et se complètent sur de nombreux points.

Trois thèmes différents seront présentés du point de vue du photoréalisme et accompagnés de la création de techniques personnelles. Le thème du flou de profondeur de champ nous amènera à prendre en compte la forme de la lentille de la caméra virtuelle à travers l'algorithme des *Hexagonal Summed Area Tables*. Nous aborderons ensuite la matière, la lumière et surtout la réflexion spéculaire ambiante et l'importance de la parallaxe pour celle-ci. Notre troisième thème sera celui du mouvement des fluides et de l'advection de textures en fonction du courant afin d'ajouter du détail facilement et efficacement.

Ces mêmes sujets seront ensuite intégrés au rendu expressif et utilisés comme des outils d'expression pour l'artiste, à travers la création d'un effet « rêve », de fluides en espace-écran et du travail de la matière hachurée. Enfin, nous présenterons nos créations spécifiquement dédiées au rendu expressif et à la stylisation des contours.

Mots-clés : jeux vidéo / infographie / temps-réel (informatique) / algorithmes / traitement d'image - techniques numériques

Abstract

From photorealism to expressive rendering in real-time 3D imagery for video games: graphic programming for depth of field, material, reflection, fluids and strokes.

This study seeks to go beyond the standardized video games aesthetics by adding new depiction techniques for real-time digital imagery.

Photorealistic rendering is often limited in control and flexibility for an artist who searches beyond fidelity to real. Achieving credibility and immersion will then often require stylization for the view to be more convincing and aesthetic. Expressive rendering goes further by taking care of the artist's personal vision, while being based on some real life attributes and phenomena, altering them at the same time. We will show that photorealism and expressive rendering join and complete each other under numerous aspects.

Three themes related to photorealism will be presented, then some personal original techniques will be introduced. The theme of depth of field will lead us to consider the shape of the virtual camera's lens through the Hexagonal Summed Area Table algorithm. We will then look at material, light and especially ambient and specular reflections and the importance of parallax correction with regards to them. Our third theme will be the rendering of fluid motion and the advection of textures according to the flow for easy and efficient detail addition.

These three subjects will then be integrated into expressive rendering and used as expression tools for the artist, through the creation of a dream effect, of screen-space rendered fluids, and of hatched material shading. Finally, we will show our creations specifically dedicated to expressive rendering and stroke stylization.

Key-words : video games / computer graphics / real-time (computer science) / algorithms / image processing – numerical techniques

Remerciements

Je tiens tout d'abord à remercier Marie-Hélène Tramus pour ses conseils au cours de la recherche et de l'écriture de la thèse. Je remercie également les membres du jury de leur intérêt pour mon travail, ainsi que les prérapporteurs : Gilles Methel et Fawzi Nashashibi.

Je remercie l'entreprise DONTNOD Entertainment et en particulier Oskar Guilbert et Hervé Bonin pour m'avoir accueilli en CIFRE et de m'avoir offert un cadre de recherche très stimulant.

Je remercie particulièrement Jerome Banal, directeur technique chez DONTNOD, qui a toujours été source d'encouragements dans la réalisation de mes recherches. Mes remerciements vont également à Sébastien Lagarde et Laury Michel, membres de l'équipe « engine » ainsi qu'à Samuel Hocevar et Stéphane Hubart, membres de l'équipe « Fluidz », pour leurs collaborations et leurs conseils au cours de mes recherches.

Je remercie également Frédéric Cross, Timothée Letourneux et Thomas Iché avec qui j'ai travaillé régulièrement. Je remercie l'ensemble des employés de DONTNOD pour leur travail formidable.

Je remercie enfin mes camarades de doctorat pour leur soutien durant la rédaction du manuscrit : Suzanne Beer, Maxime Causeret, Somphout Chanhthaboutdy, Gilles-Alexandre Deschaud, Judith Guez, Karleen Groupierre et Edwige Lelièvre.

Je terminerai par ma famille qui m'a encouragé dans mes projets tout au long de mon cursus.

Sommaire

Introduction générale

Le développement de l'informatique a permis l'émergence de l'interactivité numérique grâce à laquelle la découverte de mondes virtuels devient possible. L'exploration de ces univers peut désormais s'effectuer librement, le joueur étant capable de parcourir les décors et d'influer sur le monde virtuel d'une manière fluide, presque naturelle. Les œuvres qui font appel à cette interactivité sont créées dans deux contextes différents : les œuvres artistiques interactives et les jeux vidéo. Les jeux vidéo ont généralement été considérés comme faisant partie de l'industrie du divertissement. Les impératifs financiers, le marketing et le ciblage sont les premières contraintes qui façonnent les jeux vidéo commerciaux. Cependant, de même que le cinéma ne se résume pas à ses blockbusters, le jeu vidéo a également ses grands maîtres et ses discrets innovateurs. La scène du jeu indépendant s'est notamment incroyablement développée ces dernières années, et avec elle l'arrivée d'une grande diversité de jeux.

En 2009, c'était déjà plus de 25 millions de joueurs en France[1], que ce soit avec le jeu console, le jeu sur ordinateur, le jeu sur internet ou sur téléphone portable. Le public s'est diversifié et des démarches créatives et originales sont désormais viables et peuvent trouver un public.

Contexte

Le jeu vidéo n'est pas forcément perçu comme une œuvre d'art à part entière, avec sa spécificité ludique et interactive. Dans un premier temps, il est intéressant de constater que le processus de création d'un jeu vidéo fait appel à des talents en rapport avec les Arts. Ainsi, les « concept-artists » vont produire des illustrations, par des méthodes généralement numériques, qui vont servir à représenter le but à atteindre et permettre d'imaginer les éléments du jeu avant sa construction. Les graphistes 3D étendent la sculpture au numérique tandis que les animateurs s'inspirent du spectacle vivant dans leur travail. La réalisation d'un jeu vidéo est

1 « La France compte 25,4 millions de gamers », 2009, http://www.afjv.com/press0910/091008_joueurs_en_france.htm.

ainsi un processus de création et d'expression artistique qui fait appel à de nombreuses disciplines issues des arts.

L'immersion du joueur dans l'œuvre interactive passe par le visuel et par le sonore. Nous allons nous concentrer sur le visuel, et particulièrement sur la représentation graphique d'un « monde » virtuel tridimensionnel qui va être perceptible par le joueur via une projection en perspective sur un écran bidimensionnel. Ce lien entre les données qui constituent le monde virtuel et l'image affichée sur l'écran doit pouvoir s'adapter au comportement et aux interactions avec le joueur. L'adaptation de l'affichage nécessite donc un traitement temps-réel de l'image.

Le « temps réel » décrit la possibilité de prendre en compte l'évolution du monde virtuel, soit par les simulations internes au programme soit par les actions du joueur, au fur et à mesure que les images s'affichent.

> l'interaction en temps réel est obtenue si l'utilisateur ne perçoit pas de décalage temporel (latence) entre son action sur l'environnement virtuel et la réponse sensorielle de ce dernier.[2]

Pour que l'image soit considérée comme « temps réel », il est donc important que le joueur puisse ressentir une quasi-simultanéité entre ses actions et la réponse visuelle et sonore produite par le programme. Il s'agit donc de jouer sur les limites de la perception humaine pour produire une image fluide et réactive. Entre le moment où le joueur influe sur le programme, par le biais d'une manette par exemple, et le moment ou l'image s'affiche, il se passe généralement 33 millisecondes. Ce délai nous permet de maintenir une cadence de 30 images par secondes. Dans ce temps très réduit, il faut mettre à jour notre simulation du monde virtuel, sélectionner les éléments qui seront visibles à l'image, et les afficher sur l'écran en calculant la couleur de chaque pixel. Les algorithmes utilisés pour la simulation et l'affichage de l'image doivent donc être très rapides. L'utilisation de matériels dédiés au graphisme temps réel est devenue la norme depuis plusieurs années, ce qui ajoute des contraintes sur la structure des algorithmes et leurs possibilités.

Les jeux vidéo présentent souvent des actions très rapides, que ce soit dans les jeux de simulation automobile ou dans les jeux de tir. Cette action très rapide n'est pas très lisible avec

2 Philippe Fuchs et al., *Le traité de la réalité virtuelle : Volume 2, Interfaçage, immersion et interaction en environnement virtuel* (Presses de l'Ecole des Mines, 2006).

une fréquence de 30 images par seconde. La norme dans ces jeux passe donc à 60 images par seconde, laissant seulement 16 millisecondes pour le calcul de chaque image.

Le temps réel est ainsi une grande spécificité de l'interactivité. Cette nécessité d'un retour immédiat de l'image n'existe pas dans le film d'animation par exemple.

Problématique

Cette étude est issue de la combinaison de mes recherches en programmation graphique, notamment en rendu expressif, et des recherches effectuées dans le cadre de l'entreprise de développement de jeux vidéo DONTNOD Entertainment, par le biais d'un contrat CIFRE. Les principales recherches chez DONTNOD ont portées sur la profondeur de champ, la matière et la réflexion, et les fluides. Ces trois sujets semblent au premier abord dédiés uniquement au rendu photoréaliste. Pourtant, nous allons voir que beaucoup de points communs existent entre ces sujets particuliers et le rendu dit « expressif ».

Le rendu expressif a pour but la prise en compte du regard personnel de l'artiste dans la représentation d'une scène. Ce type de rendu est très influencé par les techniques picturales et les œuvres des divers courants artistiques de la peinture, mais ne s'y limite pas. Le rendu expressif regroupe toutes les techniques de représentation d'une scène virtuelle qui se basent sur la subjectivité artistique plutôt que sur l'objectivité pure de la réalité.

Ainsi, notre étude va s'attacher à étudier le rendu photoréaliste et le rendu expressif afin d'en dégager les spécificités et les points communs.

Constats

Lorsque nous observons l'évolution de l'aspect graphique des œuvres interactives, nous pouvons constater l'existence d'une esthétique « par défaut » qui est associée à l'infographie 3D. Cette apparence standardisée est apparue à la suite des premières cartes graphiques, qui ne laissaient que très peu de choix dans le processus d'affichage. Ces limitations apportaient néanmoins un rendu très rapide de l'image.

Désormais, le matériel a grandement évolué et permet d'intervenir bien plus en profondeur dans le processus d'affichage de l'image. Ce travail de la programmation graphique permet de dépasser et de faire évoluer le rendu « par défaut » des œuvres interactives.

Nous avons également effectué un second constat. L'image qui est captée par une caméra peut sembler « réaliste » et neutre. Pourtant, la manipulation du dispositif technique cinématographique contribue à donner un véritable style visuel. Parmi ces manipulations, nous trouvons la création de nouvelles lentilles, l'adoption de nouveaux types de caméras ou encore le choix des pellicules et de la manière de les développer.

Une grande partie des techniques de la 3D cherchent à atteindre le réalisme cinématographique. Nous pouvons supposer ainsi que le même travail de dépassement des limites du médium qui est effectué pour le cinéma peut également être effectué dans le cadre du graphisme tridimensionnel.

L'évolution des technologies a également amené une amélioration des outils et donc une ouverture plus importante aux artistes non-techniciens. En effet, de nombreux outils permettent désormais d'intervenir en profondeur dans le processus de création de l'image temps réel sans nécessairement maîtriser la technique sous-jacente. Le rôle du programmeur de ce type d'outil est alors de comprendre les préoccupations de l'artiste et d'imaginer des algorithmes et des processus de production qui vont permettre à l'artiste de travailler l'image de la manière la plus intuitive et la plus complète.

Pour résumer, la capacité d'action du programmeur sur le processus graphique devient de plus en plus importante, ce qui permet d'étendre le champ des styles graphiques qui peuvent être utilisés. La recherche du photoréalisme n'est alors qu'un choix plastique, au même titre que les styles empruntés à la peinture.

Notre problématique première est donc de déterminer si la programmation de techniques innovantes rend possibles des choix plastiques inédits dans la création d'un espace virtuel tridimensionnel affiché en temps réel. Nous allons étudier comment le programme traduit une scène 3D virtuelle en une image bidimensionnelle compréhensible, convaincante et particulière. Les liens entre les avancées techniques et les possibilités de l'image temps réel seront également abordés. De nombreux styles d'images 3D naissent au moment de la découverte de nouvelles techniques.

Hypothèses

Notre thèse va donc se concentrer sur l'influence de la technique sur l'image 3D temps réel, et notamment sur le lien entre les techniques utilisées et le style graphique obtenu. Notre hypothèse principale est que la programmation de techniques personnelles porte une subjectivité et une intention esthétique et permet de travailler les caractéristiques visuelles de l'image d'un espace virtuel tridimensionnel affiché en temps réel.

- Chacun des éléments, chacun des détails qui finissent par constituer l'image peut être manipulé et mis en liaison avec les autres de toutes les manières possibles grâce à l'outil informatique, par programmation. Parmi ces choix se trouve notamment la simulation physique. Le choix des paramètres utilisés et de leurs influences visuelles est le résultat d'une décision qui révèle une intention esthétique.
- Grâce au degré de contrôle apporté par la programmation de techniques personnelles, nous pouvons faire émerger une cohérence globale dans l'image. Une fois toutes les caractéristiques internes à l'espace virtuel combinées, une image finale en mouvement est affichée sur l'écran. Tous ces choix, à propos de la lumière, de la matière, de la profondeur de champ, des fluides, influencent profondément l'ambiance globale de l'espace virtuel présenté et servent une intention artistique globale. Les techniques de programmation graphique sont ainsi des outils qui permettent de sculpter l'espace virtuel afin de lui procurer un aspect particulier, un comportement personnel.
- Les spécificités du temps réel interactif doivent être intégrées à cette recherche, car il s'agit d'appliquer procéduralement et instantanément un « style » graphique à tout un espace en fonction des actions du joueur. L'utilisation de techniques de programmation est alors nécessaire à cette adaptation fine du rendu graphique en fonction des caractéristiques de l'espace virtuel.
- Les styles d'images « photoréaliste » et « expressif » sont des cas particuliers du travail de l'image. Les techniques de création de l'image peuvent être les mêmes et doivent servir essentiellement à donner le contrôle aux artistes. Pouvoir personnaliser l'image jusque dans son rendu offre une identité visuelle indispensable à l'application interactive.

Les techniques de rendu développées ici vont servir à façonner une image 3D interactive. Elles doivent tenir compte de l'immersion et de la sensation d'interaction que nous souhaitons atteindre dans l'espace virtuel temps réel. Ainsi, ces techniques de rendu n'ont pas seulement une portée visuelle, mais concernent également les aspects immersifs et interactifs du médium. Cependant, nous aborderons très peu dans cette étude l'aspect interactif de l'image, qui se limitera pour nous au déplacement libre de la caméra dans l'univers virtuel. Ce point de vue nous permettra de nous concentrer sur la production de l'image comme représentante d'un univers stable et figé, sans la complexité que peuvent apporter les univers emplis d'actions des jeux vidéo. Ainsi, bien que l'image puisse bien évidemment être en mouvement, dans le cas des fluides ou du rendu expressif par exemple, l'univers reste pour nous essentiellement le même au cours du temps.

Le rôle de l'artiste, en tant qu'utilisateur des techniques de programmation mises en place, sera abordé dans la partie sur le rendu expressif, mais ne constitue pas le cœur de notre sujet.

Démarche

Cette étude vise à sortir de l'esthétique classique et standardisée des jeux vidéos et des applications de réalité virtuelle par l'apport de nouvelles techniques de calcul des éléments de l'image numérique temps réel. Certains effets sont déjà en grande partie maîtrisés et nous savons les reproduire avec une certaine efficacité alors que d'autres restent des défis constants, notamment en temps réel, et font l'objet d'innovations constantes ces dernières années. Nous nous concentrerons sur trois de ces sujets, notamment par l'introduction de nouvelles techniques. Nous montrerons ensuite que le rendu expressif est capable de les intégrer. Il peut ainsi profiter de leurs qualités plastiques et les détourner de leur but photoréaliste initial.

Nous commencerons par introduire le domaine du jeu vidéo et notamment ses différentes incarnations visuelles. Nous présenterons également plus en détail le contexte de création de cette étude ainsi que les contraintes et atouts de la 3D temps-réel.

Trois premiers axes seront ensuite abordés comme autant d'outils permettant à l'artiste de façonner l'image dite « photoréaliste ». Ces axes sont : le flou de profondeur de champ, les matières et réflexions et les fluides.

Le flou de profondeur de champ est un moyen efficace d'ajouter un degré de liberté dans l'image, celui du flou. C'est également une référence directe aux outils du cinéma, qui permettent de travailler la narration et l'esthétique de l'image dans sa profondeur. Nous montrerons que la prise en compte de la forme de la lentille de la caméra virtuelle va transformer l'aspect du flou de profondeur et donc de l'image. Nous introduirons notre propre technique de simulation d'une forme de lentille hexagonale pour une application en temps réel, et effectuerons la comparaison avec diverses techniques récentes du domaine.

Le travail de la matière est un outil extrêmement utile pour façonner à la fois des surfaces qui ont du caractère, mais aussi leurs réponses face aux changements d'environnement. La réponse à la lumière est la principale caractéristique d'une matière. Nous montrerons que le réalisme de la simulation de la matière n'a pas uniquement vocation à la création d'une image fidèle à la réalité, mais sert également à améliorer le contrôle par l'artiste. Avoir un modèle complexe de reproduction de la matière permet de donner un caractère spécifique à une surface. Parmi les types de réponse à la lumière, la spéculaire ambiante est la plus complexe à mettre en œuvre en temps réel aujourd'hui. Il s'agit de la prise en compte d'un environnement lumineux complexe dans la reproduction de la réflexion lumineuse. Cette réflexion doit pouvoir s'adapter aux caractéristiques du matériau telles que la rugosité. Nous introduirons de nouvelles techniques de prise en compte des échantillons lumineux et du calcul de la réflexion spéculaire ambiante. Nous nous intéresserons notamment à la reproduction du phénomène de parallaxe dans la réflexion et à la création efficace d'une réflexion floue.

Les fluides permettent de reproduire les phénomènes de liquides et de gaz, mais présentent également un intérêt plus étendu. Il s'agit d'appréhender l'image numérique et son mouvement par un autre biais que celui de la modélisation polygonale. La prise en compte des équations des fluides nous permet d'obtenir une image toujours en mouvement, qui répond aux changements dans le monde virtuel et aux interactions du joueur. Ce comportement parait tout à la fois familier et étonnant aux joueurs. Nous nous intéresserons particulièrement à la reproduction de surfaces d'eau semi-planes et au problème de l'advection de textures. Il s'agit d'améliorer l'aspect visuel du fluide en couplant la simulation physique avec la mise en mouvement d'une texture. Cette texture va se déplacer en fonction du courant, comme si elle était déposée à la surface de l'eau. L'apport de l'advection de textures, au-delà de

l'amélioration visuelle, permet de personnaliser aisément l'aspect du fluide par la modification d'une texture, qui sera répétée et mise en mouvement par le fluide.

Pour finir, le rendu expressif servira de lien final entre les trois sujets précédant tout en apportant de nouveaux outils de personnalisation de l'image. Comme nous l'avons déjà mentionné, le rendu expressif cherche avant tout à prendre en compte le regard personnel de l'artiste plutôt que d'effectuer une reproduction fidèle de la réalité. Les outils présentés dans les trois parties précédentes sont autant de voies qui peuvent à la fois reproduire la réalité, mais aussi la détourner en permettant à l'artiste de les contrôler à volonté. La profondeur de champ et les fluides seront ainsi abordés à travers la perspective du rendu expressif. Le rendu expressif regroupe également les techniques de reproduction des outils traditionnels associés à la peinture et au dessin. Nous introduirons de nouvelles techniques pour créer des images de ce type à partir des données classiques habituellement destinées à la création d'une image photoréaliste. Nous chercherons à améliorer le travail de la matière par le moyen d'un rendu hachuré. Pour finir, nous travaillerons sur le sujet du contour dessiné, outil primordial si nous souhaitons nous inspirer de l'esthétique du croquis dessiné.

Notre travail a donc un double objectif. Il se propose d'une part de contribuer à la compréhension et à l'amélioration des quatre sujets précités pris individuellement. D'autre part, il vise à apporter une réflexion sur l'expressivité de l'image et sur la stylisation de l'image 3D temps réel par le biais des techniques programmées.

Chapitre I - Contexte

1 Le jeu vidéo

1.1 L'évolution des styles graphiques

Différents styles graphiques se sont développés au cours de la jeune histoire du jeu vidéo, en même temps que l'évolution du rendu dans l'image précalculée, que ce soit pour du film d'animation « full 3D » ou pour les effets spéciaux des films classiques.

Au début, le style adopté était grandement imposé par les limitations des machines de calculs que ce soit du côté de l'espace mémoire disponible ou bien de la vitesse de calcul. Ces contraintes empêchaient complètement d'obtenir un rendu réaliste et forçaient donc les créateurs d'effets spéciaux à styliser le plus possible les graphismes, à les simplifier afin qu'ils puissent être calculés par la machine.

Le film « Tron » (1982) par exemple, le premier film utilisant des séquences entièrement réalisées numériquement, se décalait totalement de la réalité et proposait un design graphique très coloré, à base d'aplats de couleurs vives, afin de ne pas subir la comparaison avec l'image filmée.

Figure I.1: « Final Fantasy : The Spirit Within » (2001), « La Légende de Beowulf » (2007) et « Là haut » (2009)

Par la suite, la puissance des machines augmentant, l'intérêt des créateurs et du public se focalisa sur toujours plus de réalisme. Les films d'animation ont fini par atteindre un apogée avec le film « Final Fantasy : The Spirit Within » (2001), qui fut une déception, malgré les

talents et les moyens impressionnants mis au service du film. Ce film d'animation appelle, de par son scénario et son traitement graphique, à une comparaison frontale avec le film traditionnel, par rapport auquel il perd beaucoup en émotion et en empathie avec les personnages. Les films d'animation qui sortent maintenant trouvent leur voie dans un léger décalage vers un rendu plus « pâte à modeler », avec une gestion réaliste de la lumière et des matières, mais toujours une stylisation, une caricature. C'est le cas des films de Pixar tel que « Là haut » (2009). Le film « La Légende de Beowulf » (2007) présente une certaine caricature, malgré des graphismes très détaillés et crédibles. Certains dessins animés, japonais notamment, montrent des designs novateurs et particuliers, et cherchent généralement à entremêler les techniques 2D et 3D tout en gardant un rendu cohérent. Pour cela, les techniques utilisées sont mixtes, très dépendantes de chaque plan.

Figure I.2: L'influence des techniques 3D dans le film d'animation japonais. À gauche, « Ghost in the Shell 2 : Innocence » (2004) et à droite « Paprika » (2006),

Le jeu vidéo a gardé bien plus longtemps ces contraintes de puissance de la machine, et nous ne sommes pas encore sortis complètement du jugement basé sur la puissance d'un moteur 3D ou d'une console.

Nous pouvons remarquer le même processus que pour le rendu précalculé : les premiers jeux vidéos étaient très abstraits, avec un rendu très éloigné de la réalité, puis avec le temps, les jeux sont devenus de plus en plus figuratifs et photoréalistes.

Ainsi de nombreux jeux du début de l'histoire du jeu vidéo peuvent être qualifiés d'abstraits. Asteroids (1979), Pong (1972), le premier jeu Star Wars en 3D (1983), bref tous les anciens jeux à base de graphismes vectoriels possèdent un aspect non réaliste qui leur donne un certain charme.

Malheureusement, l'argument du réalisme permet de vendre des consoles de jeux et de nouvelles machines, et il est donc très soutenu par les constructeurs, et les arguments sont faciles à trouver pour séduire le public.

Nous ne parlerons pas beaucoup ici des jeux en deux dimensions, dont le style graphique du rendu n'est lié en général qu'à la manière de dessiner les « sprites », c'est a dire les éléments qui constitueront décors et personnages. C'est un processus purement artistique, qui ne donne pas beaucoup de place à l'évolution technique. Néanmoins, la plupart des jeux en deux dimensions du début du jeu vidéo utilisaient des graphismes dessinés à la main, la synthèse d'image étant encore très balbutiante. Seuls les jeux à base de graphismes vectoriels peuvent être qualifiés de réellement synthétiques. Ainsi la majorité des jeux dont les composants sont dessinés à la main pourraient être qualifiés de non-réalistes. De plus, les capacités limitées des consoles de l'époque rendaient le choix des couleurs limité, ce qui déformait la réalité, et la faible résolution forçait une certaine caricature. Mais la majorité des jeux de cette époque cherchent le réalisme, sans y parvenir.

Nous parlerons donc essentiellement des jeux qui cherchent à représenter une scène 3D virtuelle, avec ses décors et personnages, sur un écran 2D.

Les premiers jeux 3D suivent totalement l'idée du photoréalisme et la troisième dimension n'est qu'un moyen de plus pour l'atteindre. Ainsi « Wolfenstein 3D » (1992), « Doom » (1993) et finalement une apothéose de l'époque : « Tomb Raider » (1996), poussent le graphisme le plus possible dans le réalisme et dans l'immersion. En parallèle de ce phénomène, nous trouvons le genre du jeu de plate-forme destiné aux enfants, avec les jeux Nintendo tels que « Super Mario 64 » (1996) et « The Legend of Zelda : The Wind Waker » (2002). Ils sont les premiers à se démarquer de la recherche du photoréalisme en s'inspirant plus de l'atmosphère des dessins animés.

Figure I.3: « Super Mario 64 » (1996), « The Legend of Zelda : The Wind Waker » (2002) et « Tomb Raider » (1996)

Nous avons donc continuellement cette opposition : réalisme dans les jeux pour adultes et adolescents, dessin animé pour les jeux destinés aux enfants. Ces deux styles sont alors pratiquement les seuls viables en terme de marketing, car le seul public qui était visé il y a

encore quelques années était constitué des hardcores gamers (passionnés de jeux vidéo), et des plus jeunes, les enfants. Globalement, un public masculin et enfant ou adolescent.

Le monde du jeu vidéo a fini par s'ouvrir à un nouveau public, à s'adresser même spécifiquement au grand public et à se diversifier : les adultes, les femmes, les personnes âgées. L'esthétique habituelle du jeu vidéo ne fonctionnait plus vraiment avec ce public, peu sensible aux changements apportés par une simple évolution graphique. Le gameplay des jeux a également changé et donc l'objectif du jeu, ce qui influe sur l'esthétique qu'il doit dégager. Le jeu vidéo n'est plus simplement un processus immersif, cherchant à simuler au maximum la réalité, mais aussi un moyen de communiquer, un outil (Programme d'Entrainement Cérébral du Dr Kawashima (2005)), un animal de compagnie (Nintendogs (2005))...

La Nintendo Wii (2006) est l'exemple le plus marquant de ce changement. Cette console est à peine plus puissante que la précédente (la Nintendo Game Cube (2001)), mais concentre ses arguments sur de nouvelles manières de jouer, et vise un public très diversifié. Le nombre de consoles Wii vendue a rapidement dépassé ceux des autres constructeurs, Microsoft Xbox 360 (2005) et Sony PlayStation 3 (2006), qui conservent essentiellement le même discours centré sur l'évolution technique et la puissance de la machine.

1.2 Les styles graphiques dans les jeux vidéo actuels :

Nous allons effectuer une classification des jeux vidéo actuels en fonction de la stylisation du rendu qu'ils adoptent. Cette classification est essentiellement arbitraire, mais va nous permettre d'introduire les différentes directions artistiques empruntées par les développeurs dans le travail graphique.

1.2.a Rendu cartoon simple

Tout d'abord, les jeux au rendu « cartoon » très simple. Ils s'adressent généralement aux enfants ou présentent des atmosphères enfantines, et sont les héritiers directs de l'esthétique dessin animé. Nous pensons aux jeux Mario en 3D (64, Sunshine et Galaxy) ou encore à « Zelda Wind Walker » dont nous avons déjà parlé. Les jeux « Jet Set Radio Future » (2002), « One Piece : Grand Adventure » (2006) et « Eternal Sonata » (2007) utilisent également ce type de graphismes. Les graphismes de ces jeux sont généralement simples et cherchent la lisibilité et la « ligne claire » avant tout.

Figure I.4: "Jet Set Radio: Future » (2002), « One Piece : Grand Adventure » (2006) et « Eternal Sonata » (2007)

1.2.b Rendu dessiné, peint ou « pâte à modeler »

La seconde catégorie regroupe les jeux qui utilisent un rendu expressif, en cherchant à atteindre une certaine complexité dans le rendu. Cette complexité peut provenir de l'utilisation de hachures, ou bien d'un modèle d'éclairage avancé, ou encore de l'utilisation de particules.

De nombreux jeux de cette catégorie font partie de stéréotypes de jeux qui utilisaient habituellement un style graphique photoréaliste, mais qui sont maintenant passés à un rendu plus expressif. « Team fortress II » (2007) est un exemple révélateur, car il fait suite à un premier épisode photoréaliste et à lui-même tout d'abord été présenté ainsi. Il a subi ensuite un changement d'orientation total et se rapproche maintenant visuellement beaucoup du film de Pixar, « Les Indestructibles » (2004). Les personnages sont caricaturaux, le rendu ressemble à des miniatures en pâte à modeler. Le moteur utilisé est pourtant celui de « Half Life II » (2004), un moteur très réaliste. Il permet ainsi d'obtenir un rendu complexe, évolué, avec des calculs de lumière très poussés, tout en gardant un rendu expressif.

Le « Prince Of Persia » de 2008 créé la surprise. Même si les 3 récents épisodes adoptaient un rendu assez stylisé, au niveau des couleurs et des ambiances, ce nouvel opus change radicalement et utilise des effets de contours autour des personnages et des textures très graphiques. Le jeu massivement multijoueur indépendant « Love »[3] propose un rendu graphique à base de particules, chacune comme un coup de pinceau qui vient dessiner un univers onirique.

Le jeu « Okami » (2006) est un très bel exemple de rendu expressif. Il reprend l'aspect des estampes japonaises (courant « Ukiyo-e »), un univers qui n'avait jamais été exploité en trois dimensions.

Le projet suivant de l'équipe à l'origine de « Okami », le studio Clover, est cette fois pour les adultes. Il se nomme « Madworld » (2009) et présente un rendu totalement en noir, blanc et

3 « Quel Solaar : Love », s. d., http://www.quelsolaar.com/.

rouge. L'atmosphère visuelle se rapproche du film Renaissance, avec un contraste noir/blanc énorme. La grande violence du titre est permise justement grâce au rendu non réaliste. Les jeux « Killer 7 » (2005) et « No More Heroes » (2007) sont des jeux violents pour les adultes mais qui utilisent pourtant un rendu « cel shading », montrant une certaine évolution dans le choix du style graphique.

Figure I.5: "Prince of Persia" (2008), « No more heroes » (2007) et « MadWorld » (2009)

1.2.c Les jeux dit « photoréalistes »

La dernière catégorie cherche à respecter au maximum la réalité perçue par les appareils photographiques. Pour cela, certains développeurs utilisent directement des captures de la réalité par le biais de l'appareil photographique pour réaliser les textures de l'environnement. C'est le cas de beaucoup de simulations et de « serious games », comme par exemple le jeu de simulation de pilotage « Microsoft Flight Simulator X » (2006).

Cependant, même parmi ces jeux considérés comme photoréalistes, tel que « Crysis » (2007) ou « Uncharted 3 » (2011), une certaine stylisation du rendu est adoptée. Le choix de l'emplacement où se déroule l'action dans le jeu influence déjà l'ambiance graphique qui se dégage. « Crysis » se déroule essentiellement dans des jungles alors que « Uncharted 3 » met en scène des décors désertiques. Le rendu est très contrasté, privilégiant l'immersion sans chercher à être strictement fidèle à la réalité, mais plutôt à proposer une image percutante et crédible. Les développeurs souhaitent nous plonger dans une ambiance forte et originale. L'image ressemble à une reproduction de la réalité, mais nous pouvons sentir le travail sur les couleurs, la lumière et les matières. Nous nous rapprochons de l'influence d'un directeur de la photographie sur un long métrage filmé. D'autre jeux, comme « Ico » (2001) et « Shadow Of The Colossus » (2005), utilisent des couleurs, une saturation, certains effets qui ne sont pas

fidèles à la réalité et contribuent à donner une ambiance légèrement décalée par rapport à la réalité.

Figure I.6: « Microsoft Flight Simulator X » (2006) en haut à gauche, « Crysis » (2007) en bas à gauche et « Shadow of The Colossus » (2005) à droite.

2 Contexte de réalisation

2.1 L'entreprise DONTNOD Entertainment

Les recherches qui sont présentées dans cette étude ont été développées dans le cadre d'un contrat CIFRE avec l'entreprise parisienne de développement de jeux vidéo DONTNOD Entertainment. Mon travail dans l'équipe de recherche et développement m'a permis d'adapter mes idées en terme d'algorithmes mais aussi ma vision du processus technique de création d'une image interactive temps réel. Il m'a également apporté une vision plus précise du processus de développement d'un jeu vidéo, ce qui a profondément fait évoluer mes idées, notamment au niveau de l'importance des outils dédiés aux artistes.

La société DONTNOD travaille sur un jeu vidéo pour consoles de dernière génération (PS3 et Xbox 360). Nous sommes environ 70 personnes à travailler sur le projet, dont 13 développeurs. Il s'agit d'un jeu d'action-aventure futuriste dont le nom de travail est « ADRift ». Il sortira en 2013 sous le nom de « Remember me » et sera édité par Capcom. Les déplacements du personnage sont visualisés à la troisième personne, avec une caméra qui suit l'action. Le jeu est centré autour de l'exploration des décors, de la narration et du combat.

Le travail dans l'équipe R&D regroupe plusieurs tâches différentes. Tout d'abord, la veille technologique est indispensable, afin de comprendre l'évolution technique et graphique des jeux vidéo. La recherche pure permet d'approfondir les différents points clés du rendu graphique. Il s'agit également de confronter son intuition aux diverses techniques existantes pour trouver ce qui fait la force de chacune. La création de techniques innovantes devient alors possible en cherchant à combiner ces différentes forces. Un dernier travail consiste à implémenter concrètement l'algorithme dans le moteur de jeu utilisé, à l'optimiser au maximum et surtout à le rendre utilisable par les artistes. Ils doivent pouvoir manipuler les paramètres de l'effet. Ils peuvent souvent appliquer l'effet à des utilisations imprévues. Le

dialogue qui s'instaure entre développeur et artiste devient alors très riche et améliore grandement le résultat obtenu.

2.2 Contraintes de la 3D temps réel

Le temps réel possède de nombreuses limitations, dont nous avons déjà un peu parlé. Contrairement au rendu précalculé, utilisé pour les films d'animations par exemple, nous disposons de très peu de temps pour effectuer tous les calculs liés à l'affichage de l'image. Bien sûr, en pratique, le rendu précalculé est également limité dans le temps qui peut être consacré au calcul d'une image. Ainsi, Dylan Sisson, qui travaille sur RenderMan, avait évoqué au cours d'une conférence[4], une étude interne qui avait été faite sur la durée moyenne du calcul d'une image dans les films d'animation de Pixar. Cette étude avait montré que le temps de rendu n'évoluait pas vraiment, malgré les avancées technologiques matérielles et logiciels, pour rester aux alentours d'une heure pour l'image finale. Ce temps est une sorte d'équilibre entre la qualité d'image qui augmente à chaque projet et les contraintes matérielles et humaines. En effet, il est nécessaire de pouvoir obtenir un aperçu de l'image définitive régulièrement au cours de la création d'un film. Le calcul final doit également pouvoir être effectué suffisamment vite pour respecter la date de sortie du film.

Par contraste, la durée de 33 millisecondes du temps réel paraît très limitée. Et pourtant, la qualité de certaines scènes est tout à fait à la hauteur des rendus précalculés qui datent de quelques années. L'évolution dans le domaine du temps réel s'est effectuée très vite, à la fois en adaptant des techniques issues du rendu précalculé, mais aussi en innovant, avec par exemple l'introduction des textures de normales, qui est devenu très populaire dans le rendu temps réel avant de devenir un outil pour le précalculé.

Pour diminuer les temps de calcul, nous nous appuyons en grande partie sur du matériel particulier, qui impose ses propres contraintes. Ainsi, chaque plateforme à ses limites en terme de mémoire et de puissance de calcul CPU (processeur principal) et GPU (processeur graphique). Les approches efficaces peuvent ainsi varier selon le contexte. Les adaptations à ces différentes plateformes et contextes se font dorénavant par le biais de « middlewares », qui sont des logiciels intermédiaires qui facilitent la création de jeux. Au lieu de programmer toute la gestion du jeu, des animations à l'affichage en passant par l'intelligence artificielle, nous pouvons ainsi nous concentrer sur la production associée directement à un jeu précis, et

4 Dylan Sisson, « Soirée PIXAR RenderMan », novembre 10, 2009.

utiliser les « briques » de constructions qui sont apportées par un ou plusieurs logiciels intermédiaires. Par contre, si cela nous permet d'aller beaucoup plus vite dans la création concrète du jeu, c'est également une perte de contrôle et une limitation sur certains aspects. Les effets programmés dans le jeu doivent en effet être en adéquation avec la structure du moteur 3D. Il est parfois difficile d'utiliser certaines techniques ou de les optimiser, car elles n'ont pas été envisagées lors de la conception du moteur.

2.3 Atouts de la 3D temps réel

Malgré ces contraintes, le rendu temps réel propose plusieurs avantages par rapport au rendu précalculé.

L'avantage le plus évident est le caractère immédiat des retouches apportées au jeu. Il est ainsi très facile de procéder par itérations rapides en améliorant le rendu d'une scène tout en ayant le résultat final sous les yeux. Au contraire, le rendu précalculé impose souvent une longue étape de calcul avant d'obtenir une image réellement représentative. Le rendu temps réel améliore ainsi l'expérience de création des artistes, qui peuvent observer le résultat de leur travail immédiatement.

Un second atout, qui peut paraître contradictoire, concerne la prise en compte de l'interactivité et de la liberté du point de vue. Cet atout est en effet une limite sur les techniques utilisées. Dans le rendu précalculé, tel que le film d'animation, il est souvent tentant de « tricher » sur certains plans, en travaillant l'image telle que vue depuis la caméra, plutôt que comme une scène 3D complète. Le rendu temps réel ne permet pas ce type de solution dès que le joueur a le contrôle de la caméra. Nous sommes donc tout naturellement dirigés vers des techniques procédurales, qui peuvent s'adapter à la position de la caméra et aux autres interactions. Pour nous, c'est un avantage important, car elle met en avant une approche par la programmation de techniques qui, nous le pensons, a le potentiel d'améliorer la qualité de l'image et son adéquation avec la volonté artistique préalable.

Ces atouts ont d'ailleurs été compris par le milieu du rendu précalculé, qui commence à utiliser des technologies issues du jeu vidéo. Ainsi, la prévisualisation de nombreux effets, tels que la lumière et les fluides, utilisent une technologie aux temps « interactifs » de quelques images par secondes. Pour aller encore plus loin, certains longs-métrages d'animation ont été réalisés avec des moteurs de rendu temps réel, et la tendance se vérifie encore plus pour la série d'animation. En effet, avoir un retour immédiat des modifications et pouvoir rendre un

film entier en quelques heures sont des atouts précieux pour les milieux en « flux tendus ». Généralement, ces moteurs nécessitent des machines plus puissantes que la moyenne et ajoutent quelques calculs plus longs sur le rendu final, par exemple un flou de profondeur de champ de grande qualité et un anticrénelage.

2.4 Outils et logiciels

Lors de la réalisation de cette recherche, nous avons utilisé divers outils, principalement pour l'écriture de programmes et de « shaders ».

- Unreal Engine 3 : Le jeu vidéo sur lequel nous avons travaillé chez DONTNOD utilise le moteur multiplateforme « Unreal Engine 3 » développé par « Epic Games ». Ce moteur propose une gestion poussée du « gameplay », des animations et du graphisme. Le moteur peut être complètement remanié grâce aux sources dont nous disposons. Les consoles PS3 et Xbox 360 sont supportées. Cet outil à un très grand intérêt de par son éditeur, qui permet de transférer une grande partie des décisions et de l'expérimentation vers les artistes. Ainsi, au lieu de devoir monopoliser en permanence des programmeurs pour implémenter les idées des graphistes, animateurs ou game-designer, l'utilisation de l'éditeur permet de faire évoluer rapidement le contenu du jeu. La base du système et les évolutions majeures nécessitent toujours le travail des programmeurs. Cet éditeur va clairement dans le sens de donner la liberté créative à l'artiste. L'Unreal Engine 3 est disponible dans une version gratuite grâce à l'UDK (Unreal Development Kit), qui permet d'utiliser toute la puissance de l'éditeur, sans toutefois autoriser les interventions dans le code C++, réservées aux entreprises payant la licence complète.
- Visual C++ et DirectX SDK : La combinaison de l'écriture d'un programme en langage C++ et de l'utilisation d'une librairie graphique comme DirectX (ici dans sa version 9 généralement) procure le maximum de contrôle. Le programme peut ainsi effectuer des calculs sur le processeur principal, ou bien sur le processeur graphique, et gérer finement les transferts d'informations entre les deux. Cette solution est idéale pour la réalisation d'un produit fini, qui doit proposer les meilleures performances et doit pouvoir s'adapter aux contraintes de production (différents décors, gameplay…). Pour la création de prototypes, cette solution est un peu longue à mettre en place et

nécessite des compilations longues et fréquentes. Cette solution est totalement gratuite grâce à la version « express » de Visual C++.

- Visual C# et XNA : Le XNA est un moteur minimaliste gratuit proposé par Microsoft, permettant de développer dans le langage C# pour les plates-formes Windows ou Xbox 360. C'est une solution très pratique pour le prototypage d'effets, puisqu'elle fournit directement le support des « pixel shader » et « vertex shader », ainsi que le chargement de textures et de maillages 3D. Malheureusement, le support des fonctions de DirectX n'est pas complet, certaines sont manquantes, ce qui empêche le développement de certains algorithmes. C'est néanmoins la solution que nous avons utilisé pour beaucoup de prototypes.
- Virtools : Le développement de prototypes est très simple avec Virtools, qui fournit à la fois un langage visuel très simple pour l'interactivité et le support des « shaders ». C'est la solution que nous utilisions au début de la recherche, mais que nous avons laissée de côté en raison de sa politique tarifaire.
- Render Monkey : Ce logiciel de AMD est dédié à l'écriture d'effets et se révèle très pratique pour tester rapidement un algorithme. Les paramètres peuvent être manipulés par des « glissières » et différentes « passes » peuvent être ajoutées. Néanmoins, le logiciel est surtout adapté au travail sur un seul objet et beaucoup moins sur une scène complexe. Ce logiciel a été abandonné par AMD et ne sera donc plus mis à jour.
- Shader Toy[5] : Cet outil est disponible directement en ligne, permettant de tester ses « shaders » sans installer de logiciels spécifiques. C'est une démonstration des capacités de WebGL, qui permet d'utiliser la librairie OpenGL dans un navigateur internet. Le navigateur utilisé doit être suffisamment récent. « Shader Toy » est dédié à la création de « shaders » bidimensionnels dessinés directement sur l'écran et autorise l'utilisation d'images et d'une interactivité limitée.

5 Iñigo Quilez, « Shader Toy », 2009, http://www.iquilezles.org/apps/shadertoy/.

Chapitre II - Le flou de profondeur de champ

1 Le rôle du flou de profondeur de champ

1.1 Le rôle du flou en cinéma, photographie et jeux vidéo

Considéré au départ comme un défaut technique qui doit être gommé et réduit au minimum, le flou dans l'image est devenu incontournable au fur et à mesure du développement du langage photographique et cinématographique. Le flou dans une image filmée est généralement obtenu en manipulant la focale et l'ouverture du diaphragme d'un objectif pour créer des zones plus ou moins floues en fonction de la distance entre le sujet et la caméra. C'est le flou de profondeur.

1.1.a Le flou comme indicateur spatial

Le degré de flou dans une image est un indice important de la profondeur relative des éléments de l'image. C'est un outil essentiel dans la perception tridimensionnelle des images planes. Le rôle premier du flou de profondeur dans une image, capturée du réel ou calculée dans le virtuel, est d'organiser les plans en différentes profondeurs de flou en respectant la spatialité de la scène. Ainsi, l'œil du spectateur, placé devant l'image, utilise ces informations de flou pour reconstruire les positions relatives des différents plans, et donc une notion d'échelle entre les éléments. Ces indices spatiaux peuvent bien entendu être manipulés afin de suggérer de fausses informations au spectateur. Par exemple, l'utilisation d'un flou de profondeur très fort en arrière-plan et en avant plan créé une image

Figure II.1: La profondeur de champ très réduite suggère une scène miniature.
Photographie Greg Keene

évoquant les vues macroscopiques. La scène paraît minuscule. C'est l'effet dit « Tilt shift » qui est obtenu en photographie par l'utilisation d'objectifs spécifiques.

Figure II.2: Exemples du flou de profondeur en photographie.

1. *La structure géométrique liée au flou de profondeur créée une image presque abstraite.*
2. *Le flou sur l'arrière-plan permet de concentrer l'attention sur le portrait.*
3. *Le fond participe à la fois à l'ambiance visuelle de l'image et à sa compréhension.*

Photographies JWCreations, Daniel Flather et Carlos Luis

1.1.b Le flou comme élément structurant de l'image

Le second grand rôle du flou de profondeur est la simplification de l'image. Le flou permet de diriger le regard vers les éléments jugés importants sans perturber l'œil du spectateur avec des détails inutiles présents dans toute l'image. Ainsi, les portraits de personnages ont très souvent un fond flouté afin que toute l'attention soit portée sur les personnages. Ceux-ci apparaissent alors séparés du fond et l'image est plus lisible. Il est possible d'opérer à l'inverse et de flouter le premier plan lorsque c'est le fond qui nous intéresse, en créant par exemple un cadre flou avec les parties en avant-plan de l'image. Le flou peut également servir à lier un élément à son fond, lorsque la zone de transition du net au flou est visible, sur le personnage d'un portrait par exemple. La forme tridimensionnelle de l'objet est ainsi évoquée tout en gommant la séparation nette avec l'environnement. Le flou peut également être un élément de structure temporel qui peut se substituer à un découpage d'une scène en plusieurs plans. Les changements de profondeur de champ peuvent découper une action en plusieurs parties sans pour autant modifier la position de la caméra. Le flou sert d'élément structurant spatial et temporel pour mettre en avant certaines parties de l'image à certains moments.

1.1.c Le flou comme élément d'expression esthétique

Le troisième grand rôle du flou de profondeur concerne plus l'esthétique et les idées que l'auteur de l'image souhaite nous transmettre. Les effets de lentilles dont nous parlerons plus tard sont très utiles pour donner une dimension esthétique, un caractère à un décor autrement

sans réel intérêt. Les formes abstraites de la lentille utilisée (ronde, hexagonale, ellipsoïdale…) vont réagir avec les zones fortement lumineuses de l'image pour créer un jeu fascinant de formes et de couleurs. Les lumières ponctuelles en grand nombre sont particulièrement adaptées à cet exercice, car chacune d'elle va créer sa propre forme lumineuse. Le résultat final est une image abstraite qui relève de principes simples d'optique, mais dépend de l'interaction entre la scène d'origine et les caractéristiques du dispositif de captation.

Figure II.3: Utilisation du flou de profondeur comme élément esthétique. Détail issu du film "The Go Master" (2006).
Le paysage au fond devient pratiquement abstrait à travers la lentille de la caméra.

La notion de flou peut également s'étendre à d'autres effets qui cherchent à rendre l'image moins lisible. Ainsi, les effets de bruits sur le signal de l'image, ou bien les taches sur l'objectif, les effets de diffractions de lentilles qui aveuglent une partie de l'image.

Tous ces effets peuvent participer à masquer les détails d'une partie de l'image, nous les considérerons donc comme des sous-ensembles de la notion de flou.

Figure II.4: Effets de lentilles dans le film « Star Trek » de J.J.Abrams (2009)

1.1.d Le flou comme outil narratif et expressif

En ce qui concerne l'utilisation symbolique et narrative du flou, de nombreux plans de grands cinéastes doivent être analysés finement pour espérer décrypter le rôle du flou ainsi que sa combinaison avec les autres éléments de l'image : cadrage, éclairage, mouvement… Les films du cinéma expérimental, avec par exemple Stan Brakhage, ont explorés amplement les possibilités d'utilisation du flou de caméra. Wees à étudié l'esthétique du cinéma expérimental dans un livre intitulé « Light Moving in Time »[6].

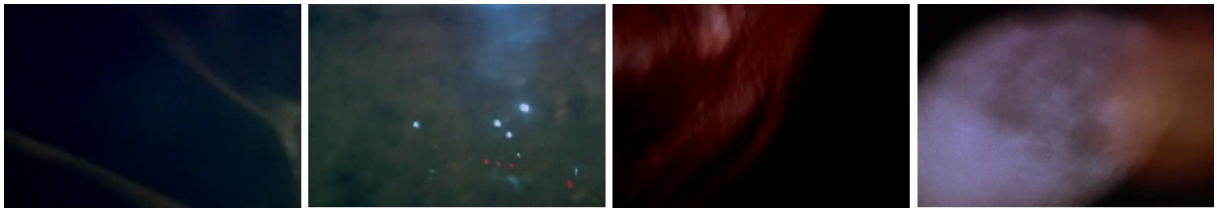


Figure II.5: Quatre images extraites de « Prelude: Dog Star Man » de Stan Brakhage (1961) illustrant une étude du flou et de la surimposition de plusieurs images.

Une utilisation classique dans le film d'horreur est de suggérer par le flou une peur qu'une simple image nette ne saurait montrer. Le fantasme du spectateur peut alors pleinement remplir le vide de détail laissé par le flou. L'état mental d'un personnage peut lui aussi se refléter dans la qualité de l'image (pixellisation, grain, netteté…), pour suggérer par exemple les états de folie ou de confusion.

Ces deux rôles sont également très utilisés dans le jeu vidéo, par exemple dans le jeu « Amnesia, The dark descent » où l'image devient floue à la moindre manifestation surnaturelle dans le jeu, afin de suggérer l'état émotionnel et psychologique dans lequel se trouve le personnage que le joueur incarne.

Figure II.6: Limbo (PlayDead - 2010) et Amnesia : The Dark Descent (Frictional Games - 2010)

6 William C. Wees, *Light Moving in Time: Studies in the Visual Aesthetics of Avant-Garde Film* (University of California Press, 1992).

Dans le jeu vidéo « Limbo », tout le décor en arrière-plan est plongé dans un flou qui sert à séparer arrière-plan et avant-plan. L'arrière-plan est présent uniquement comme décor alors que le premier plan contient des éléments utiles pour le joueur. Le flou nous renseigne ainsi également sur l'importance des éléments du point de vue du « gameplay ». Les éléments flous, qui se fondent dans le décor, ne nous seront pas utiles pour avancer dans le jeu. Le flou dans Limbo a également un rôle très important dans l'aspect visuel singulier et l'ambiance morbide et mystérieuse. Plus subtilement, flouter un personnage dans certains plans d'un film peut nous inciter, en tant que spectateur, à penser qu'il n'est pas ce qu'il prétend être, qu'il ment et que nous devons voir au-delà des apparences. Le contraste entre les moments utilisant beaucoup le flou et ceux présentant une grande netteté peuvent également faire passer des messages au cours du temps plutôt que sur une seule image. Dans le jeu « Silent Hill 2 », la qualité de l'image se dégrade au fur et à mesure que le joueur avance dans l'histoire, ce qui peut prendre plusieurs heures, puis suit les apparitions fantastiques. Au départ, l'image est nette, puis elle devient bruitée, sale. Au cinéma, le flou est bel et bien un outil narratif important que le réalisateur utilise pour exprimer ses idées. Cette utilisation se retrouve également dans le jeu vidéo, pour servir le même type de rôle.

1.1.e Le flou comme tradition cinématographique

Nous avons vu que le flou de profondeur pouvait être utilisé dans les applications temps réel pour de multiples raisons esthétiques ou narratives. Malheureusement, dans la plupart des cas il est utilisé uniquement dans le but de se rapprocher du cinéma et particulièrement de l’esthétique du blockbuster hollywoodien. L'utilisation de l'effet tient plus d'une référence culturelle que d'un véritable choix esthétique. Les cinématiques temps réel dans les jeux perdent effectivement beaucoup en crédibilité si le flou de profondeur est absent ou bien s'il manque de qualité. Le spectateur est habitué à ce que l'effet de flou de profondeur accompagne les moments d'inactivités, lorsqu'il s'agit uniquement d'assister passivement au déroulement de l'action, exactement comme dans un film. La recherche d'une esthétique cinématographique est un frein au plein développement des possibilités permises par un flou de profondeur numérique. Cet effet fait partie de l'esthétique « standard » que s'attend à retrouver le joueur, et toute déviation du stéréotype semble être un risque que les développeurs de jeux ne sont pas prêts à prendre. L'utilisation de la profondeur de champ dans les jeux vidéo a pourtant un potentiel qui va bien au-delà de la réplique fidèle de la caméra.

1.2 Le rôle perturbateur du flou

Le flou peut être travaillé par le créateur d'une œuvre pour de nombreuses raisons, telles que celles que nous avons présentées ci-dessus. Néanmoins, si l'utilisation du flou n'est pas suffisamment maîtrisée, il devient nocif pour l'expérience de l'utilisateur, particulièrement dans les applications interactives.

1.2.a La liberté du regard

En effet, c'est la notion de liberté du regard qui est remise en cause lorsque nous utilisons le flou. Dans la vie de tous les jours, nous sommes habitués à pouvoir faire le point sur les objets qui nous intéressent. Cela nous permet de faire le même type d'opération que fait le réalisateur lorsqu'il choisit la composition d'une image. Nous pouvons faire varier le cadrage, isoler les éléments intéressants du fond grâce au flou de profondeur et ainsi mieux percevoir les formes et les distances. Lorsque le flou est imposé, c'est cette liberté qui est compromise.

Dans le cas du cinéma, un véritable langage cinématographique s'est mis en place au fil des années et de nombreux codes sont maintenant acceptés sans que cela nous perturbe. Lorsque nous regardons un film, nous savons que nous n'aurons pas le contrôle du cadre, ni des zones de flou. De plus, notre habitude du cinéma nous permet de décrypter immédiatement la plupart des images, et de suivre instinctivement le sens de lecture que nous propose le réalisateur. Nous identifions directement les points d’intérêts sans nous perdre dans les fonds flous, car nous savons qu'ils ne sont pas considérés comme importants par le réalisateur. Le découpage haché et la structure d'un blockbuster actuel seraient probablement incompréhensibles pour un spectateur du début du cinéma, encore habitué à la caméra fixe, car il ne posséderait tout simplement pas le langage de base pour interpréter les images à la vitesse à laquelle un spectateur actuel est habitué.

1.2.b L'exemple du cinéma relief stéréoscopique

Le cinéma relief stéréoscopique est un exemple intéressant, car il pose plusieurs problèmes avec cette liberté. Il rajoute en effet au cinéma classique une sensation de tridimensionnalité qui nous donne instinctivement envie de faire le point sur des éléments qui ne sont pas ceux voulus par le réalisateur, et qui ne sont pas forcément dans la zone de profondeur de champ. C'est à notre avis un des ingrédients de la fatigue visuelle accrue avec les films en relief, car notre œil passe sont temps à tenter sans succès de rendre nettes les parties de l'image qu'il

survole. De plus, dans la réalité, l'œil doit coordonner deux mécanismes pour rendre plus net un élément : l'ajustement de la convergence des deux yeux et la contraction du cristallin de chaque œil pour faire le point à une certaine distance. La convergence des yeux est liée à la parallaxe qui existe entre les deux yeux du fait de leur écartement. Avec les films reliefs stéréoscopiques, seule la convergence des yeux est gérée, grâce à la séparation en deux images reproduisant la parallaxe de chaque œil. La contraction du cristallin n'est par contre pas gérée, ce qui oblige l'œil à désapprendre des réflexes de coordination inscrits depuis toujours entre la convergence et la contraction. Ce problème tient principalement à la technique de diffusion qui ne reproduit pas exactement la situation du réel, mais aussi à la captation, qui enregistre deux images, mais pas l'information de distance des points, qui est implicite dans la parallaxe entre les deux images. Une évolution technique nous offrira peut-être une solution. Dans le cas contraire et si le cinéma relief fini par être définitivement adopté, nous nous habituerons probablement à séparer les deux mécanismes de l'œil et d'ici quelques années nous saurons décoder le langage du cinéma stéréoscopique instinctivement comme nous l'avons fait avec celui du cinéma traditionnel.

1.2.c Le cas des jeux vidéo

Dans le cas des applications interactives, et notamment des jeux vidéo, le problème est encore plus visible, car le joueur a généralement le contrôle du point de vue de la caméra. L'utilisation d'un flou de profondeur nous empêche alors de contempler à loisir les décors du jeu dont nous sommes pourtant censés nous approprier l'espace et la structure. Le joueur curieux trouve souvent cela insupportable, et certains ont même mal aux yeux, comme dans le cas du cinéma relief, à force de tenter de regarder des éléments de décor masqués par le flou. Si le joueur à la liberté de contrôler la caméra, il devrait également pouvoir contrôler le point de focus de la profondeur de champ. Pour ces raisons, l'utilisation du flou doit être très limitée et ne pas entraver l'impression de liberté du joueur. Les cinématiques intégrées au jeu peuvent,

Figure II.7: Resident evil 4 (Capcom – 2005)

et même doivent pouvoir utiliser la profondeur de champ, afin de donner un aspect cinématographique à ces scènes et pouvoir exploiter le vocabulaire très riche du cinéma. Le jeu vidéo « Resident Evil 4 » utilise la profondeur de champ uniquement dans les cinématiques, à la fois en raison de considérations techniques, le calcul du flou étant coûteux en performance, mais aussi dans le but de conserver des phases de jeu très lisibles. Certains jeux comme « Limbo » ou « Proun » exploitent merveilleusement bien le flou en tant qu'effet de style sur les décors. Le résultat visuel possède énormément de caractère et une ambiance marquée. Dans ces jeux, nous pouvons remarquer que les décors sont très simples, donc facilement lisibles, et la caméra n'est pas directement contrôlée par le joueur. Pour « Limbo », elle suit un chemin prédéfini sur un plan à la manière d'un jeu dit « 2D », et dans « Proun » elle fait toujours face à la course, un peu comme si c'était le décor qui bougeait. C'est cet automatisme de la caméra et sa grande inertie qui permet à ce flou de profondeur très prononcé de ne pas être frustrant, mais au contraire de participer à l'expérience du jeu.

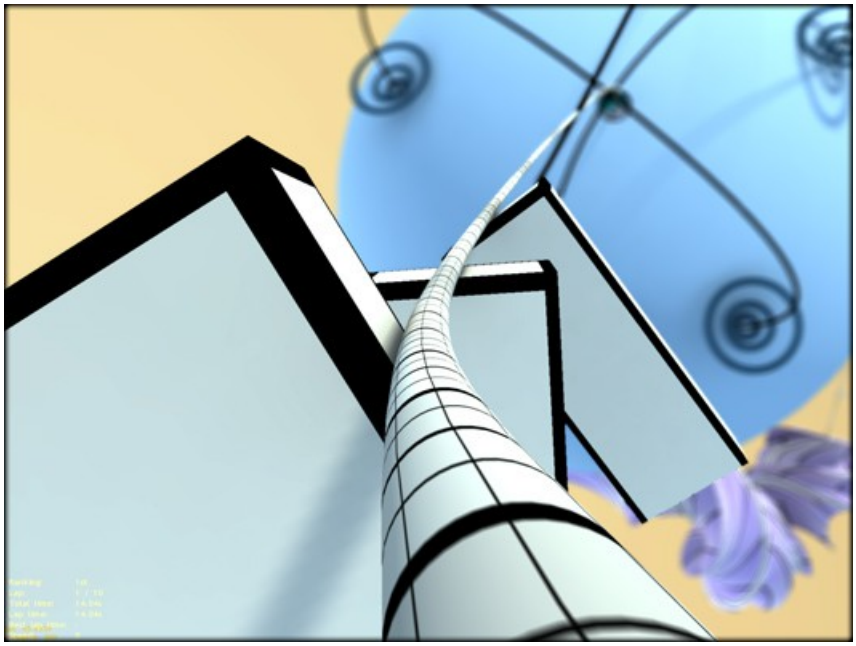


Figure II.8: Proun (Joost van Dongen - 2011)

2 Principes optiques

2.1 Les modèles de caméra

2.1.a Modèle de caméra « sténopé »

Une caméra « sténopé » ou « pinhole » (soit trou d'épingle en anglais) est un modèle de caméra qui est utilisée pour projeter une scène 3D sur une image 2D, la surface réceptrice. C'est une projection en perspective, issue de la « camera obscura », déjà utilisé par les peintres au seizième siècle. La lumière qui parcourt la scène entre dans le dispositif au moyen d'un trou supposé infiniment petit puis frappe la surface réceptrice. Comme un seul rayon de lumière par point de l'image peut passer par le trou, il en résulte une image parfaitement nette. Ce modèle de caméra est simple et est donc devenu le plus utilisé en imagerie 3D.

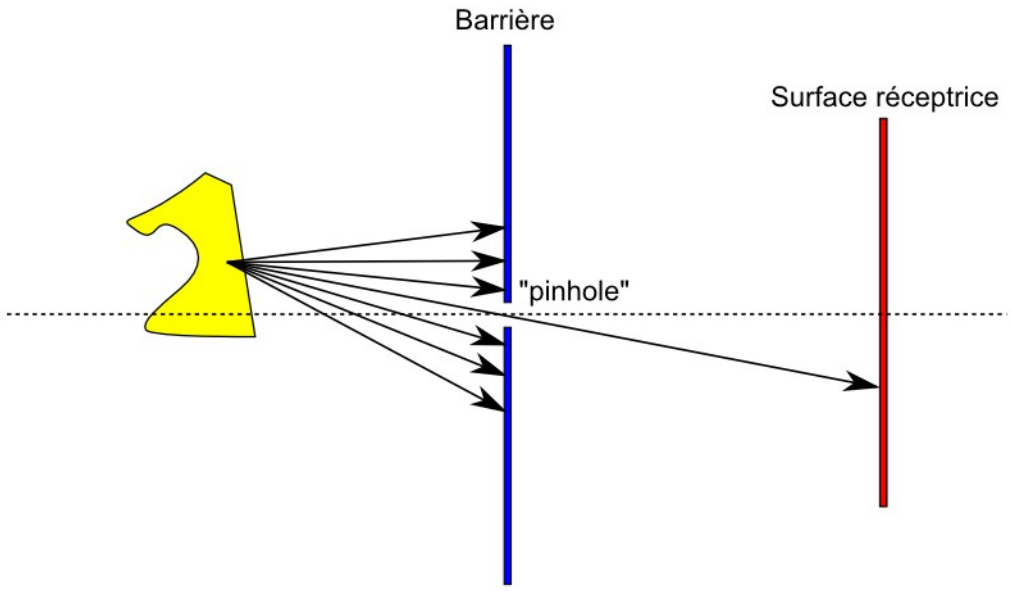


Figure II.9: Modèle de caméra « sténopé » ou « pinhole ».
Un seul rayon atteint chaque pixel de la surface réceptrice.

2.1.b Modèle de caméra à lentille mince

Dans un système optique, comme une caméra ou l'œil humain, l'image est formée par les rayons de lumières, réfractés par une lentille, qui viennent frapper la surface réceptrice. Dans le cas de l'œil, il s'agit de la rétine, pour une caméra analogique, de la pellicule photosensible et dans le cas d'une caméra numérique, c'est le capteur CCD ou CMOS. Afin de capturer suffisamment de lumière pour faire réagir la surface photosensible, il est nécessaire d'utiliser une grande ouverture, puis de faire converger la lumière à travers une lentille pour la

concentrer sur la surface réceptrice. C'est le modèle de la lentille fine, ou « thin lens » en anglais, présenté dans le domaine informatique par Potmesil[7]. La distance à laquelle la lumière va converger, après son passage dans la lentille, dépend de la distance de l'émetteur lumineux. Si la surface sensible se trouve à cette distance, la lumière va converger vers un unique point et donc former une image nette, sinon le point lumineux va s'étaler et former un rond (en anglais nommé CoC : circle of confusion), et donc une image floue.

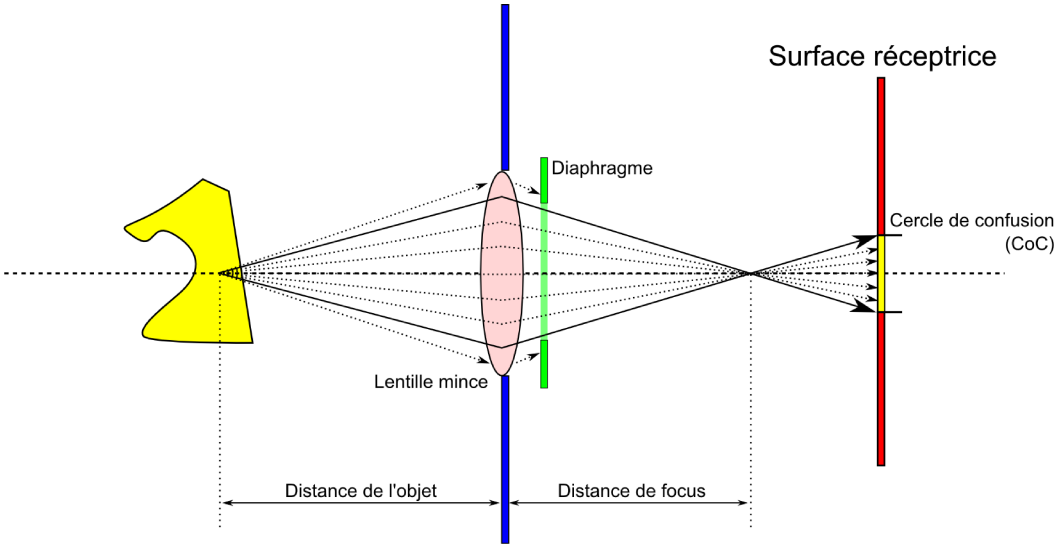


Figure II.10: Modèle de caméra à lentille mince.
Plusieurs rayons convergent à travers la lentille, et forment un cercle si la surface réceptrice n'est pas à la distance de focus, par rapport à un objet à une distance donnée.

2.2 La profondeur de champ, entre la caméra et l'œil

> *S'il était besoin de justifier le prestige de la profondeur de champ, il suffirait de faire remarquer qu'elle correspond à la vocation dynamique et exploratrice du regard humain qui fixe et fouille dans une direction précise (du fait de l'étroitesse de son champ de netteté) et à des distances très variées (du fait de son pouvoir d'accommodation).*
>
> *Le langage cinématographique*[8]

La caméra semble au premier abord être une réplique de l'œil humain. Nous allons voir que si certains principes sont les mêmes, de nombreuses différences existent à la fois dans les mécanismes, mais aussi dans l'impression résultante.

7 M. Potmesil et I. Chakravarty, « A lens and aperture camera model for synthetic image generation », *ACM SIGGRAPH Computer Graphics* 15, nº 3 (1981): 297–305.

8 Marcel Martin, *Le langage cinématographique*, Collection Septième art (Paris: Ed. du Cerf, 1994).

La profondeur de champ est définie comme la zone entre le point le plus proche de la caméra et le point le plus loin qui produit une image nette sur la pellicule. Le plan focus est la distance à la caméra qui offre une image complètement nette. Les images ont donc une profondeur de champ limitée. Les objets trop éloignés de la distance de focus vont apparaître flous.

Le dernier élément important de la caméra à lentille fine est le diaphragme, qui permet de faire varier la taille de l'ouverture à la lumière. Le but principal du diaphragme est de pouvoir faire varier la taille de l'ouverture, ce qui influence la taille de la profondeur de champ. Nous verrons que le diaphragme est très important dans la forme que prendra le flou. Le rôle du diaphragme dans l'œil humain est tenu par l'iris, qui peut s'ouvrir et se fermer pour contrôler la quantité de lumière qui entre dans l'œil.

Une différence importante existe néanmoins entre l'œil humain et une caméra. La caméra, pour ajuster la distance du plan de focus, fait varier la distance entre la surface réceptrice et la lentille. Au contraire, l'œil humain utilise la contraction du cristallin (qui joue le rôle de lentille) pour faire varier son indice de réfraction et donc la distance focale. Cette contraction, que l'on nomme accommodation, est en grande partie instinctive, l'œil s'adapte pour ajuster la distance de focus afin de rendre l'objet du regard le plus net possible. L'œil se concentre sur une petite partie de son champ de vision et ajuste automatiquement la distance focale pour la rendre nette, ce qui élimine pratiquement l'impression de flou de profondeur dans notre vision de tous les jours. La précision dans le plan de l'image est également concentrée sur un point précis, car la répartition des récepteurs de l'œil n'est pas homogène. Nous ne percevons précisément qu'une petite partie de notre champ de vision, mais les mouvements du globe oculaire permettent au cerveau de reconstituer une image complète. L'œil possède ainsi cette caractéristique supplémentaire de concentrer la précision disponible sur une partie spécifique d'un décor, le reste se perdant dans le flou. Au contraire, la caméra est un dispositif capturant une image complète avec une résolution répartie également sur toute la surface. L'image est destinée à être montrée à un spectateur qui concentrera lui-même son attention oculaire sur une partie de l'image. La caméra ne capture qu'une image intermédiaire qui sera réinterprétée au moment de la diffusion. L'intégration de la profondeur de champ dans cette image intermédiaire est un moyen artificiel de recréer le mécanisme naturel de l'œil et d'en donner le contrôle au réalisateur.

2.3 Paramètres de la profondeur de champ

La profondeur de champ est liée à un nombre important de réglages que peut manipuler le photographe traditionnel afin de choisir quels objets apparaîtront nets alors que les autres seront flous. Le facteur principal est bien sûr le choix de la lentille, et principalement sa taille et son rayon de courbure, qui sont en photographie des valeurs fixes que nous ne pouvons pas faire varier à volonté, car il faut posséder l'objectif physique correspondant à chaque taille voulue, et opérer le changement d'optique à chaque fois. La distance entre la surface sensible et la lentille, la distance focus, va servir à régler la distance de netteté de la profondeur de champ. L'ouverture du diaphragme, ou ouverture focale, a une influence directe sur la taille du cercle de confusion (CoC), c'est-à-dire la force du flou. En effet, plus le diaphragme est fermé et moins la lumière passe et donc moins elle est dispersée par la lentille. Bien sûr, il est en général nécessaire de compenser la perte de luminosité liée à la fermeture du diaphragme par une augmentation du temps d'exposition. Ainsi, au lieu d'utiliser un grand nombre de rayons lumineux divergents sur un petit temps d'exposition à la lumière, nous allons utiliser des rayons concentrés sur la même trajectoire pendant un temps plus long.

2.4 Formes de lentilles

La taille et l'intensité lumineuse du flou va dépendre notamment de la focale et de l'ouverture du diaphragme, ainsi que de la taille de la « pellicule ». La forme du flou sur l'image, quant à elle, sera déterminée principalement par la forme du diaphragme. Cette forme est le plus souvent référencée sous le nom de « bokeh ».

Habituellement, le but premier des concepteurs de lentilles pour appareils photo ou caméras est d'obtenir la meilleure qualité d'image sur le plan de focus de la lentille. Pour cela, ils se concentrent sur l'élimination des imperfections de la lentille, sur les façons d'associer plusieurs lentilles pour obtenir les caractéristiques d'ouverture et de distance focale voulue avec la meilleure netteté. Par contre, les qualités esthétiques des parties hors focus de l'image sont entrées plus récemment dans les exigences des fabricants de lentilles et chez les photographes. Le concept de « bokeh », dérivé du japonais « boke », désigne justement les caractéristiques esthétiques des parties floues de l'image.

Le nombre de lames dans le diaphragme et son degré d'ouverture va former sur la pellicule les figures géométriques habituelles telles que pentagones, hexagones, octogones ou même

cercles si le diaphragme est complètement ouvert ou bien constitué de suffisamment de lames pour que la différence ne soit plus perceptible entre un polygone et un cercle. D'autres caractéristiques influencent la forme, mais aussi la répartition de la luminosité dans la forme, et même la répartition des couleurs. Il s'agit notamment des aberrations optiques présentées par les lentilles utilisées, qui vont concentrer la lumière différemment selon la provenance des rayons. L'image du bokeh peut ainsi se présenter sous bien des aspects, certains auront une forme différente selon qu'il s'agit d'un flou d'avant-plan ou d'arrière-plan, certains présenteront une forme ovale orientée vers le point central de l'image (vignetage optique), certains auront des bords colorés, une forme de « donut » avec la lumière concentrée sur les bords du cercle de confusion. La longueur d'onde de la lumière, c'est-à-dire sa couleur, va entraîner une différence dans l'angle de réfraction par la lentille et provoquer un effet d'irisation, de bordure d'une couleur particulière. Les principes optiques liés aux caméras, aux lentilles et aux aberrations optiques sont développées par Ray[9], l'implémentation informatique est présentée par Heidrich[10].

Il est tout à fait possible de fabriquer un « bokeh » personnalisé, avec toutes les formes possibles, tout simplement en plaçant un cache devant l'objectif avec la forme voulue en découpe au centre, ce qui fonctionne comme un diaphragme avec une forme fixe.

La forme du bokeh n'est pas toujours facilement perceptible dans le flou, par exemple si le fond n'est pas très contrasté, peu de différences seront visibles entre les différentes formes de bokeh. La notion importante pour comprendre les effets de « bokeh » est que le point lumineux a été capté à l'origine dans la scène à une certaine intensité lumineuse et que celle-ci va se retrouver répartie sur toute la zone du bokeh. Ainsi, les points qui ne sont pas très lumineux vont laisser une tache très pâle une fois étalée par le bokeh. Cette tâche se noiera dans l'étalement de la lumière de ses voisins. Au contraire, les éléments ponctuels très lumineux vont former une figure très nette qui se détachera du reste des points peu lumineux. Il faut souvent un temps d'exposition important et une focale très grande pour que la quantité de lumière qui arrive soit suffisamment grande pour que les points lumineux hors focus ne soient pas invisibles, indistinguables. L'idéal pour observer ce type d'effet est donc d'avoir une

9 Sidney F. Ray, *Applied Photographic Optics: Lenses and Optical Systems for Photography, Film, Video, Electronic and Digital Imaging* (Focal Press, 2002).

10 W. Heidrich, P. Slusallek, et H.P. Seidel, « An image-based model for realistic lens systems in interactive computer graphics », in *Graphics Interface*, 1997, 68–75.

partie floue constituée de lumières ponctuelles très vives sur un fond sombre et une partie nette éclairée normalement.

L'utilisation d'un objectif réel pose certaines contraintes. Notamment sur les réglages qu'il faut effectuer. De nombreux éléments sont interdépendants : le niveau lumineux de l'image, la taille de la profondeur de champ, la distance où placer la caméra dépendent tous les trois des mêmes facteurs, qui sont : la taille de la lentille, l'ouverture du diaphragme, la taille de la « pellicule » (ou du capteur numérique) et le degré de « zoom ». Ainsi il peut être compliqué d'obtenir la bonne profondeur de champ avec telle focale et à telle distance de l'objet à filmer. Pour avoir une image suffisamment lumineuse il faut alors souvent augmenter le temps d'exposition, ce qui dans le cas du cinéma n'est pas possible au-delà d'une certaine valeur si l'on veut avoir 24 images par secondes et peu de flou de mouvement. Il y a également des problèmes avec la géométrie du décor, car souvent pour obtenir une faible profondeur de champ, le photographe se place loin de la scène et utilise un fort zoom. La géométrie de l'image est alors proche de celle d'une vue dite « isométrique » c'est-à-dire presque sans effets de perspective, ce qui est perturbant dans la perception de l'espace du spectateur et diminue son sentiment d'être inclus dans l'action.

Avec une simulation numérique, toutes ces contraintes disparaissent et les paramètres auparavant interdépendants peuvent désormais être manipulés à volonté. Nous perdons le lien avec l'optique réelle, mais les réglages en sont grandement facilités. Nous pouvons déplacer les plans de début et de fin de la profondeur de champ sans toucher à la position de la caméra, et modifier la durée d'exposition à souhait, ce qui s'apparente plus à une simple modification de la luminosité de l'image. Pour faciliter les réglages, notamment auprès des personnes habituées à travailler avec des caméras réelles, le choix est souvent fait de présenter un modèle virtuel de véritable caméra, avec tous les paramètres traditionnels de vitesse d'exposition, de focale, d'ouverture de diaphragme, de taille de pellicule, tout en proposant au besoin de manipuler la profondeur de champ de manière indépendante pour aller plus loin que les possibilités des caméras réelles.

2.5 Aller plus loin que la modélisation de la caméra

Jusqu'à présent nous n'avons envisagé la profondeur de champ que comme une copie la plus fidèle possible du dispositif optique de la caméra, mais il est tout à fait possible d'étendre l'idée de profondeur de champ à toute une gamme d'effets qui servent aux mêmes buts :

donner des informations spatiales, diriger le regard, apporter une ambiance et une esthétique particulière à l'image et faire passer un message. L'informatique peut alors jouer un rôle très important où les techniques disponibles vont rendre possible la mise en relation de différentes caractéristiques de l'image d'une manière complexe, mais avec une logique propre et des règles précises. Dans le cas du flou de profondeur, il s'agit de faire le lien entre la distance d'un pixel et sa couleur ainsi que celles de ses voisins.

En ce qui concerne la maîtrise du regard et surtout la simplification des parties les moins importantes de l'image, des techniques simples sont connues depuis longtemps en peinture, telles que la modification de la couleur, de la saturation et de la luminosité d'un pixel en fonction de la distance. Les effets de brouillard de distance entrent ainsi dans cette catégorie et correspondent à la « perspective atmosphérique » utilisée en peinture. Désaturer le fond d'une image permet une bonne lisibilité et une meilleure appréhension de l'éloignement des objets. Diminuer le contraste de certaines parties a le même type d'effet qu'un flou en rendant certaines parties moins lisibles et plus homogènes, permettant au regard de se focaliser sur les parties contrastées et pleines de détails de l'image. Nous pouvons également trouver d'autres manières de simplifier une image, par exemple par les effets de rendu non-photoréalistes et expressifs, où le but est de donner un aspect dessiné et parfois plus abstrait à l'image. Ces effets peuvent être combinés à une sélection selon la distance ou bien toute autre caractéristique pour différentier les parties de l'image qui doivent être simplifiées et celles qui doivent au contraire conserver tous leurs détails.

Ces techniques ont bien entendu aussi un caractère esthétique propre et vont influer l'ambiance ressentie par le spectateur. Elles peuvent également véhiculer un sens et évoluer au fil du temps pour illustrer la progression de l'histoire ou des personnages. Une application de la profondeur de champ dans le cadre d'un effet évoquant une ambiance « rêve » sera développée dans la partie sur le rendu expressif.

3 Introduction aux techniques

3.1 Contraintes d'implémentation

3.1.a La profondeur de champ comme un « post-process »

Au cours de cette étude, nous allons considérer le processus de création de l'image et l'application de l'effet de profondeur de champ comme deux étapes séparées. La première étape, que nous n'étudierons pas, consiste à créer une image purement bidimensionnelle, c'est à dire constituée d'une grille 2D de pixels. Cette image doit être accompagnée d'une information de profondeur pour chaque pixel. En général, nous utilisons simplement la distance entre le pixel et la caméra. L'image est alors constituée de pixels décomposés en 4 valeurs : la valeur de rouge, de vert et de bleu pour la couleur et la valeur de profondeur. Tous les outils classiques peuvent être utilisés pour créer cette image : modèles 3D, lumières, animations, effets spéciaux programmés. Par contre, une fois l'image créée, le « post-process » n'a plus accès à aucune de ces données préalables et utilise uniquement la grille de pixel.

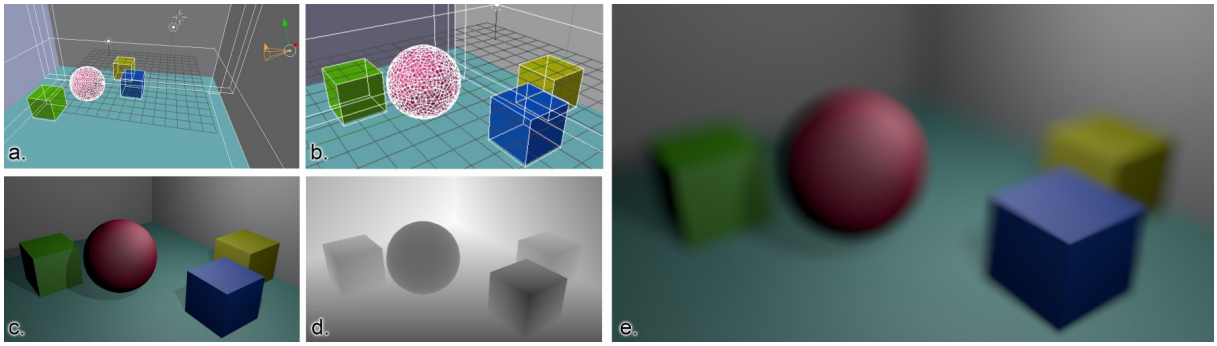


Figure II.11: La profondeur de champ par un post-process.
la scène 3D (a.) est vue depuis la caméra (b.), seul le rendu (c.) accompagné de son Z-buffer (d.) seront utilisés pour le calcul du flou de profondeur dans l'image finale (e.).

La seule contrainte sur les techniques utilisables dans la première étape est que les effets doivent pouvoir définir une profondeur précise à chaque pixel. Ainsi, nous verrons que la gestion des effets de transparences est délicate, puisqu'un pixel donné est influencé par des informations situées à plusieurs distances différentes.

La seconde étape est celle du flou de profondeur proprement dit, qui est appliqué ici comme un « post-process » c'est-a-dire une étape séparée du reste du calcul de l'image, qui utilise en entrée la grille de pixels, avec leurs couleurs et leurs informations de distance et produit en

sortie une nouvelle grille de pixels imitant l'effet de profondeur de champ d'une caméra réelle. Nous considérons donc la profondeur de champ avant tout comme un filtre de flou avec certaines caractéristiques spéciales. Nous verrons néanmoins également au cours de l'étude qu'il est possible de l'envisager d'autres façons. L'avantage de séparer le calcul de la profondeur de champs du reste de l'image est de rendre l'application du flou orthogonale aux techniques et à la complexité de l'image initiale. Ainsi, les performances sont plus facilement prédictibles et le coût du calcul de profondeur de champ est pratiquement fixe quelque soit la complexité de l'image initiale.

3.1.b À propos du rendu HDR

L'effet de profondeur de champ joue avec des valeurs de luminosité très diverses et une grande partie de l'effet provient des lumières de très fortes intensités qui sont ensuite dispersées par le flou sur une zone très large, produisant les formes de bokeh caractéristiques. La capacité à stocker et travailler avec ces hautes lumières est donc essentielle et pour cela nous devons utiliser un stockage dit « HDR » (« High Dynamic Range ») et des calculs effectués en espace linéaire[11].

L'affichage sur un écran permet d'atteindre 16 millions de couleurs. Ce chiffre parait grand, mais est en fait bien insuffisant pour représenter l'amplitude de luminosité et de couleur que l'œil humain est capable de percevoir, depuis les faibles lueurs dans une nuit noire jusqu'à la lumière du soleil vue directement. D'un côté, il est préférable que les écrans ne soient pas capables de reproduire la lumière du soleil et par la même de nous brûler les yeux. Mais d'un autre, il est alors nécessaire d'établir un mécanisme d'équivalence entre les valeurs physiques et leur image sur l'écran. L'œil humain possède la capacité d'adapter automatiquement l'amplitude lumineuse qu'il perçoit, à un moment donné, par le biais de deux mécanismes. Tout d'abord l'iris bien sûr, qui joue le même rôle que le diaphragme d'une caméra en contrôlant la quantité de lumière autorisée à entrer. Le second dispositif est le basculement entre les différents capteurs de lumières. L'œil possède en effet deux types de capteurs photosensibles : les bâtonnets et les cônes, qui ne sont pas sensibles aux mêmes quantités de lumière. Le jour, en présence d'une grande quantité de lumière, ce sont les cônes qui sont actifs et qui fournissent les couleurs. La nuit, lorsque l'éclairage est faible, les bâtonnets jouent alors leur rôle, car ils sont sensibles à des variations faibles de lumières. Ils sont par

11 Larry Gritz et Eugene d' Eon, « The Importance of Being Linear », *GPU Gems 3, Addison-Wesley* (2008): 529–542.

contre incapables de percevoir les couleurs, la vision de nuit est monochrome et pratiquement aveugle aux couleurs rouges. Le passage d'un type de capteur à l'autre prend un certain temps, l'activation de la vision de nuit après un éblouissement prend entre vingt et trente minutes, même si les cônes s'activent beaucoup plus rapidement dans l'autre sens. Ainsi, les couleurs et luminosités que nous percevons sont toujours à mettre en relation avec le contexte de captation afin de retrouver une valeur absolue correspondant à une quantité de lumière donnée.

3.1.c La correction gamma

Afin de reproduire ces phénomènes avec un écran dont la précision et l'amplitude lumineuse sont limitées, le plus adéquat est d'effectuer tous les calculs dans un espace absolu, avec des valeurs indiquant une quantité de lumière par canal de couleur par exemple, représentées de manière linéaire. Une fois l'image complète, il est alors nécessaire de la convertir dans l'espace des valeurs reproductibles par l'écran.

Les 16 millions de couleurs sont en réalité insuffisants pour encoder directement les valeurs de luminosité que l'œil perçoit à un moment donné, sans même prendre en compte son caractère adaptatif. En effet, l'œil humain n'a pas une courbe de précision linéaire, mais au contraire, il est plus sensible aux petits écarts de luminosité dans les basses lumières que dans les valeurs plus hautes. Dans le domaine chromatique, l'œil est plus sensible aux verts qu'aux rouges et aux bleus. Pour améliorer la répartition des échantillons de couleurs, nous utilisons donc une courbe de gamma qui va augmenter la précision dans les basses lumières en diminuant celle des hautes lumières. La courbe gamma suit une équation de type G(x)=x^g avec une valeur typique de 2.2 pour g. Les écrans sont prévus pour recevoir de tels signaux et les afficheront en compensant la courbe de gamma pour retrouver les luminosités correctes. Il y a encore quelques années, les jeux travaillaient uniquement en espace gamma, du niveau des textures jusqu'à l'image affichée par l'écran. Effectuer des calculs dans cet espace est difficile, une simple addition donne un résultat faux lorsqu'elle est effectuée en espace gamma. Il est nécessaire de travailler dans un espace linéaire dans lequel nous faisons les calculs, puis d'appliquer la transformation gamma juste avant l'affichage.

En plus de la courbe de gamma, il est possible d'utiliser le même type d'adaptation des valeurs lumineuses qu'utilise l'œil. En fonction de la quantité lumineuse moyenne perçue à un moment donné ou sur une période donnée, la courbe est modifiée pour utiliser telle ou telle plage de valeurs lumineuses dans l'image finale.

La compression des couleurs et des luminosités est une technique utilisée en photographie et au cinéma. Les différents types de pellicules photographiques ont des sensibilités différentes et permettent de compresser certaines valeurs lumineuses pour produire une image donnant la sensation de la scène réelle sans nécessiter la même amplitude lumineuse. Certains jeux ont tenté de reproduire le type de courbe présenté par les pellicules afin d'obtenir le même type de rendus. Ces courbes de transformations lient les valeurs HDR (avec une amplitude de 0 à 16 par exemple) à des valeurs entre 0 et 1. Voici quelques courbes classiques : la courbe gamma, la courbe de Reinhard[12], la simplification de la courbe de Haarm-Pieter Duiker par Jim Hejl et Richard Burgess-Dawson, et finalement la courbe utilisée pour le jeu Uncharted 2[13].

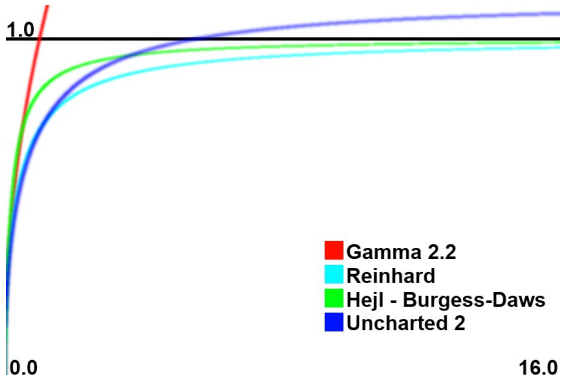


Figure II.12: Courbes de compression HDR vers LDR

Les équations sont extraites du blog de John Hable[14]. Pour plus d'informations sur les courbes possibles et leurs intégrations en temps réel, voir « HDR Tone-mapping in real time »[15].

Dans le cadre du travail sur la profondeur de champ, il est très important de pouvoir gérer des valeurs de luminosité plus grandes que celles qu'il sera possible d'afficher au final sur l'écran. Dans cet espace, les valeurs peuvent dépasser 1 et les calculs sont précis (flottants) et linéaires. La profondeur de champ disperse les hautes lumières sur les voisins par le biais du flou. Sans la capacité de représenter toute une variété de hautes lumières, il serait impossible d'obtenir un effet de flou de profondeur photoréaliste.

12 E. Reinhard et al., « Photographic tone reproduction for digital images », *ACM Transactions on Graphics* 21, nº 3 (2002): 267–276.

13 John Hable, « Uncharted 2 : HDR Lighting » (Naughty Dog - Conférence GDC, 2010).

14 John Hable, « Filmic Games », 2010, http://filmicgames.com/archives/75.

15 Jonas Nilsson, « HDR tone-mapping in real time » (2007).

3.2 Algorithmes de calcul du flou

Un flou est essentiellement une convolution, c'est à dire l'application d'une fonction donnée (le filtre) en tout point d'une seconde fonction (le signal à filtrer). La fonction de filtrage possède en général une fenêtre d'application en dehors de laquelle elle est presque nulle et peut donc être ignorée. L'algorithme procède alors en appliquant la fonction en chaque point de l'image, en utilisant les valeurs de la fenêtre d'application pour déterminer la force avec laquelle chaque pixel voisin va influencer le résultat final. Pour que la convolution n'influence pas la luminosité globale de l'image, il est nécessaire que le résultat soit normalisé, c'est-à-dire que la somme des contributions de chaque pixel du filtre soit de 1.

3.2.a Flou gaussien

Le premier flou que nous allons étudier est le flou gaussien. Les poids utilisés par les échantillons suivent une loi normale gaussienne. La formule pour le calcul du poids est du type : $y=e^{-x^2}$ Ce flou possède plusieurs caractéristiques qui le rendent très attrayant. Son influence deviens rapidement proche de zéro et même si elle n'atteint jamais réellement zéro, nous pouvons la considérer nulle très rapidement et donc conserver une fenêtre assez réduite, ce qui est essentiel pour avoir un algorithme rapide. Le flou gaussien est également séparable, ce qui signifie que si nous appliquons le flou sur une dimension uniquement, puis ensuite sur la seconde, nous obtenons le même résultat que si nous avions appliqué directement sur les deux dimensions. Au niveau de la complexité par contre, un algorithme séparable est beaucoup moins cher à calculer qu'un algorithme non séparable dès que la fenêtre de filtrage s'élargit. La complexité d'un filtre arbitraire en deux dimensions de taille $n\times n$ est de $O(n^2)$ alors que sur un filtre séparable il est seulement de $O(n\times 2)$. Les deux courbes se croisent à la valeur 2. Au-delà de cette valeur, la séparabilité est plus efficace. Le filtre gaussien correspond à une lentille parfaitement ronde et forme un « bokeh » très flou.

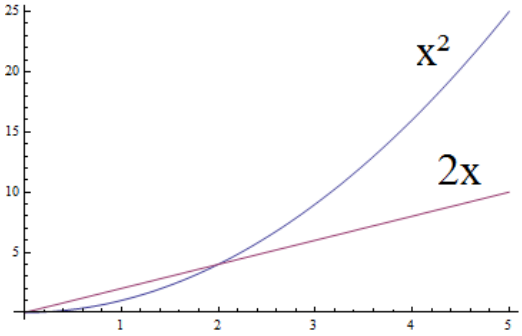


Figure II.13: Comparaison de la complexité entre les filtres séparables (x^2) et non-séparables ($2x$).

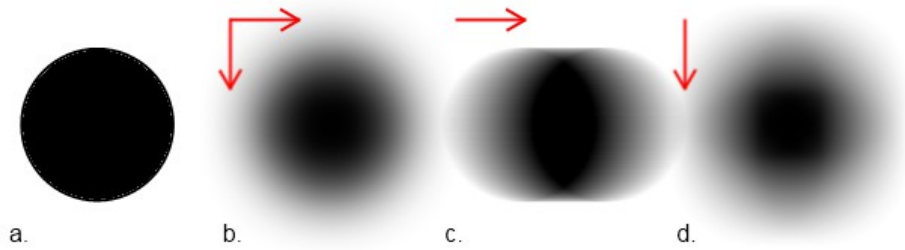


Figure II.14: Le flou gaussien séparable.
***a** : Forme initiale à flouter. **b** : application d'un flou gaussien directement sur les deux dimensions. **c** : application d'un flou gaussien sur l'axe x uniquement. **d** : application du flou sur l'axe y à partir de l'image c.*

3.2.b Flou avec disque de Poisson

Une autre méthode pour obtenir un flou important est d'utiliser une distribution en forme de disque de poisson. En anglais, on le désigne souvent sous le nom de « kernel poisson disk ». Cette distribution est issue d'observations biologiques sur la répartition des récepteurs photosensibles dans l'œil humain. Il s'agit simplement d'aller chercher un certain nombre d'échantillons placés aléatoirement sur la surface d'un disque, sans être forcé d'échantillonner chaque pixel du disque. Les échantillons sont choisis selon certaines caractéristiques, la plus importante étant qu'ils soient répartis équitablement sur toute la surface en respectant une distance minimum et maximum entre chaque échantillon[16]. Il est également possible d'utiliser une fonction de densité afin d'obtenir une plus grande proportion d'échantillons au centre du cercle.

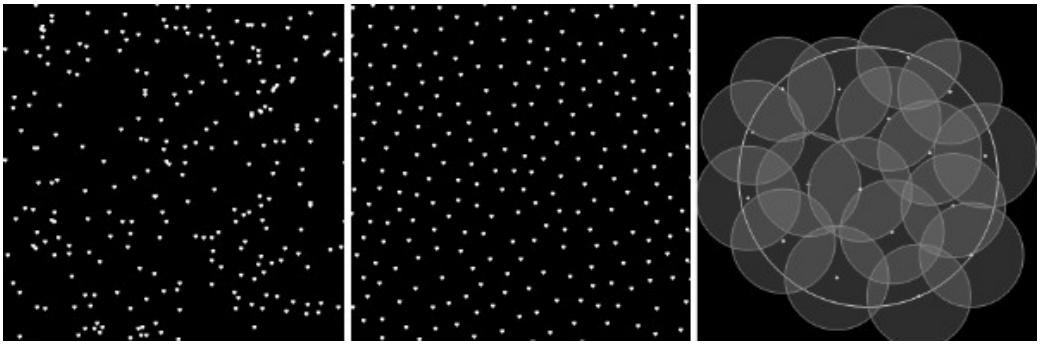


Figure II.15: Distribution de points aléatoires (gauche), selon une distribution de Poisson (centre) et suivant la forme d'un disque de Poisson (droite)
Image H. Tulleken et Coderhaus

Le poids des différents échantillons peut être fixe, pour obtenir une forme nette, ou bien suivre une fonction gaussienne ou toute autre fonction pour simuler une répartition de la lumière différente. Concrètement, l'utilisation d'une distribution en disque de poisson permet de diminuer le crénelage qui interviendrait forcément à partir du moment ou trop peu

16 R.L. Cook, « Stochastic sampling in computer graphics », *ACM Transactions on Graphics (TOG)* 5, nº 1 (1986): 51–72.

d'échantillons sont utilisés. En échange, le bruit de l'image deviendra plus prononcé, ce qui est en général moins perturbant pour l'œil que le crénelage. L'idéal est d'utiliser une distribution différente pour chaque pixel, afin d'obtenir un bruit le plus homogène possible. Pour cela, une astuce couramment choisie consiste à utiliser une unique distribution, mais en y appliquant une rotation différente pour chaque pixel. Une revue des différentes techniques et de leurs implémentations dans le contexte du calcul de la profondeur de champ est disponible dans « Real Time Depth Of Field Simulation »[17]. Il existe un petit outil[18] pour calculer une distribution aléatoire en disque de Poisson basée sur une méthode de calcul très rapide[19].

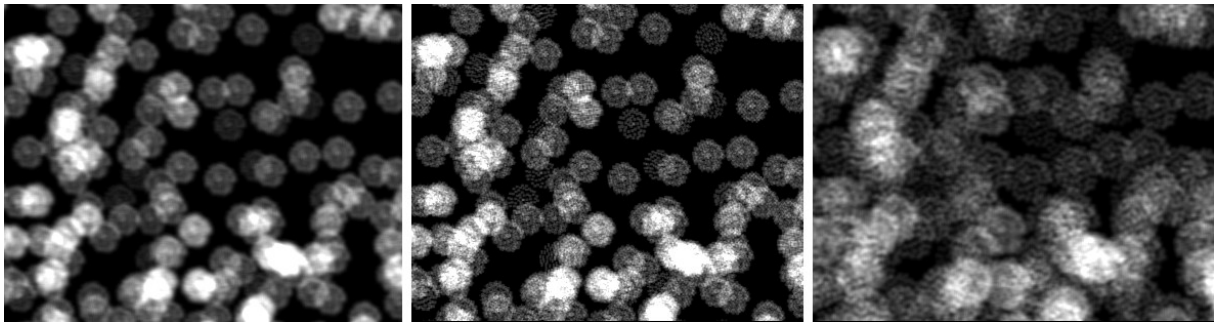


Figure II.16: Influence de la résolution sur la taille du filtre de flou nécessaire. Flou avec 55 échantillons suivant un disque de Poisson appliqué sur une image basse résolution (gauche). Le même flou, mais avec une image haute résolution (centre), le nombre d'échantillons n'est plus assez important, des points individuels commencent à apparaître. Le même flou, sur une basse résolution mais avec une plus grande taille de filtre (droite).

3.2.c Flou suivant la forme d'une lentille

Pour obtenir des lentilles de formes spéciales, le plus simple est d'appliquer une fenêtre de la taille du filtre, ce filtre pouvant être une texture, ce qui permet d'obtenir n'importe quelle forme. Malheureusement, dès que la taille du filtre augmente, la complexité augmente au carré, ce qui rend cet algorithme très peu utilisable. L'astuce de la séparabilité ne fonctionne plus car chaque pixel doit être interprété séparément pour prendre en compte la forme complexe de la lentille.

17 G. Riguer, N. Tatarchuk, et J. Isidoro, « Real-time depth of field simulation », *ShaderX2: Shader Programming Tips and Tricks with DirectX* 9 (2003): 529–556.

18 « Poisson Disk Generator », 2010, http://www.coderhaus.com/?p=11.

19 D. Dunbar et G. Humphreys, « A spatial data structure for fast Poisson-disk sample generation », in *ACM Transactions on Graphics (TOG)*, vol. 25, 2006, 503–508.

3.2.d Techniques par dispersion et par rassemblement

Il existe essentiellement deux manières différentes d'appliquer un filtre : par dispersion et par rassemblement. Le rassemblement est la technique la plus utilisée, car elle est efficacement supportée par les cartes graphiques. Il consiste à calculer la nouvelle valeur de chaque pixel en allant rassembler les valeurs des pixels voisins pour en calculer une moyenne pondérée en fonction des valeurs du filtre. La dispersion consiste au contraire à disperser la valeur de chaque pixel sur les pixels voisins.

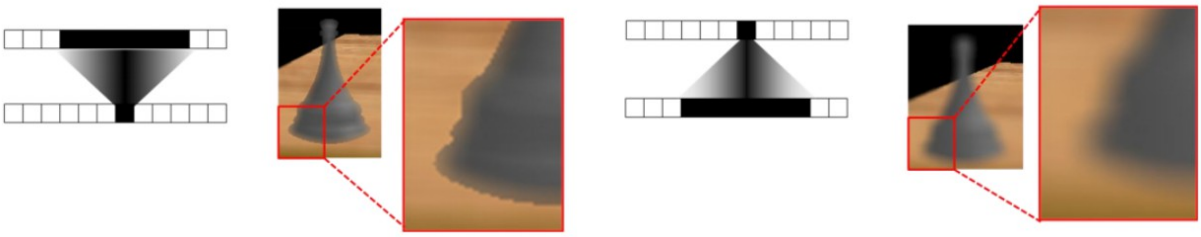


Figure II.17: Comparaison entre rassemblement et dispersion.
Le rassemblement produit des silhouettes dures sur les objets flous (gauche). La dispersion floute correctement les bords en avant plan.

Images J.Kosloff et al.

Si les caractéristiques du filtre ne changent pas spatialement, c'est à dire en fonction de l'emplacement sur l'image, les deux techniques donnent un résultat équivalent. Malheureusement, le calcul du flou de profondeur de champ nécessite de modifier la taille du flou en fonction de la distance du pixel concerné. Dans ce cas précis, la technique par rassemblement ne fonctionne plus aussi bien, car physiquement, c'est chaque point lumineux qui va être étalé sur la pellicule, donc c'est bien la profondeur, et donc la taille de dispersion du pixel initial qui est important. Lorsque nous opérons par rassemblement, nous calculons en fait une moyenne des valeurs de dispersion dans le voisinage du pixel final, mais ce n'est qu'une moyenne, donc moins précise que la technique par dispersion qui tient compte réellement de la dispersion de chaque pixel. Il est également important de noter que la variabilité de la taille du filtre en fonction du pixel rend le filtre non séparable, même pour un filtre gaussien.

La taille du flou est l'élément qui va influencer le plus la complexité de l'algorithme de flou. Il est donc important de pouvoir choisir la plus petite taille possible, et de faire ce choix le plus tôt possible dans l'algorithme afin de pouvoir restreindre ensuite le nombre de calculs à faire. L'algorithme par rassemblement ne peut pas connaître la taille du flou nécessaire, car en réalité celle-ci est multiple, chaque point voisin venant influencer le point concerné selon sa propre taille de flou.

Avec l'algorithme par rassemblement, il existe alors deux possibilités. Avec la première, nous sommes obligés de parcourir tous les pixels selon une taille de flou maximum, et d'opérer par rejet. Selon la taille de flou propre à chaque échantillon, nous choisissons de l'utiliser ou non dans le calcul du pixel courant. Cette technique peut se coupler avec une distribution en disque de Poisson, mais si le nombre d'échantillons retenu est trop faible, le bruit sera d'autant plus grand. La seconde possibilité consiste à utiliser une approximation de la taille du flou nécessaire, sans s'occuper des artefacts qui en résultent. En utilisant cette approximation, la technique par rassemblement est si efficace qu'elle reste très utilisée malgré les nombreux artefacts gênants.

3.2.e Artefacts de la méthode par rassemblement

L'utilisation de la méthode par rassemblement avec un coefficient de convolution (une valeur de flou) variable selon les pixels présente de nombreux artefacts qu'il est possible d'atténuer en partie. L'erreur la plus visible apparaît lorsque l'avant-plan net va « baver » sur un arrière-plan flou. En effet, les pixels du fond étant flous, ils vont utiliser un schéma d'échantillonnage très étendu qui va déborder sur les pixels nets de l'avant-plan. Il est bien sûr possible d'exclure ces pixels, mais seulement une fois que nous connaissons la valeur de ces pixels, donc trop tard pour recalculer la taille du schéma d'échantillonnage. Si l'on exclut les pixels qui ne doivent pas être floutés, le nombre d'échantillons restant est souvent trop restreint pour obtenir un flou correct[20].

Figure II.18: Artefacts typiques du flou par rassemblement.
Les bords de l'avant-plan net créent un halo sur l'arrière-plan flou (gauche)
L'avant-plan flou conserve des bords nets à cause de l'arrière-plan net (droite)
Image P.Lönroth et M.Unger

Un second artefact gênant est en quelque sorte l'inverse du premier. Lorsque l'avant-plan est flou et que l'arrière-plan est net, il est difficile de flouter correctement les bords de l'avant-

20 P. Lönroth et M. Unger, « Advanced Real-time Post-Processing using GPGPU techniques » (2008).

plan. En effet, il nous manque ici des informations, car l'avant-plan masque une partie de l'arrière-plan qui devrait apparaître derrière une fois flouté. Or ces informations n'existent plus, car nous considérons ici l'effet de profondeur de champ comme une passe séparée qui est calculée a posteriori de l'image nette. Il n'est alors plus possible d'accéder à des informations de la scène tridimensionnelle si elles ne sont pas présentes dans cette image. Pour contrer cet artefact, il faut trouver un équilibre entre la dureté des contours de l'avant-plan et la qualité voulue.

3.3 Crénelage spatial et temporel

Le flou de profondeur est un problème difficile dans le rendu temps réel, car contrairement à la réalité, il n'est pas possible, pour des raisons de performances, d'envoyer de nombreux rayons pour chaque pixel afin d'obtenir le flou due à l'utilisation de la lentille. En effet, chaque rayon de lumière passant par un récepteur de l'œil a suivi un chemin légèrement différent. Il est parti de la source lumineuse, a rebondi sur certains objets vers l'œil puis a été concentré par le cristallin sur la surface de réception. La diversité de rayons différents qui arrive à un endroit donné de la surface de réception est liée principalement à la taille de la lentille. Plus celle-ci est grande et plus des rayons avec des directions et des positions différentes vont entrer dans la lentille pour atterrir sur la surface de réception, ce qui entraîne une quantité de flou plus importante.

Certaines recherches visent néanmoins à accumuler les valeurs de ces rayons, soit en décomposant la scène en un grand nombre de couches rendues séparément[21], soit en découpant le trajet de la lumière en sphères[22] dont il est possible de calculer une approximation de l'émission lumineuse par des techniques hiérarchiques (tel qu'un stockage de la scène dans un octree).

Le flou de profondeur fait partie de l'ensemble des problèmes liés au faible nombre de rayons qu'il est possible d'envoyer par pixel en temps réel. Il existe d'autres problèmes liés à ce manque d'information, et nous allons voir qu'il est possible de les rassembler afin d'obtenir, globalement, de meilleurs résultats.

21 Sungkil Lee, Elmar Eisemann, et Hans-Peter Seidel, « Depth-of-field rendering with multiview synthesis », *ACM Transactions on Graphics* 28, nº 5 (décembre 1, 2009): 1.

22 Cyril Crassin et al., « Beyond Triangles : Gigavoxels Effects In Video Games », *SIGGRAPH 2009: Talks*, ACM SIGGRAPH '09 (2009).

Les interactions entre les récepteurs visuels dans un œil ou une caméra ne se limitent pas à un rayon par pixel toute les trentièmes de seconde. La principale différence entre l'œil physique et le calcul informatique, est que chaque cellule photoréceptrice va recevoir des centaines de photons et en faire une intégrale avant de la communiquer au cerveau alors que le calcul informatique va utiliser un seul rayon par pixel. Cette différence implique des problèmes de crénelage (ou aliasing) de deux sortes, le crénelage spatial et le crénelage temporel.

3.3.a Le crénelage spatial

Pour illustrer notre propos, prenons l'exemple d'un cercle noir sur un fond blanc. Quelle résolution d'image faut-il pour représenter ce cercle ? Si chaque pixel ne peut être que soit noir soit blanc, car soit à l'intérieur du cercle soit à l'extérieur, il faut une résolution très importante pour que le cercle paraisse parfait. Par contre, si nous autorisons les valeurs de gris intermédiaires entre noir et blanc, il est alors possible d'afficher un cercle semblant parfait avec beaucoup moins de pixels, en utilisant, comme valeur de gris, le pourcentage du pixel qui serait à l'intérieur du cercle, donc noir. Cela permet d'améliorer la qualité de l'image au niveau du sous-pixel. Dans l'exemple du cercle, cela nécessite de pouvoir calculer facilement le pourcentage d'un pixel qui est à l'intérieur, ce qui est clairement possible de manière analytique, à l'aide de l'équation mathématique du cercle. Nous calculons l'aire de l'intersection entre un cercle et un carré. Le ratio entre cette aire et celle du carré nous procure notre niveau de gris. Cette méthode peut se révéler bien moins chère qu'une augmentation du nombre de pixels à calculer. Malheureusement, il est difficile de calculer ce pourcentage pour des formes arbitraires, complexes, et nous devons nous contenter d'approximations. Les formes des lignes, des carrés, des ovales et des triangles peuvent être calculées analytiquement.

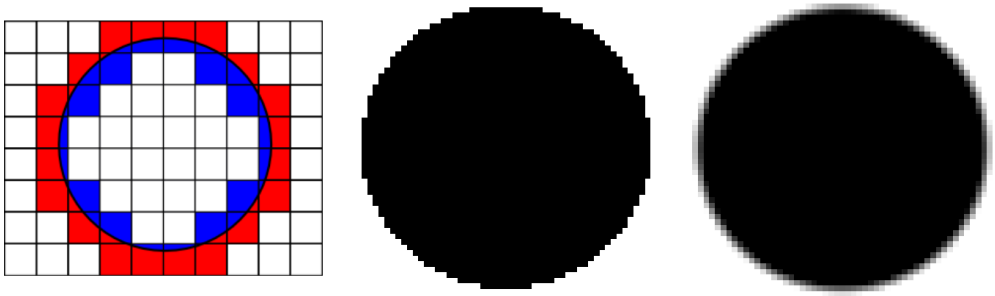


Figure II.19: Anticrénelage spatial d'un cercle.
À gauche, les pixels de bordures auront une valeur de gris correspondant au ratio entre la partie bleu (intérieur) et la partie rouge (extérieur). Au centre, le cercle n'autorise pas les niveaux de gris. À droite, le cercle est anticrénelé grâce à un léger flou.

Cette méthode d'anticrénelage est parfois approximée tout simplement par un flou appliqué uniquement sur les bords des objets. Les différents paramètres du flou sont réglés afin d'obtenir un flou important sur les bords tout en gardant le centre des objets net.

Une fois que le pourcentage de recouvrement d'une forme est connu, ou approximé, il nous faut encore connaître la couleur de cette forme et celle du fond afin de faire un mélange entre les deux. Une heuristique valable ici est d'utiliser les couleurs des pixels voisins. En effet, en général, une forme recouvre plusieurs pixels voisins, et nous cherchons principalement à améliorer les zones de contacts entre ces zones plus ou moins uniformes. Nous pouvons utiliser un pixel voisin en provenance de chacune des formes qui empiètent sur le pixel, et si nous connaissons la proportion de chaque forme, nous pouvons mélanger les valeurs des voisins pour obtenir un pixel anticrénelé.

3.3.b Le crénelage temporel

Nous utilisons toujours l'exemple du cercle noir sur fond blanc, mais en mouvement très rapide cette fois. Sans anticrénelage temporel, le cercle apparaît simplement à différents endroits, mais toujours net. Il est difficile de savoir si le cercle bouge réellement ou bien si au contraire ce sont plusieurs cercles qui s'affichent successivement, surtout si le mouvement est périodique. Si nous ajoutons un anticrénelage temporel, ce qui serait le cas si nous filmions avec une caméra utilisant un temps d'exposition suffisamment long, le cercle ne paraîtra pas net, mais laissera une trace de son passage. Nous pouvons maintenant déterminer immédiatement que le cercle bouge et l'image est plus naturelle. Ce type d'anticrénelage est souvent appelé « flou de mouvement » (« motion blur »). Il s'agit ici, pour connaître l'intensité lumineuse d'un pixel, de calculer l'intégrale du temps pendant lequel un pixel est resté à

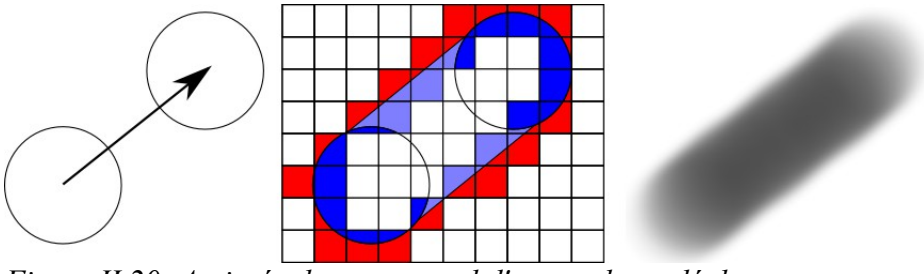


Figure II.20: Anticrénelage temporel d'un cercle en déplacement linéaire (à gauche).
Au centre, les pixels traversés par le cercle au cours de son déplacement obtiendront un niveau de gris selon le rapport entre la partie rouge et la partie bleu, et selon la vitesse de déplacement du cercle. Le résultat est une tache allongée floue (à droite).

l'extérieur du cercle au cours de la durée d'une image (c'est à dire en général 1/30ième de seconde).

3.3.c Unification des techniques : les filtres spatio-temporels

Nous avons pu voir que les trois problèmes que sont la profondeur de champ, l'anticrénelage et le flou de mouvement, proviennent en fait du même déficit en informations au niveau du pixel. Il est donc possible de créer un seul filtre qui cherche à contrer ces trois problèmes. Pour le moment, ce sont principalement l'unification de l'anticrénelage spatial et de l'anticrénelage temporel qui ont fait l'objet de recherches dans le domaine. La profondeur de champ paraît néanmoins une suite logique, qui est déjà évoquée hors du domaine du temps réel[23].

L'idée est d'utiliser le maximum d'informations présentes, que ce soit en utilisant les voisins du pixel (filtre spatial) ou les pixels dans les images précédentes (filtre temporel), afin de calculer une nouvelle image qui présentera moins de crénelages spatial et temporel. L'ensemble des informations disponibles forme un échantillonnage quadridimensionnel. Le principal intérêt d'une telle approche est de répartir l'erreur équitablement sur les quatre dimensions afin de minimiser les artefacts visibles[24]. Ce type de technique a été développé en temps réel pour le jeu « Crysis 2 »[25], en utilisant les images précédentes pour effectuer un superéchantillonnage, c'est-à-dire calculer plusieurs valeurs pour un seul pixel final de l'image. Les valeurs des images précédentes d'un pixel viennent ainsi améliorer nos informations. Le rayon utilisé pour le calcul de chaque pixel est légèrement différent à chaque image, même si la caméra ne bouge pas, afin d'obtenir des informations différentes. Le rayon est choisi au hasard dans la pyramide de vision formée par le pixel concerné. Le grand avantage provient du fait que l'ensemble des informations soit filtré, que ce soit le modèle lumineux, les bords des objets ou même les textures, au contraire des techniques classiques d'anticrénelage qui se contentent de filtrer les bords. L'intégration du flou de profondeur dans un filtre spatio-temporel est encore incomplète, mais semble être une voix prometteuse, notamment lorsque la scène est représentée par un nuage de point dans une structure hiérarchique au lieu des traditionnels polygones.

23 C. J Gribel, R. Barringer, et T. Akenine-Möller, « High-quality spatio-temporal rendering using semi-analytical visibility », *ACM Transactions on Graphics (TOG)* 30, nº 4 (2011): 54.

24 R. Herzog et al., « Spatio-temporal upsampling on the GPU », in *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, 2010, 91–98.

25 Tiago Sousa, « Anti-Aliasing Methods in CryENGINE 3 » (Siggraph 2011).

4 Techniques avancées du domaine

Les techniques classiques que nous avons vus jusqu'à maintenant sont efficaces et s'adaptent bien à différentes résolutions et différents matériels, notamment les consoles actuelles (PS3, Xbox360). Nous allons maintenant voir des évolutions plus récentes dans le domaine, et qui permettent d'exploiter réellement, dans des jeux vidéos, la forme de la lentille.

4.1 Effets de lentilles sous forme de « sprites »

La base de cette technique a d'abord été présentée dans le domaine du rendu précalculé[26] et consiste à dessiner la forme du « bokeh » centré sur chaque pixel, en utilisant le CoC pour fixer la taille de la forme. L'image de la lentille peut dès lors se présenter sous la forme très contrôlable d'une texture, que l'artiste peut peindre selon la forme désirée. Bien évidemment, le coût en performance est ici maximal et impraticable avec une haute résolution en temps réel.

La technique a ensuite été modifiée en constatant que la forme de la lentille joue en majorité sur les endroits ou la luminosité de l'image est très variable. Ainsi, un fond uniforme, une fois flouté, présentera peu de différences selon la lentille utilisée. Au contraire, un point lumineux très intense placé au milieu d'un fond noir se dilatera selon la forme caractéristique de la lentille, qui deviendra de ce fait très apparente. Le principe est donc d'utiliser un flou simple (gaussien par exemple) pour filtrer les parties à faible fréquence et d'ajouter par dessus l'image de la lentille sur les points de l'image les plus lumineux et identifiables, c'est-à-dire les hautes fréquences.

Le point délicat de la technique concerne ce que l'on nomme en anglais « l'overdraw », c'est à dire le nombre de recouvrement qu'un pixel subit. Si les performances n'étaient pas un problème, nous pourrions simplement afficher l'image de la lentille sur chaque pixel de l'image. Malheureusement, chaque pixel devrait alors être réécrit de très nombreuses fois, ce qui est incroyablement lourd. L'essentiel est donc de réduire le nombre de formes qui vont être dessinées tout en conservant la plus grande précision au niveau de la position des hautes lumières. Le résultat est très bon, l'aspect de la lentille peut être choisi complètement arbitrairement et les performances sont liées à la taille de la lentille par pixel, sans qu'une

26 Cyril Pichard, Sylvain Michelin, et Olivier Tubach, « Photographic Depth of Field Blur Rendering », *Proc. WSCG 2005* (2005).

taille maximum doive être choisie. Ainsi, si peu de pixels ont une taille de flou importante, le coût sera réduit.

Figure II.21: Détails d'images issues de "The Samaritan" de Epic (2011) à gauche et de "Crysis 2" de Crytek (2011) à droite.

Des techniques de ce type ont été implémentées avec succès par plusieurs développeurs de jeux, notamment Crytek pour Crysis 2[27], Epic pour leur démonstration technologique DirectX 11 « The Samaritan »[28] et Futuremark dans leur outil d'évaluation des performances 3DMark 11[29]. Dans les trois cas, cette technique n'est utilisée que sur les matériels les plus puissants et récents, notamment en se basant sur DirectX 11. L'utilisation des « geometry shaders » permet en effet de facilement changer le nombre et l'emplacement des carrés texturés à afficher sans avoir à les envoyer depuis le CPU. L'algorithme est gourmand en ressource et nécessite une carte graphique DirectX 11 très récente, ce qui rend cette implémentation impossible sur les consoles actuelles (PS3, XBOX 360).

4.2 Implémentation de test par l'utilisation d'une histopyramide

Nous avons implémenté cette technique lors d'un test, en utilisant une variation très intéressante utilisant directx 9. Elle a été proposée par Matt Swoboda, membre du groupe de démomaker « Fairlight » sur son blog[30]. La technique consiste toujours à trouver les points les plus lumineux et à y placer un « sprite » représentant la forme du « bokeh ». Pour cela, nous

27 Nickolay Kasyan, Nicolas Schulz, et Tiago Sousa, « Secrets of CryENGINE 3 Graphics Technology » (Crytek, Conférence Siggraph 2011).

28 Martin Mittring et Bryan Dudash, « The Technology Behind theDirectX 11 Unreal Engine"Samaritan" Demo » (Epic Games/Nvidia, GDC 2011).

29 Futuremark, « 3DMark 11 Whitepaper », 2011, http://www.3dmark.com/wp-content/uploads/2010/12/3DMark11_Whitepaper.pdf.

30 Matt Swoboda, « Numb Res by CNCD & Fairlight », 2011, http://directtovideo.wordpress.com/.

utilisons une « histopyramide », ou histogramme pyramidal. Il s'agit de stocker le nombre d'éléments qui correspond à une certaine caractéristique dans une structure hiérarchique pyramidale. Cette construction est utile pour rassembler des points éparpillés dans un espace essentiellement vide, en 2D, 3D ou même n'importe quel nombre de dimensions. Dans notre cas, nous partons de l'image contenant la luminosité et la profondeur des pixels et nous appliquons un filtrage pour ne conserver que les pixels hors focus et très lumineux. Une fois cette passe appliquée, nous construisons l'histopyramide de cette sélection. La première « marche » de la pyramide mesure la moitié en x et en y de la taille de l'image d'origine. Ainsi, chaque pixel de ce niveau regroupe quatre pixels de l'image d'origine, et contient une valeur représentant le nombre de pixels de celle-ci qui sont sélectionnés. Les niveaux suivants se basent ensuite sur celui qui vient d'être calculé, avec une taille divisée par deux à chaque fois, et chaque pixel accumule à nouveau les valeurs de quatre pixels du niveau précédent. Une fois la pyramide complète, le niveau tout en haut mesure une taille de 1 par 1, et contient le nombre total de pixels sélectionnés dans toute l'image. Nous pouvons alors lire cette information depuis le CPU afin de savoir combien de « sprites » il est nécessaire d'envoyer à la carte graphique. Il est également possible de créer dès le départ une liste de « sprites » très longue stockée sur la carte graphique (sous la forme d'un vertex/index buffer par exemple), et de simplement envoyer l'ordre de rendu accompagnée du nombre de « sprites » à rendre.

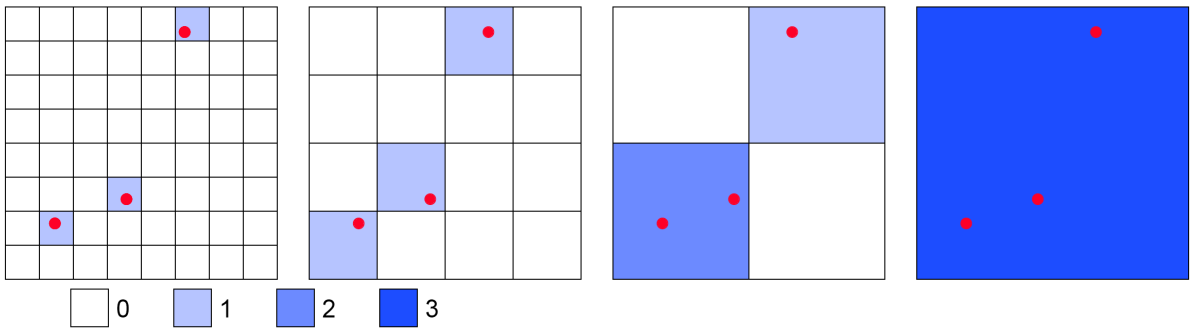


Figure II.22: Les différents niveaux d'une histopyramide contenant trois points.Chaque niveau est une grille de moins en moins grande dont la valeur (en bleu) dépend du nombre de points contenus dans la case.

La seconde partie de l'algorithme, une fois le nombre de « sprites » connu, est de pouvoir retrouver la position de chacun directement sur le GPU. La valeur que contient chaque « sprite » envoyé à la carte graphique est simplement son numéro. À partir de celui-ci, il est très simple de retrouver la position du « sprite » en traversant la pyramide de haut en bas. À chaque niveau, nous recherchons dans lequel, parmi les 4 pixels du niveau suivant, se trouve le numéro du « sprite » courant. En effet, chacun de ces 4 pixels contient le nombre de

« sprites » qu'il représente, il nous suffit donc de soustraire successivement les valeurs de chaque pixel au numéro du sprite jusqu'à obtenir une valeur négative, ce qui signifie que le pixel concerné contient le sprite courant. L'utilisation d'une texture au stade du vertex shader est requise afin de modifier la position des sommets des « sprites » en fonction de l'histopyramide contenue dans des textures. Cette fonctionnalité existe dans les shaders depuis la version 3.0, qui est disponible sur PS3 et Xbox360. Cette technique est très efficace, surtout si le nombre de pixels sélectionnés est faible au regard de la taille de l'image. Une certaine redondance de calcul est à constater, car chaque sommet d'un « sprite » devra faire sa propre recherche, donc le nombre de calculs est multiplié par 4 par rapport à une implémentation avec des « geometry shaders ». La complexité de l'algorithme, pour une image de taille $m \times n$ et une sélection de k éléments est constitué de la création de l'histopyramide (soit $O(\log(\frac{1}{m \times n}))$) puis de la consultation de cette histopyramide pour chaque sommet (4) de chaque élément k rendu (soit $O(4 \times k \times \log(\frac{1}{m \times n}))$. Cette structure hiérarchique fait penser à un algorithme de « quadtree », et il est en effet possible d'ajouter des éléments de cette technique. Il s'agit alors de stocker, tant que le nombre d'éléments pour un pixel est de 1, la position de celui-ci. Ainsi, cette position remonte la hiérarchie jusqu'à ce qu'un autre élément soit ajouté. Lors de la recherche de la position, nous pouvons donc nous arrêter plus tôt, dès que le nombre d'éléments est de 1, et connaître immédiatement la position sans avoir besoin de descendre jusqu'en bas de la hiérarchie. Cette dernière idée n'est pas forcément efficace, car la programmation de la carte graphique se base sur une exécution en parallèle des calculs ce qui rend les embranchements difficiles. En clair, il est mieux d'exécuter un calcul homogène, mais parfois inutile, plutôt que d'ajouter du code et de la complexité pour éviter ces calculs.

Grâce à l'histopyramide, nous pouvons créer un « sprite » sur les points les plus lumineux. Pour faire varier leurs luminosité, taille et couleur, nous utilisons une version floutée de la texture originale. Cette version floutée peut être réalisée avec un flou gaussien séparable et sert principalement à limiter le crénelage temporel. En effet, le processus de sélection et d'affichage du bokeh accentue volontairement les variations dans l'image initiale. Ainsi, de faibles variations d'une image à la suivante peuvent produire des variations importantes une fois l'algorithme appliqué. Avec l'utilisation d'un flou préalable, ce risque est moindre. Cette technique est, au final très intéressante, mais notre test d'implémentation nécessite encore

quelques ajustements avant de pouvoir être utilisé en production. Il s'agit notamment d'améliorer la qualité et les variations brutales d'une image à la suivante. Cette technique ne sera pas utilisée sur le jeu de la société DONTNOD.

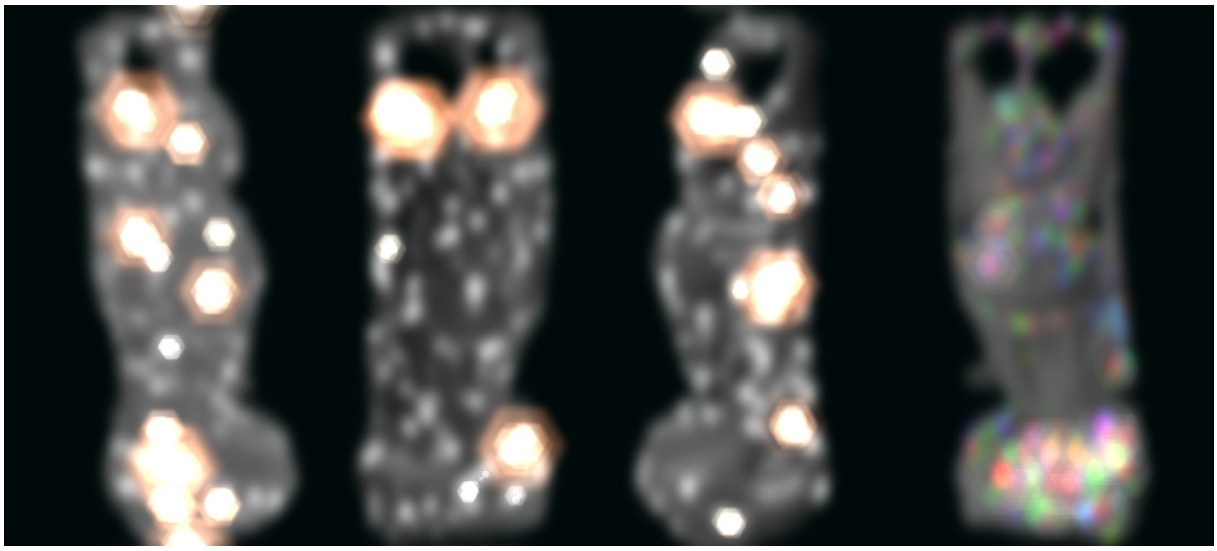


Figure II.23: Différents tests de flous à partir de "sprites" hexagonaux positionnés grâce à une histopyramide.

4.3 Flou multipasse hexagonal

Dice, les développeurs du jeu « Battlefield 3 » ont proposé un algorithme de profondeur de champ très intéressant et qui présente plusieurs points communs avec les résultats obtenus avec notre technique, présentée plus bas. Le but est de profiter de certaines caractéristiques particulières de la forme hexagonale pour proposer une méthode de flou avec une lentille de cette forme. Une technique similaire à été présenté par Masaki Kawase dans une conférence au CEDEC 2009[31].

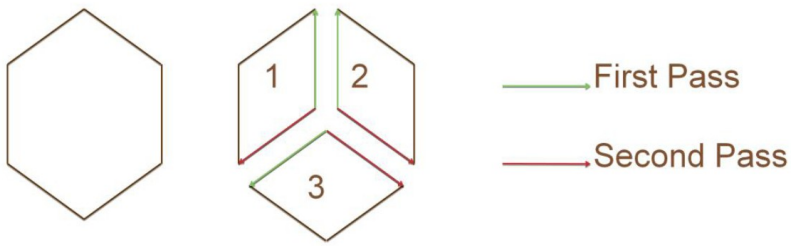


Figure II.24: Organisation des passes de flous directionnels pour former un hexagone.

31 Masaki Kawase, « Anti-Downsized Buffer Artifacts » (Présentation en japonais, CEDEC, 2009), www.daionet.gr.jp/~masa/archives/CEDEC2009_Anti-DownsizedBufferArtifacts.ppt.

L'algorithme de flou gaussien, tout comme celui du « box filter » (flou carré sans variation du poids des échantillons) bénéficie d'être séparable, c'est-à-dire qu'il est possible d'obtenir le même résultat en effectuant deux passes de filtres, l'une sur x puis l'autre sur y. La complexité est alors linéaire et non quadratique (2n au lieu de n²). Les développeurs de Dice ont très justement remarqué que les deux axes d'un box filter n'ont pas besoin d'être alignés. La forme du flou appliqué peut dès lors être un parallélogramme quelconque. La forme hexagonale se décompose en trois parallélogrammes et peut donc être dessinée en appliquant trois flous séparables en deux passes, puis en réunissant ces trois résultats en une passe finale afin d'obtenir la distribution hexagonale.

Un nombre de sept passes est assez important, il faut donc déjà avoir un filtre de grande taille pour que le coût de la séparation soit intéressant par rapport à un simple parcours de tous les pixels ($O(n^2)$). La complexité de l'algorithme présenté ici, en terme de nombre d'appels à une texture pour un filtre de taille n, est de $O(\frac{6\times n}{2}+3)$ ou $O(3\times n+3)$, les six premières passes étant des passes de flous sur une direction de taille n/2 (complexité O(n/2) car chacun ne parcourt que la moitié de la taille n de l'hexagone) et la dernière passe utilisant les résultats des 3 derniers flous (O(3)). La valeur d'équilibre entre les deux techniques est environ de 3,79 : en effet à cette valeur les deux calculs de complexité s'égalisent à peu près :

$$O(n^2)\simeq O(3\times n+3) \quad \rightarrow \quad 14{,}36\simeq 14{,}37 \ \textit{pour } n = 3{,}79$$

Il faut donc au moins une taille de filtre de 4 de côté pour que la séparabilité commence à payer. Pour des valeurs faibles de n, le coût reste très important et un filtre gaussien séparable sera bien plus rapide, de même qu'un filtre par disque de poisson. De plus, l'algorithme nécessite une place mémoire plus importante, car il faut stocker les six passes de flous séparément avant de réunir les résultats des trois dernières. Les développeurs de chez Dice ont néanmoins réussi à faire baisser le nombre de passes à deux seulement, pour 4 flous en

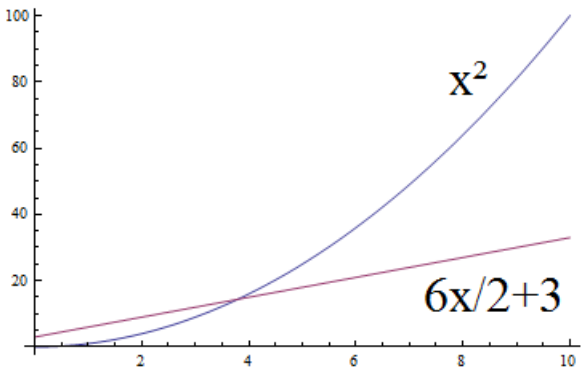


Figure II.25: Comparaison de la complexité entre les filtres séparables (x²) et non-séparables (6x/2+3) dans le cas de l'hexagone.

tout, en utilisant astucieusement le rendu dans plusieurs textures en même temps (Multiple Render Target). La complexité en terme de nombre d'appels à une texture est alors la même que celle d'un flou gaussien séparable : $O(\frac{n}{2}\times 4) = O(n\times 2)$, mais la division en 4 flous au lieu de deux et l'utilisation du MRT demandent le double de bande passante et diminuent quelque peu les performances. Pour des tailles de flou plus importantes, les développeurs de Dice introduisent une technique permettant d'atteindre des performances plus élevées que le flou gaussien séparable, le raffinement itératif. Cette technique exploite l'homogénéité des poids présents dans le cas du filtre hexagonal, au contraire d'un flou gaussien, qui utilise des poids variables selon les pixels.

Figure II.26: Détails des effets de profondeur de champ par l'utilisation des techniques de M.Kawase (à gauche) et de Dice (au centre et à droite).

Finalement, cette technique est très efficace, mais reste un algorithme de rassemblement et non de dispersion. La variabilité de la taille du filtre pose ainsi certains problèmes et peu d'informations nous indiquent quelles sont leurs solutions pour les contrebalancer.

5 Création personnelle : Hexagonal summed area table

Nous avons passé en revue plusieurs techniques récentes de reproduction du flou de profondeur. Nous allons maintenant étudier notre principale contribution de cette partie. Nous avons créé ici une méthode innovante de flou de profondeur qui mise sur la reproduction de la forme hexagonale de la lentille.

5.1 Les Summed Area Table

5.1.a Présentation de l'algorithme classique

Une summed area table (SAT) est un type de stockage d'image 2D. Toute texture peut être convertie et stockée dans ce format. Nous remplaçons l'intensité d'un pixel de l'image donnée par la somme des intensités de tous les pixels situés dans le rectangle délimité par ce pixel et l'un des coins de l'image. Dans l'implémentation originale[32], c'est le coin inférieur gauche qui a été choisi. Durant le reste de cette étude, nous utiliserons le coin supérieur gauche, ce qui facilite certains calculs. Le principe reste le même quel que soit le coin choisi.

Ainsi, le dernier pixel en bas à droite contiendra la somme de tous les pixels de l'image. Grâce à ce format, nous pouvons calculer la somme de n'importe quelle zone rectangulaire avec seulement quatre appels à la SAT. Le coût de calcul et le nombre d'accès mémoire sont donc constants, quelle que soit la taille de la zone rectangulaire dont on souhaite connaître la somme. De plus, diviser la somme par l'aire du rectangle nous donne la moyenne des valeurs contenue dans le rectangle. Il est dès lors possible de calculer par exemple des flous de type « box filter » sans que la complexité soit dépendante de la taille du filtre. Cette taille peut même être variable selon les pixels sans problèmes.

Ainsi, pour connaître la somme E des valeurs du rectangle délimité par les pixels a (situé en haut à gauche du rectangle) et b (en bas à droite), nous allons calculer tout d'abord les positions des points c et d, qui vont constituer les deux autres coins du rectangle. Ensuite, la somme ne dépendra que des valeurs de ces 4 points.

32 F.C. Crow, « Summed-area tables for texture mapping », *ACM SIGGRAPH Computer Graphics* 18, nº 3 (1984): 207–212.

Le calcul est le suivant :

$$E = a + b - c - d$$

Les SAT peuvent servir à de nombreux effets, notamment pour les opérations de filtres[33], par exemple pour calculer des réflexions floutées[34] [35], des flous sur des textures d'ombrages[36], des effets de profondeurs de champs[37], mais aussi du superéchantillonnage (supersampling) ou du mip-maping comme dans l'application initiale.

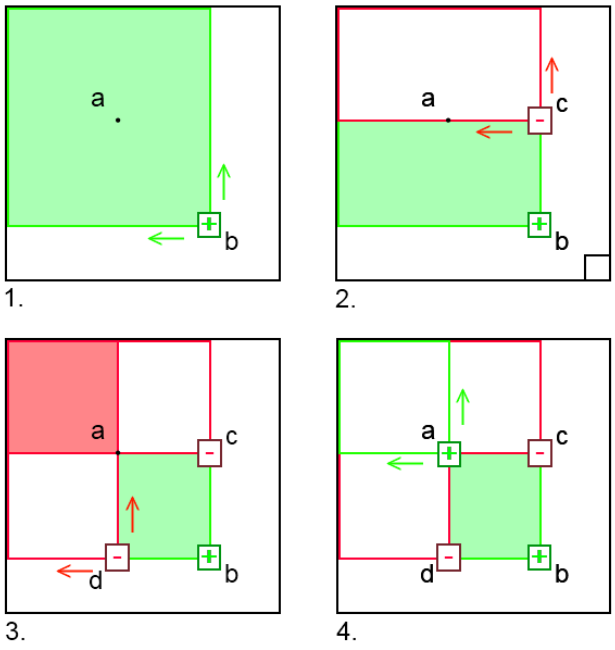


Figure II.27: SAT : calcul de la valeur d'un rectangle
1. :Ajout de la valeur en positif au point b, qui contient la somme de toutes les valeurs de la zone verte.
2.:Soustraction de la valeur au point c, ce qui va éliminer le rectangle au-dessus du point c sur l'axe y.
3.:Soustraction de la valeur au point d, éliminant la zone à gauche de d, mais créant une zone d'influence négative (en rouge) ou les valeurs ont été soustraites deux fois.
4. :Ajout de la valeur au point a, afin d'annuler la zone négative et ainsi compléter le rassemblement. La valeur finale est bien la somme de la zone en vert sans prendre en compte le reste de l'image.

Le point délicat de l'algorithme est le calcul préalable de cette SAT. À partir de l'image originale, nous devons créer la SAT avant que le reste de l'algorithme ne puisse en utiliser les valeurs. Sur le CPU, le calcul est très simple et augmente linéairement avec la taille de la texture. Le CPU profite de la structure linéaire du calcul pour réutiliser les pixels déjà

33 P.S. Heckbert, « Filtering by repeated integration », *ACM SIGGRAPH Computer Graphics* 20, nº 4 (1986): 315–321.

34 Justin Hensley et al., « Interactive Summed-Area Table Generation for Glossy Environmental Reflections », s. d.

35 J. Hensley et al., « Fast Summed-Area Table Generation and its Applications », in *Computer Graphics Forum*, vol. 24, 2005, 547–555.

36 Andrew Lauritzen et Michael McCool, « Summed-Area Variance Shadow Maps », *GPU Gems 3* (2007): 157–182,.

37 T.J. Kosloff, M.W. Tao, et B.A. Barsky, « Depth of Field Postprocessing For Layered Scenes Using Constant-Time Rectangle Spreading », *Graphics Interface Conference* (2009).

calculés. L'algorithme fonctionne en calculant les pixels un à un le long d'une ligne puis la suivante. Il commence en haut à gauche, les valeurs en dehors de la texture (comme la ligne avant la première ligne) sont considérées comme nulles.

Le principe général de création d'une Summed Area Table à partir d'une image calcule un pixel en fonction de trois pixels déjà calculés et de la valeur du pixel dans l'image initiale.

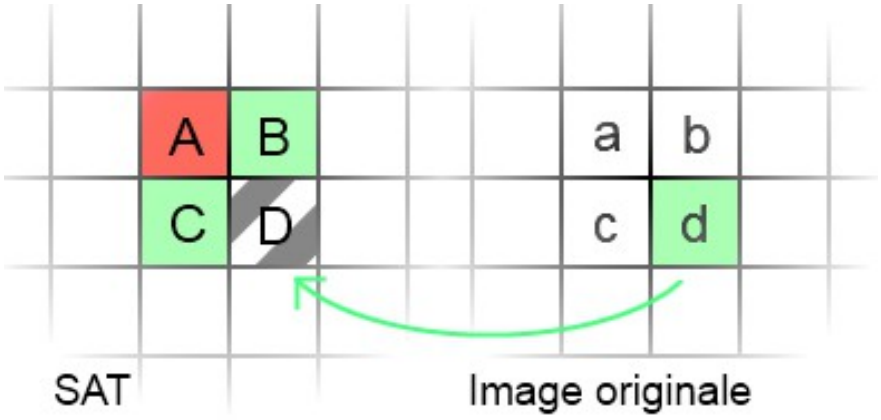


Figure II.28: Calcul itératif d'une SAT
D = B + C – A + d
A, B et C sont des valeurs de la SAT déjà calculées
d est l'intensité du pixel dans l'image initiale
D est la valeur du pixel à stocker dans la SAT

Pour calculer la valeur d'un pixel donné, il est ainsi nécessaire d'avoir calculé les valeurs de tous les pixels précédents. C'est ce qui rend cette technique inadéquate pour les cartes graphiques, car elles utilisent des techniques de calcul massivement parallèle et donc ne peuvent connaître les valeurs des pixels précédents, car ils sont calculés en même temps. Il existe néanmoins des techniques de calcul de SAT sur la carte graphique, mais plus complexe que l'algorithme linéaire. Le calcul doit se décomposer en plusieurs passes ($\ln(n)/\ln(2)$) et chaque passe va additionner la valeur d'un voisin de plus en plus éloigné. Après la dernière passe, chaque pixel contiendra la somme de tous les pixels situés dans le rectangle à gauche et au-dessus de lui, la SAT est complète.

5.1.b Gestion des bords

Les positions proches des bords posent problème lorsque nous cherchons à connaître la moyenne des valeurs d'un rectangle. En effet, dans ces zones là, un ou plusieurs des coins risquent de déborder de la texture. Nous devons alors limiter notre rectangle pour qu'ils ne débordent pas, mais l'aire du rectangle n'est alors plus la même, ce qui fausse les calculs et assombrit les bords. La solution correcte est de calculer la nouvelle aire du rectangle une fois

réduit aux bords, au lieu d'utiliser l'aire supposée à partir de la taille de filtre désiré. Comme ces calculs peuvent être fastidieux et détruire les performances (notamment dans le cas d'un shader GPU), la solution la plus simple est d'ajouter des bords de remplissage (« padding ») sur les quatre côtés (en copiant par exemple en miroir la même image). Le « padding » est une zone qui appartient à l'image, mais qui ne sera pas visible dans l'image finale, car située en dehors des bords. Cela permet de masquer les artefacts indésirables liés aux bords dans la zone de « padding » en sacrifiant un peu de performance dans le calcul de ces zones. La taille de la zone de « padding » doit correspondre à la taille du filtre maximal divisée par deux, ce qui nous assure que les artefacts n'en déborderont pas. Un inconvénient majeur de la technique du « padding » est que la complexité de l'algorithme est de nouveau liée à la taille du filtre, mais de manière beaucoup moins extrême, ce qui la rend praticable.

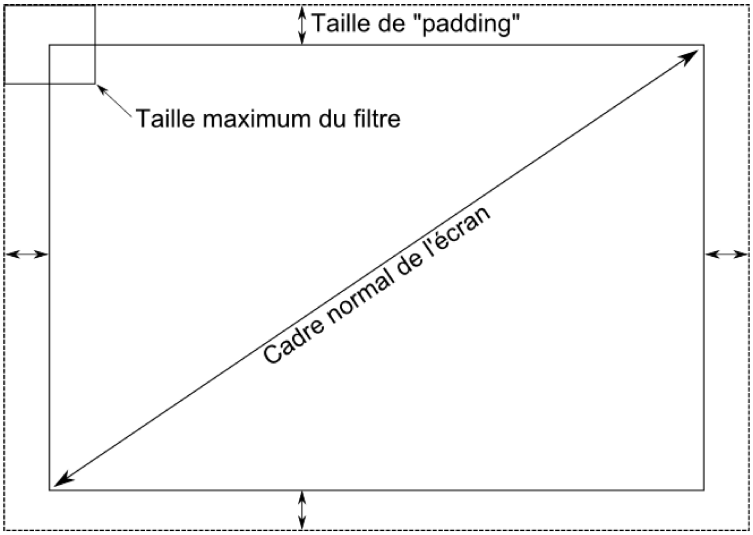


Figure II.29: Ajout d'une zone de padding tout autour de l'écran. L'épaisseur de cette zone correspond à la moitié de la taille maximum de la fenêtre de flou.

5.1.c Précision d'une Summed Area Table

Une autre difficulté qui va limiter fortement l'utilisation des SAT est le grand besoin en bits par pixel. En effet, une texture classique de taille n par m aura par exemple besoin de 8 bits par composant (donc 3×8 bits pour du RGB) pour avoir une qualité classique, mais la SAT qui y correspond aurais besoin en théorie de $8\times n\times m$ bits par composant pour gérer le fait de pouvoir additionner dans un seul pixel toutes les valeurs des autres pixels. En pratique, plusieurs astuces peuvent y remédier. Si le format est en flottant, la première astuce est de recentrer les valeurs autour de zéro, en gérant donc des valeurs négatives. Ainsi, globalement, certains pixels augmentant la somme et d'autres la diminuant, l'amplitude moyenne ira moins

haut et donc perdra moins en précision. Le plus simple est de simplement soustraire 0.5, mais le plus efficace est probablement de calculer la moyenne des valeurs de l'image et de soustraire celle-ci à toutes les valeurs. Cette dernière technique permet de s'assurer que non seulement la valeur dans le coin supérieur gauche est près de zéro, mais aussi que le coin inférieur droit est exactement à zéro (puisque c'est la moyenne de toutes les valeurs multipliée par l'aire). Une dernière astuce consiste à faire ses calculs à partir du centre de l'image, et non du coin. C'est un peu comme si nous divisions la texture en 4 sous-textures chacune ayant sa propre SAT, ce qui multiple par 4 la précision. Malheureusement, les calculs pour connaître la somme d'un rectangle sont alors plus complexes, car il faut gérer les quatre coins différemment. Nous verrons plus bas que l'utilisation de la technique par dispersion diminue grandement ce problème de précision.

5.2 Les Summed Area Table par dispersion

5.2.a Inversion de l'algorithme SAT : une méthode de dispersion

Les SAT présentent un autre aspect très intéressant : nous pouvons complètement inverser l'algorithme pour obtenir, à la place d'un « rassemblement » une « dispersion » (« Spreading » ou « scattering » en anglais). C'est une manière très efficace de calculer une méthode par dispersion.

La première étape est de calculer les quatre positions du rectangle sur lequel nous souhaitons faire notre dispersion, et y inscrire la valeur issue de l'image de base comme ceci : en positif sur les coins a et b et en négatif sur les coins c et d. Ensuite, nous appliquons l'algorithme habituel de transformation d'une image en Summed Area Table sur ce résultat, et nous obtenons tout simplement notre image finale. Chaque rectangle dont nous avons ajouté un coin dans la texture va s'étendre grâce à l'algorithme de SAT jusqu'aux autres coins du rectangle. Nous avons donc une dispersion en temps et accès mémoire constant au regard de la taille du filtre.

La symétrie entre les deux algorithmes (par rassemblement et par dispersion) est très intéressante : dans le premier cas, nous commençons par fabriquer la SAT puis nous allons chercher les valeurs des quatre coins, alors que dans le second cas, nous commençons par insérer les valeurs dans les quatre coins et la SAT est fabriquée ensuite.

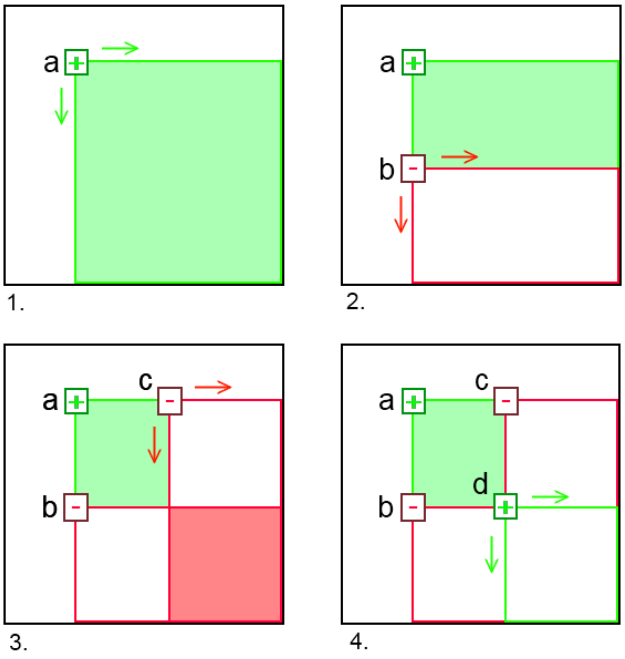


Figure II.30: SAT : dispersion sur un carré
1. : Insertion de la valeur en positif au point a, qui va être propagée sur toute la zone verte par l'algorithme de SAT.
2.:Insertion de la valeur en négatif au point b, ce qui va annuler la propagation du point a au-delà du point b sur l'axe y.
3.:Insertion de la valeur en négatif au point c ce qui annule la propagation sur x, mais crée une zone de propagation négative (en rouge).
4. :Insertion de la valeur en positif au point d, afin d'annuler la zone négative et ainsi compléter la dispersion. La valeur a bien été dispersée sur un carré sans influencer le reste de l'image.

Nous avons vu que la grande différence entre la méthode par dispersion et celle par rassemblement intervient dans la décision de la taille du filtre. Dans le cas de la dispersion, la taille du filtre est uniquement liée à chaque pixel de l'image d'origine qui pourra décider de sa taille. Dans le cas du rassemblement, c'est au contraire le pixel dans l'image finale qui va décider de cette taille et aller chercher des pixels de l'image d'origine pour y contribuer. Le cas du superéchantillonnage (supersampling) est par exemple adapté à la méthode par rassemblement, car nous cherchons à connaître la somme de tous les pixels situés dans un rectangle donné de l'image finale. Dans le cas de la profondeur de champ, par contre, c'est la profondeur du pixel d'origine qui doit servir à définir les pixels sur lesquels il va se disperser. La méthode par dispersion est alors bien plus réaliste. Elle évite notamment tous les artefacts habituels des algorithmes de DOF par rassemblement, où les bords des zones nettes bavent sur les arrières plans floutés.

5.2.b Précision d'une SAT par dispersion

La méthode par dispersion possède un autre avantage. Le nombre de bits nécessaire pour encoder correctement les valeurs est moins haut qu'avec la méthode par rassemblement. En

effet, chaque pixel stockera uniquement la somme des valeurs de ses voisins qui l'influence. Les contributions positives et négatives des voisins vont ainsi se compenser et atteindre une valeur totale raisonnable. Le nombre de bits de précision nécessaire dépend ainsi de la taille maximum du filtre au lieu de dépendre de la taille totale de l'image. En général, nous insérons les valeurs divisées par l'aire du filtre, afin d'assurer la normalisation des valeurs, ce qui montre que la précision dépend bien de la taille du filtre. Il est possible d'appliquer les mêmes techniques qu'avec la méthode par rassemblement, c'est à dire recentrer les valeurs autour de la moyenne afin d'avoir autant de valeurs négatives que positives et donc avoir des sommes par pixel qui restent suffisamment basses pour pouvoir être encodées par exemple dans un flottant de 16 bits. Avec cette technique, nous avons également besoin d'un format de pixel signé. L'image finale sur la carte graphique pourra éventuellement être en format non signé et avec une précision faible, mais tous les calculs internes doivent être faits dans une précision correcte.

Comme la plupart des algorithmes de dispersion, il s'adapte mal au calcul sur la carte graphique, car les techniques d'accès en écriture y sont spécifiques et limités. Ainsi cette version de l'algorithme est destinée au CPU (ou bien à un processeur dédié au calcul vectoriel comme les SPU de la PS3).

5.3 Summed Area Table hexagonale

5.3.a Faiblesse de la forme rectangulaire des SAT

Le grand désavantage des SAT est la restriction de la forme du filtre à celle d'un rectangle. Le filtre obtenu est un « box filter » qui laisse des rebords carrés très visibles et produit un aspect rectiligne désagréable. Nous pouvons néanmoins approcher d'autres formes en cumulant plusieurs rectangles. Ainsi, nous pouvons approcher une forme ronde en insérant des rectangles sur tout le pourtour du rond. La complexité de l'algorithme dépend alors du périmètre du filtre, ce qui est bien mieux (pour de grandes tailles de filtres) que de dépendre de l'aire du filtre comme la plupart des algorithmes. La complexité de l'algorithme est donc linéaire ($n \times 2 \times Pi$) au lieu d'être quadratique (n^2). Il existe également d'autres filtres qui peuvent être simulés, notamment des filtres gaussiens, en effectuant

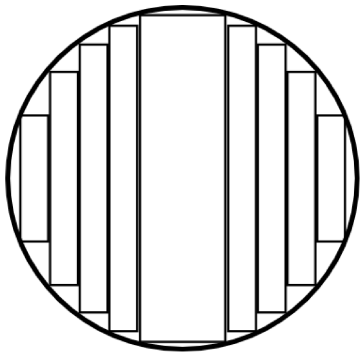


Figure II.31: Approcher un cercle par des rectangles

plusieurs passes successives de l'algorithme de la SAT[38], le tout pouvant être adapté sur la carte graphique[39]. En effet, l'application d'un flou carré (box filter) deux fois de suite va produire un flou dont le poids de dispersion diminue linéairement avec la distance. C'est un flou triangulaire. Une troisième application va former une dispersion suivant une courbe de Bézier (B-spline) uniforme, qui approxime bien une gaussienne.

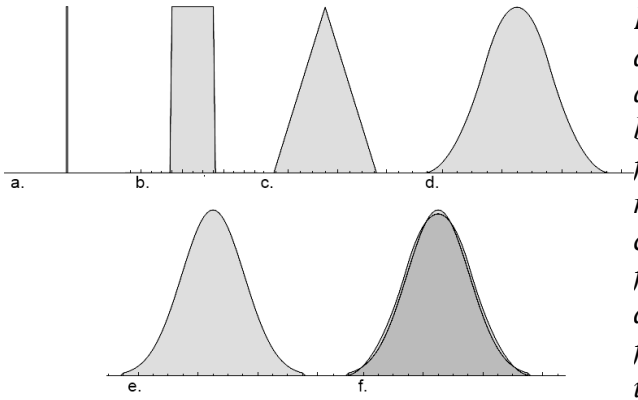


Figure II.32: Calcul d'un filtre gaussien par des box-filter successifs.
a. Valeur initiale, une seule valeur non nulle.
b. Box filter 1X : dispersion de a. par un box filter, produisant une dispersion rectangulaire.
c. Box filter 2X : dispersion de b. par un box filter, produisant une dispersion triangulaire.
d. Box filter 3X : dispersion de c. par un box filter, produisant une dispersion Bézier uniforme.
e. Gaussienne : dispersion de a. par un filtre gaussien
f. Comparaison du Box filter 3X avec la gaussienne, les deux courbes sont similaires.

5.3.b Hexagonal SAT par dispersion

La forme hexagonale présente de nombreux avantages par rapport à la forme carrée lorsqu'il s'agit d'un filtre visant à simuler le flou de profondeur de champ. L'hexagone est bien plus proche d'une lentille ronde et d'un diaphragme de caméra. Si le diaphragme est constitué de six lames, la dispersion de la lumière suivra effectivement une forme hexagonale.

Il est donc intéressant de remplacer la forme rectangulaire de la SAT par une forme hexagonale afin de pouvoir disperser directement sur cette forme. Notre hypothèse de départ est que l'hexagone est une forme suffisamment régulière pour qu'il soit possible de la générer en se basant sur l'addition de plusieurs SAT. En fait, deux SAT mises à jour différemment et séparément puis additionnées suffisent à générer une forme hexagonale.

La première SAT va être mise à jour en quinconce dans un sens (penché vers la gauche) et la seconde va l'être dans le sens inverse (vers la droite).

38 Peter Kovesi, « Arbitrary Gaussian Filtering with 25 Additions and 5 Multiplications per Pixel » (2009).

39 D. Nehab et al., « GPU-efficient recursive filtering and summed-area tables », *ACM Trans. on Graphics (SIGGRAPH Asia)* 30 (2011): 6.

La phase de dispersion va influencer quatre points dans chacune des textures, avant d'appliquer l'algorithme de SAT en quinconce sur les deux images. L'image finale sera simplement la somme des deux SAT intermédiaires.

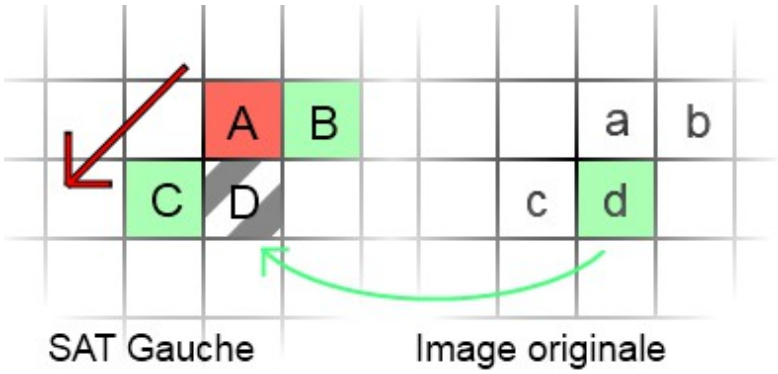


Figure II.33: Création de la SAT en quinconce vers la gauche.
L'équation est toujours D = B + C – A + d

La nécessité d'avoir deux SAT peut paraître lourde, mais en réalité ces deux tables n'ont besoin d'exister que durant un certain moment de l'algorithme, ensuite, les deux sont cumulées et une seule image sera finalement envoyée à la carte graphique.

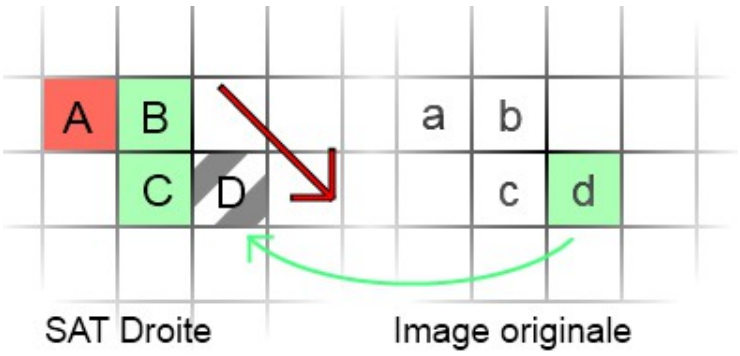


Figure II.34: Création de la SAT en quinconce vers la droite.
L'équation est toujours D = B + C – A + d

De plus, si l'espace de stockage est trop limité, il est tout à fait possible de ne stocker, pour chaque table, qu'un nombre de lignes égal à la taille du filtre maximal, et de faire le cumul dans l'image finale après chaque ligne, au lieu d'attendre la fin du calcul. Ce type de techniques est essentiel pour adapter cet algorithme sur des coprocesseurs, qui ont un espace mémoire très limité, tels que les SPU de la PS3.

Le calcul de l'aire de dispersion dans le cas des bords est plus complexe à gérer avec des hexagones qu'avec des rectangles, car dans le cas des rectangles, les côtés sont tous à angles droits et donc suivent les bords de l'image. Le résultat de l'intersection entre un rectangle et les bords de l'image reste toujours un simple rectangle. Avec un hexagone, le calcul est bien

plus complexe, avec beaucoup de cas particuliers selon l'emplacement de l'hexagone. Le calcul exact serait trop coûteux pour un algorithme qui se veut très rapide. La solution la plus simple est donc d'utiliser la technique du « padding » en ajoutant des bords qui seront invisibles au final. Les artefacts prés des bords seront donc eux aussi invisibles. Cette technique évite de complexifier l'algorithme.

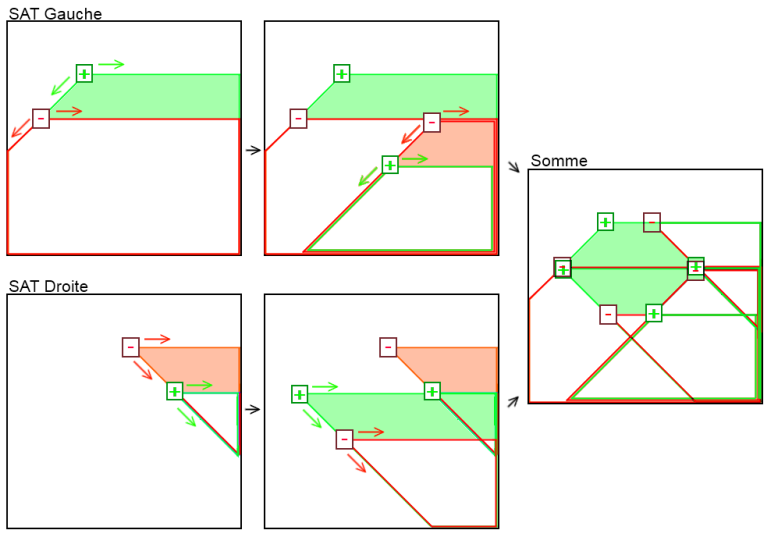


Figure II.35: Dispersion hexagonale grâce à deux SAT. ***SAT Gauche*** *est une SAT qui se propage avec un décalage sur la gauche. Insertion d'une bande horizontale positive puis d'une négative.* ***SAT Droite*** *est une SAT qui se propage avec un décalage sur la droite. Insertion d'une bande négative puis d'une bande positive.* ***Somme*** *est la somme des deux SAT, les quatre bandes insérées se complètent et forment la dispersion en hexagone.*

Pour disperser convenablement la valeur de luminosité d'un pixel sur la surface d'un hexagone, il est nécessaire de calculer l'aire de l'hexagone. Cette aire va servir à diviser la luminosité originale afin d'obtenir une conservation de l'énergie dans la dispersion du pixel et qu'ainsi la luminosité globale de l'image ne change pas une fois le flou appliqué. L'aire d'un hexagone régulier de hauteur minimale n est : $\frac{\sqrt{3}}{2} \cdot n^2$

Cependant, dans notre application, nous avons certaines contraintes sur les tailles des parties de l'hexagone, ce qui en fera notamment un hexagone non régulier. La première contrainte est que nous devons avoir une hauteur discrète puisque nous allons devoir modifier des pixels à des positions précises. La demi-hauteur de l'hexagone sera nommée h. Nous utilisons la demi-hauteur comme valeur discrète de référence, car nous souhaitons conserver le centre de l'hexagone fixe, et donc avoir la même demi-hauteur en haut et en bas de l'hexagone. La seconde contrainte concerne la taille des deux côtés alignés sur l'axe horizontal, qui elle aussi doit être discrète. De même que pour la demi-hauteur, nous utiliserons la demi-largeur comme valeur discrète afin de conserver l'alignement de l'hexagone. La dernière valeur qui serait normalement modifiable est la largeur des triangles extérieurs de l'hexagone. Cette largeur,

que l'on peut nommer t est en fait obligatoirement la même que la demi-hauteur n du fait du calcul des deux SAT. En effet, les SAT obliques utilisées dans notre algorithme propagent les valeurs en ligne droite sur l'axe X et avec un angle à 45 degrés sur l'axe Y, soit vers la gauche soit vers la droite de l'image selon la SAT concernée. Avec un angle à 45 degrés, la hauteur et la largeur du triangle rectangle sont forcément les mêmes.

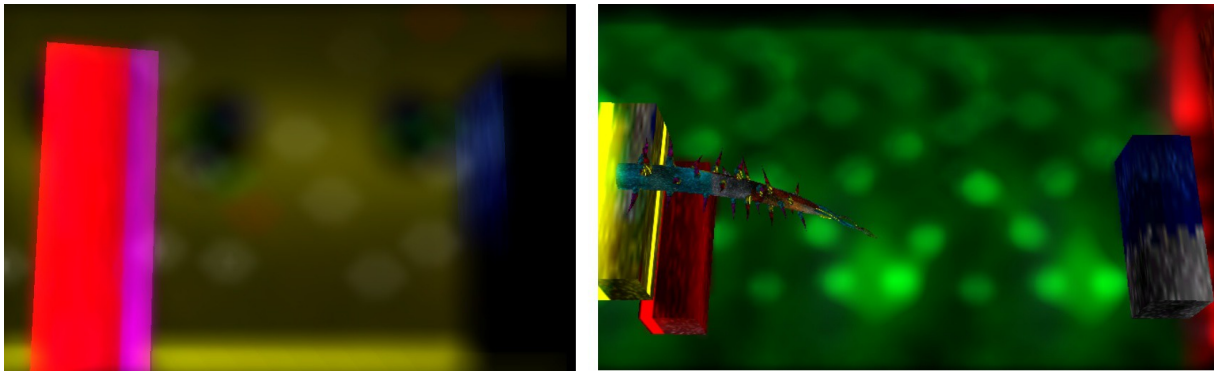


Figure II.36: Exemples de profondeur de champ par SAT hexagonale.
Ce premier prototype utilise des couleurs très saturées et contrastées pour permettre une meilleure identification des artefacts indésirables.

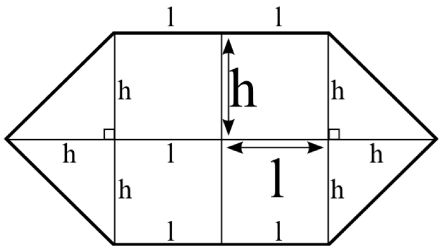
Avec nos deux paramètres discrets, la demi-hauteur h et la demi-largeur l, le calcul de l'aire de l'hexagone peut se décomposer comme celle d'un carré plus quatre triangles rectangles avec la formule :

$$Aire\_hexagone = 2h \times 2l + 4 \times 1/2 \times h \times h$$

$2h \times 2l$ *est l'aire d'un carré de taille 2h par 2l*
$1/2 \times h \times h$ *est l'aire d'un triangle rectangle isocèle de taille h*

Une fois simplifié :

$$Aire\_hexagone = 2 \times h \times (2 \times l + h)$$



Le calcul de l'aire de l'hexagone est ainsi simple et diviser les valeurs par cette aire avant l'insertion dans les SAT permet de normaliser le résultat.

Figure II.37: Second prototype de test de l'hexagonal SAT.
À gauche, le point est fait à l'infini et l'avant-plan est flou, alors qu'à droite c'est l'inverse.

5.3.c Taille de filtre décimale

Lorsque la taille du filtre doit changer, soit spatialement dans l'image, soit au cours du temps, il est nécessaire d'avoir une transition douce entre les différentes tailles possibles. Avec l'algorithme de dispersion de rectangles par SAT, la dispersion est effectuée sur des tailles de filtre discrètes. Pour étendre cela aux tailles de filtres décimales, nous allons simplement disperser deux rectangles, le second étant plus grand que le premier d'un pixel sur chaque bord. L'opacité des deux rectangles va varier avec la valeur fractionnaire de la taille de filtre. Ainsi un filtre d'une taille de 12.3 va se disperser sur un rectangle de taille 12 avec une opacité de 0.7 et un rectangle de taille 13 avec une opacité de 0.3. Pour une taille de 12.9, le rectangle de 12 sera opaque à 0.1 et le 13 à 0.9.

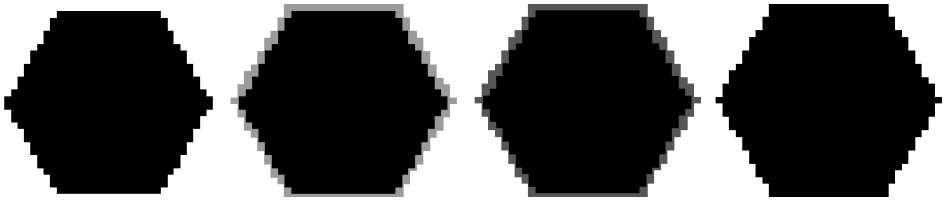


Figure II.38: Filtre hexagonal : variations inférieures à l'échelle du pixel. Le passage entre deux tailles successives est obtenu en variant l'opacité de deux hexagones.

Le résultat est correct même si par la même occasion les bords de la forme deviennent plus flous, ce qui n'est pas forcément gênant, car l'image parait également plus naturelle.

La même opération peut être effectuée sur les hexagones, même si elle est un peu plus délicate, car un hexagone voit sa taille définie par deux nombres entiers au lieu d'un seul pour le carré : la taille du haut de l'hexagone et la taille au milieu de l'hexagone. Ainsi, pour passer d'une taille discrète à la suivante, l'hexagone va alternativement gagner un pixel en hauteur, puis un pixel en largeur, et ainsi de suite. Une solution plus correcte serait d'influer sur les

points de quatre hexagones afin de gérer tous les cas possibles (l'hexagone régulier de départ, celui de la taille finale, et les deux extensions intermédiaires non régulières possibles), mais l'algorithme actuel fonctionne déjà convenablement sans le surcoût de la diffusion de deux autres hexagones supplémentaires.

Figure II.39: Détails de résultats d'un flou par SAT hexagonale.

Il est possible également d'appliquer la forme hexagonale dans le cas de l'algorithme par rassemblement. Une fois que les deux SAT orientées ont été générées depuis l'image initiale, connaître la somme de n'importe quel hexagone revient simplement à prendre 4 échantillons dans chacune des deux images, et à les additionner correctement.

5.4 Intégration au rendu de la scène

5.4.a Gestion des hautes résolutions

Pour obtenir la meilleure qualité de flou de profondeur de champ, le plus simple est d'appliquer l'algorithme sur l'ensemble de l'image haute résolution. La qualité sera très élevée, mais la taille de l'image en haute résolution pose problème. Un jeu classique sur une console « next-gen » tournera au minimum en 720p, c'est-à-dire 1280×720 pixels soit près d'un million de pixels, sans parler des jeux en « full HD » 1080p : 1920×1080 pixels soit plus

de deux millions de pixels. C'est une résolution qui implique que l'algorithme sera lent, mais surtout, dans notre cas, que le transfert qui est fait depuis le rendu stocké sur la carte graphique (contenant l'image finale juste avant le DOF) jusqu'à la mémoire accessible depuis le CPU va être très long, de même que le trajet inverse pour l'image finale après DOF.

Ainsi, en général, nous souhaitons transférer une version plus petite de l'image, 1/4 voir 1/16 de la taille. Dans le cas du DOF, la perte de résolution n'est pas normalement un gros problème, car elle sera masquée par le flou du DOF. Par contre, cela veux dire qu'il va falloir combiner, sur le GPU, la version réduite puis floutée par le DOF avec la version haute résolution qui est toujours disponible sur la carte graphique.

5.4.b Création de la version « downsampled »

À partir de l'image pleine résolution qui a été rendue en utilisant la géométrie et les lumières de la scène, nous allons créer une version avec une résolution plus faible, souvent avec un facteur en puissance de deux pour faciliter les calculs et limiter les cas particuliers. Ainsi, une diminution d'un facteur de deux sur chaque dimension mènera à une diminution par quatre du nombre de pixels. Chaque pixel de la petite texture représentera quatre pixels de la texture haute résolution. Une simple diminution de la résolution par décimation, c'est-a-dire l'utilisation d'un seul des quatre pixels de la résolution supérieure mènerait à un crénelage spatial et temporel très important. Les détails de l'ordre du pixel dans l'image haute résolution serait aléatoirement décimés ou conservés d'une image sur l'autre, faisant scintiller l'image basse résolution.

Dans le cas de l'utilisation pour un filtre de flou de profondeur, il est de plus nécessaire de connaître la profondeur de chaque pixel, ou tout au moins la valeur de CoC, usuellement stockée dans le canal alpha de l'image. Un simple flou entre les valeurs des quatre pixels de l'image haute résolution est intéressant, mais le mélange des valeurs de profondeur, ou de CoC peux mener à des valeurs absurdes et indésirables. Ainsi, dans le cas de l'intersection entre un objet en avant plan et un autre en arrière plan, certains pixels auront une grande profondeur et certains autres une très faible, le mélange étant de profondeur moyenne alors même qu'aucun pixel réel n'est présent à cette profondeur moyenne, causant une mauvaise estimation du flou à appliquer à cet endroit.

Dans le cas de la profondeur de champ, nous souhaitons conserver le plus de données parmi celles avec un fort CoC puisque ce sont celles qui seront utilisées par le calcul de flou. De

plus, nous voulons éviter au maximum de mélanger les pixels avec des CoC différents pour limiter les débordements des parties nettes sur les parties floues. Les pixels avec un faible CoC seront de toute manière pris à partir de la version haute résolution de l'image, après que le flou ait été calculé sur la version basse résolution. Ainsi, il est intéressant de conserver, parmi les quatre pixels de la version HD, le pixel qui a le plus fort CoC puis de mélanger les couleurs des trois autres pixels en fonction de la différence de CoC avec le pixel choisi. Grâce à cette méthode, des pixels à la même distance seront effectivement mélangés, mais en séparant le plus possible les différentes zones de netteté de l'image pour éviter les débordements des uns sur les autres.

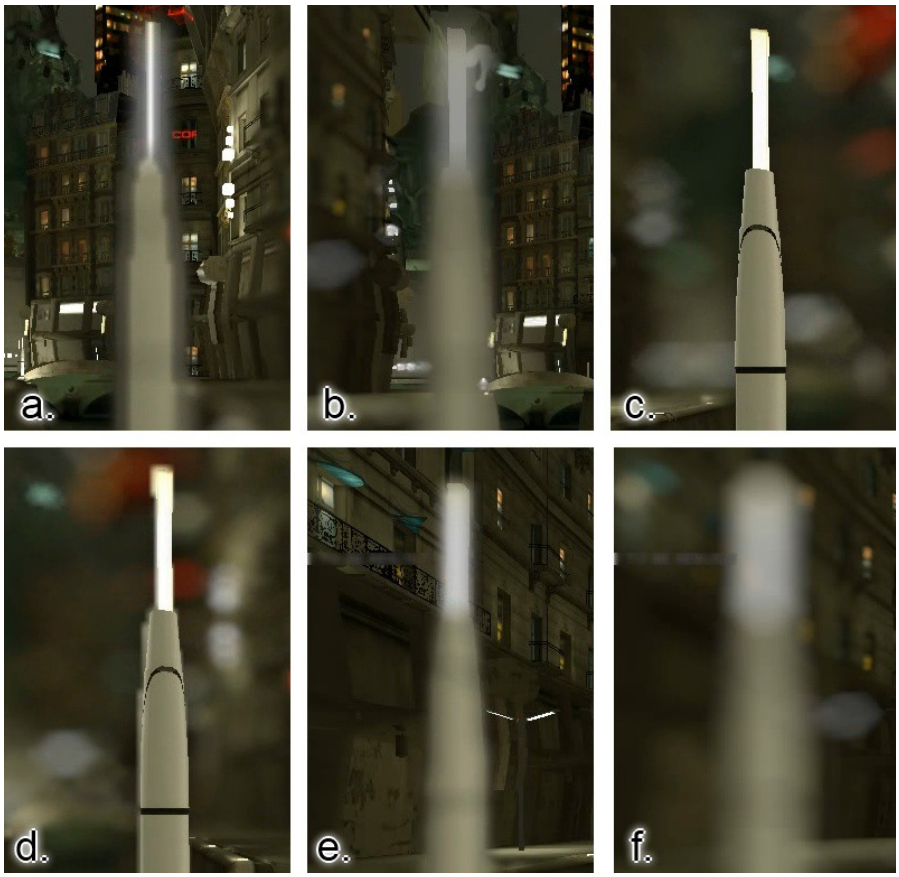


Figure II.40: Exemples d'artefacts liés au mélange des couches et aux différences de résolution.
Le mélange de couches de résolutions différentes produit des bords disgracieux (a. et d.). La couche de flou d'avant plan présente souvent des bords trop durs (b. et e.). Le fond flou devient plus sombre aux abords de l'avant-plan net si le flou n'est pas normalisé (c.). En comparaison, l'avant-plan flou de référence semble bien plus naturel (f.).

5.4.c Mélange de couches avec résolutions différentes

Pour économiser des calculs, le flou va être effectué dans une résolution plus petite. Nous avons donc l'image originale et nette à la résolution finale, accompagnée de l'information de CoC par pixel, et dans une texture plus petite, nous avons la version floutée de l'image, elle aussi avec une information de CoC associé. Le but est donc d'utiliser la version réduite floutée pour le fond flou, d'ajouter la version nette haute résolution, et par dessus la partie en avant-plan de la texture réduite flouté.

Pour simplifier le problème, considérons que le flou au loin et le flou au près (avant et après le point de focus) ont été calculés dans deux images différentes. Cela évite les mélanges entre les deux et permet d'utiliser des masques différents pour les intégrer dans l'image finale. Obtenir une profondeur de champ correcte se divise alors en deux étapes : mélanger l'image de fond floutée avec l'image nette puis mélanger celle-ci avec l'image d'avant plan floutée.

Pour mélanger l'image d'arrière-plan floutée et la partie nette, le canal de CoC stocké dans la partie nette est idéal, car déjà à la résolution finale. Grâce à la manière dont nous avons calculé l'image de faible résolution, l'image de l'arrière-plan déborde derrière l'avant-plan net, et le mélange des deux fonctionne très bien. Le calcul est le suivant :

$$DOFArrière = ImageHD \times (1 - CoCNet) + FlouArrièrePlan \times CoCNet$$

Le mélange avec l'image de premier plan flouté est plus complexe. En effet, dans le cas d'une photographie réelle, les bords de l'avant-plan flou sont également flous et l'arrière-plan apparaît derrière. Malheureusement, il nous manque des informations dans l'image nette. Il s'agit des parties de l'arrière-plan qui devraient apparaître derrière l'avant-plan qui est maintenant flouté, donc semi-transparent. Nous allons voir diverses techniques qui cherchent à reconstruire ces informations.

5.4.d Désocclusion de l'image

Nous avons considéré que l'effet du flou de profondeur de champ pouvait être appliqué après le calcul de l'image nette, en n'utilisant que les informations contenues dans cette vue de la scène. Dans la réalité, ce n'est pas le cas. En effet, le flou de profondeur est le résultat de la multitude de trajets que les rayons peuvent emprunter pour atteindre un pixel donné de l'image or le résultat d'un seul trajet par pixel peut être stocké dans notre image nette. Il existe

d'ailleurs des prototypes de caméras réelles qui capturent la direction des points lumineux au lieu d'une simple image, afin de pouvoir choisir les paramètres du flou de profondeur de champ après la prise de vue (les appareils photographiques plénoptiques[40]).

Dans la plupart des cas, les trajets alternatifs des rayons concernant un pixel sont en fait empruntés par les pixels voisins, c'est pourquoi nous pouvons utiliser un flou pour simuler cette contribution entre voisins. Il existe par contre un cas où ces rayons alternatifs ne sont pas disponibles chez les voisins : lors d'un gradient important dans le CoC. En effet, les bords d'un avant plan qui doit être flouté vont sembler devenir transparents et donc laisser apparaître les pixels qui sont derrière. Nous n'avons pas d'information sur ces pixels, c'est pourquoi la plupart des algorithmes de flou temps réel laissent des bords durs aux avant-plans hors focus.

Les pixels qui sont manquants ont néanmoins de grandes chances de ressembler aux pixels adjacents de l'arrière-plan. Utiliser ces informations pour reconstruire heuristiquement les données manquantes se nomme une « désocclusion ». Ce domaine du traitement de l'image concerne principalement la restauration d'image et le truquage. Il est ainsi possible de supprimer un personnage d'une image et de reconstruire l'arrière-plan, ou bien de supprimer un texte qui gêne la lecture d'une image. Dans le cas de la profondeur de champ, nous construisons tout d'abord une image en conservant uniquement les pixels qui sont plus loin que le début du flou en avant plan. Les pixels qui sont trop proches de la caméra et qui devraient donc être floutés sont tout simplement supprimés. L'algorithme de désocclusion est alors appliqué et il reconstruit les parties masquées par l'avant-plan à partir des données de l'arrière-plan. Dans le domaine du traitement d'image précalculée, il existe des méthodes complexes qui comblent les trous d'une image en reproduisant non seulement les couleurs voisines, mais également les textures et les contours, ce qui permet une très grande qualité de reconstruction. Masnou et Morel[41] présentent ainsi une méthode basée sur les « level lines» qui peuvent interpoler correctement les contours d'une image afin de reconstruire une grande zone d'occlusion. Bertalmio et Vesa[42] s'attachent eux plus particulièrement à détecter et reproduire les textures d'une image. Ces techniques ne conviennent pas aux contraintes du temps réel actuel pour des raisons de performance. De plus, le calcul du flou de profondeur de

40 Ren Ng, « Digital Light Field Photography », *Stanford University*, Thèse (2006).

41 Simon Masnou et Jean-Michel Morel, « Level Lines Based Disocclusion », *Proc. Int. Conf. Image Processing,* (1998): 259-263.

42 M. Bertalmio et al., « Simultaneous structure and texture image inpainting », *Image Processing, IEEE Transactions on* 12, n° 8 (2003): 882–889.

champ ne nécessite pas forcément une désocclusion très complexe, car les trous à boucher sont généralement assez petits et seront en partie masqués par l'avant plan flou. Les techniques basées sur le calcul d'une pyramide de filtrage[43] sont simples et efficaces, mais présentent tout de même un surcoût.

L'étape de désocclusion est évitée dans certaines recherches[44] en découpant le rendu en un nombre conséquent de couches, découpées selon la profondeur des pixels, ce qui permet de stocker un maximum de l'information normalement occultée. Chaque couche est floutée avec une taille de flou uniforme, ce qui diminue la complexité du filtre. Néanmoins, la gestion d'objets faisant partie de plusieurs couches est complexe, les transitions d'une couche à la suivante n'étant pas toujours aisées. La séparation en couches impose de plus un surcoût sur le calcul de l'image de base qui est prohibitif pour la plupart des applications temps réel lourdes comme le sont les jeux vidéo.

Nous avons cherché une technique qui soit la moins lourde possible et qui utilise les caractéristiques de la profondeur de champ. Le point de départ de la technique concerne la gestion des bords des objets nets en avant plan. En effet, le fond flou à tendance à devenir sombre à proximité d'un avant plan net, car celui-ci masque une partie de l'arrière-plan qui ne contribue donc pas aux pixels du fond. Pour compenser cette perte de luminosité, nous effectuons une renormalisation. Nous devons connaître, à un point donné, le nombre de contributions qui y a été apporté par l'opération de flou. Dans ce but, nous utilisons le canal alpha de l'image, qui sera flouté en même temps que l'image de base. Nous y stockons au préalable une valeur de 1 pour tous les pixels. Après dispersion, les endroits complètement flous conservent une valeur de 1, mais les endroits où la valeur est inférieure à 1 ont besoin d'être réajustés, car ils n'ont pas reçu assez de contributions. Pour cela, il suffit de diviser la couleur floutée par la valeur alpha floutée. Les bords ne perdent plus leur luminosité.

Cette technique peut s'étendre facilement à la désocclusion des parties cachées. Il s'agit de propager les pixels de l'arrière-plan par le flou, mais sans prendre en compte l'avant-plan. Pour cela, nous allons de nouveau utiliser la valeur alpha, mais cette fois, au lieu de stocker la valeur 1 nous allons stocker la valeur de CoC. Ainsi, les pixels hors focus auront une valeur de 1 et ceux en focus auront une valeur de 0. Les couleurs de l'image sont également

43 M. Strengert, M. Kraus, et T. Ertl, « Pyramid methods in gpu-based image processing », *Proceedings Vision, Modeling, and Visualization 2006* (2006): 169–176.

44 B.A. Barsky et al., « Elimination of artifacts due to occlusion and discretization problems in image space blurring techniques », *Graphical Models* 67, n° 6 (2005): 584–599.

prémultipliées par cette valeur alpha. Ainsi, une fois la dispersion faite, nous pouvons diviser la valeur de couleur par la valeur alpha pour normaliser l'image en ne prenant en compte que les éléments hors focus. Dans le cas de la séparation du flou en deux images différentes, la partie concernant l'arrière-plan va représenter le fond flou qui déborde sur les parties normalement nettes. L'image d'avant-plan contiendra les objets flous positionnés en avant. Pour éviter que les parties de l'image trop nettes ne deviennent incohérentes, nous limitons le CoC stocké dans la valeur alpha pour qu'il ne soit jamais à zéro, afin d'être toujours en mesure de diviser la couleur par la valeur alpha et retrouver une couleur. Cette technique est très simple à mettre en place et ne coûte pratiquement rien en terme de performance, puisqu'elle exploite le flou qui est de toute manière effectué. Elle est également très efficace, en tout cas lorsqu'il s'agit de deux images de flou séparées.

Une fois la désocclusion appliquée sur l'image haute définition, ou bien sur une version plus basse résolution, puis recomposée avec l'image haute définition, l'intégration de l'avant plan flouté est simple et repose sur le canal alpha, contenant la valeur de CoC. Ce canal est flouté en même temps que l'avant-plan et sert de masque pour faire apparaître la version floutée :

$$DOFAvant = HDDésoccludé \times (1 - CoCFlou) + AvantPlanFlouté \times CoCFlou$$

Malheureusement, la séparation en deux flous distincts que nous avons pris comme hypothèse de travail, un flou pour l'arrière-plan et un autre pour l'avant-plan, n'est pas conciliable avec le temps réel, car il double le temps de calcul. De plus dans notre cas, le transfert depuis le GPU vers le CPU peut se faire une seule fois pour les deux flous, mais deux images seront renvoyées au GPU ce qui double la bande passante nécessaire.

La réunification des deux images floutées en une seule n'est pas une étape évidente, nous n'avons pas trouvé de technique fonctionnant dans tous les cas. Les situations les plus délicates se présentent lorsque à la fois l'arrière-plan et l'avant-plan sont flous dans la même image, et surtout lorsqu'ils se touchent. Les surfaces continues qui passent du flou au net sont également difficiles à gérer. La transition temporelle entre totalement net et flou présente les mêmes difficultés. Concrètement, il vaut mieux éviter les transitions, et avoir un fond flou et un avant plan net bien tranché, ou bien le contraire, un avant-plan flou et tout le reste net.

5.4.e Intégration des éléments transparents

Depuis le début de notre étude, nous avons considéré chaque pixel de l'image originale comme représentant un élément unique de la scène 3D sous-jacente que nous pouvons donc relier à une distance précise et unique depuis la caméra. Bien que cela soit une hypothèse raisonnable dans le cas des objets opaques, les objets transparents rendent cette simplification fausse. En effet, la couleur d'un pixel semi-transparent dépend de deux objets placés à deux profondeurs différentes, et du degré de transparence de celui qui est le plus proche. Il n'est donc pas possible de définir une seule profondeur pour le pixel, et tous les algorithmes de flou présentés ici échoueront à calculer fidèlement le flou de profondeur. Une solution permet néanmoins de limiter les artefacts liés à ces objets transparents. Il s'agit de calculer directement le CoC au lieu de déduire celui-ci par la profondeur du pixel final. Les objets opaques doivent être dessinés en premier et donnent la valeur initiale au CoC. Ensuite, les objets transparents sont affichés les uns après les autres selon leur distance à la caméra (les plus loin d'abord). La valeur de CoC déjà présente est alors mélangée avec la valeur de CoC de l'objet transparent selon son degré de transparence. La valeur de CoC finale représente alors une moyenne pondérée des valeurs de CoC des différents objets que la lumière a traversé. Le reste du calcul du flou de profondeur peut alors s'effectuer normalement à partir de cette nouvelle valeur de CoC. Le surcoût de cette technique provient essentiellement du fait qu'il est nécessaire d'avoir un mode de fusion de la transparence qui sépare la valeur de la couche alpha de la valeur de transparence du pixel. De plus, beaucoup de moteurs de rendu temps réel utilisent déjà la couche alpha pour stocker la profondeur, et non le CoC. La solution d'implémentation choisie est en général l'utilisation du MRT (Multiple Render Target) qui permet de rendre d'un seul coup des informations dans deux textures. Le rendu des objets transparents va donc donner d'un côté une texture de couleur et de l'autre une texture de CoC. Si cette technique ne donne pas un résultat parfait, le calcul présente néanmoins suffisamment de cohérence avec l'évolution des différents paramètres en jeu (couleurs et profondeurs des pixels successifs) pour offrir une image correcte.

6 Conclusion

Dans cette partie, nous avons pu étudier la simulation en temps réel des effets optiques liés à la profondeur de champ. C'est un des outils importants de l'artiste pour la construction de son image. La profondeur de champ sert à la fois dans la perception de la spatialité de la scène, mais aussi à structurer et à clarifier une image. C'est également un outil servant l'expression esthétique de l'artiste, un support pour la narration et un rappel au médium cinématographique. Par contre, le flou de profondeur de champ a tendance à perturber la liberté du regard, essentiel dans les applications interactives.

À partir de la physique optique des caméras réelles et de l'œil humain, nous avons pu analyser les paramètres importants qui influencent le flou de profondeur de champ. La forme de la lentille (ou « bokeh ») apporte ainsi tout à la fois un photoréalisme et une stylisation de l'image.

Le sujet de la simulation en temps réel des effets optiques liés à la profondeur de champ est toujours d'actualité et chaque génération de matériel apporte de nouvelles possibilités techniques et plastiques. En plus des recherches consacrées aux effets de lentilles, les chercheurs poursuivent les tentatives d'intégration de la profondeur de champ avec les autres effets de réduction du crénelage, qu'il soit spatial ou temporel.

Le rendu pré calculé se contente parfois d'utiliser le lancer de rayon en effectuant des calculs d'optiques réalistes, mais très lents et redondants, acceptant des temps de rendu de plusieurs heures par image. Ils utilisent également des techniques bidimensionnelles, basées sur des équations optiques assez générales, mais lentes, rendant les temps de calcul prohibitifs pour notre application. Au contraire, le rendu en temps réel oblige à déterminer les éléments les plus importants de l'effet visuel et esthétique, en apportant une réflexion sur la meilleure répartition de l'erreur, sur les échanges possibles entre le crénelage et le bruit, et sur les compromis entre qualités et performances.

La technique de la Summed Area Table hexagonale que nous avons créée au cours de cette étude présente de nombreux avantages, notamment l'indépendance de la complexité au regard de la taille du flou désirée et l'utilisation de la dispersion à la place du rassemblement lors de l'application de la forme de la lentille. Cette technique ne sera finalement pas utilisée sur le jeu vidéo de DONTNOD. La technique classique reposant sur un flou Gaussien a été préférée pour ses performances. L'implémentation actuelle des SAT hexagonales fonctionne

uniquement sur le processeur principal ce qui limite les performances et donc la taille de l'image qu'il est possible de traiter dans une application temps réel. L'implémentation sur des coprocesseurs de calcul vectoriel (comme les SPU de la PlayStation 3) permettrait d'atteindre de bonnes performances à pleine résolution. L'utilisation de la carte graphique (à travers les langages avancés Cuda, OpenCL et Compute Shader) est également à envisager.

Chapitre III - Matières, lumières et réflexions

1 Le rôle du travail de la matière virtuelle

1.1 Le rôle de la matière dans les Arts

Dans les arts traditionnels, le travail de la matière est un aspect essentiel de l'œuvre de l'artiste. La sculpture en est l'exemple le plus direct, mais la peinture présente également un travail de la matière. Du choix des pigments jusqu'aux techniques de superposition de couches en passant par le grain de la surface du tableau, le peintre dispose de nombreux leviers pour travailler la matière même de l'œuvre. Le geste du pinceau, au-delà d'une simple recopie de la couleur du modèle, cherche également chez certains artistes à reproduire le grain, la réflexivité et les autres qualités plastiques d'une matière. La superposition de différentes couches de pigments chez Vermeer créée des surfaces qui réagissent à la lumière d'une façon similaire à la matière servant de modèle. Le « sfumato » de De Vinci est également une technique picturale précise qui prolonge la maîtrise du pinceau dans un travail de la matière par couches.

Figure III.1: À gauche, « Fin de collation » de William Claesz (1637) et à droite « La Jeune Fille au verre de vin » de Vermeer (1660)

Les hyperréalistes américains utilisent la peinture comme une photographie, cherchant au maximum la neutralité dans l'image, par le sujet comme par la technique picturale. Les hyperréalistes des premières années se sont beaucoup intéressés aux jeux de réflexions sur les surfaces vitrées. Ils travaillaient généralement d'après des photographies, qui se chargeaient donc d'effectuer le transfert de la scène réelle vers la reproduction en image et en couleurs. La peinture hyperréaliste laisse tout de même entrevoir sa réalité picturale par la grandeur de la toile utilisée, qui permet de s'approcher très près jusqu'à décomposer l'image en coups de pinceaux abstraits.

Figure III.2: Les hyperréalistes, de Richard Estes à gauche (« American Express Downtown » - 1979) à Alyssa Monks à droite (« Smirk » - 2009)

La reproduction « réaliste » de matières peut aller au-delà de cette fidélité au réel que recherchent les hyperréalistes. Il s'agit alors d'augmenter la palette de sensations visuelles et tactiles que le peintre est capable d'évoquer par sa peinture. Chaque type de matière nous évoque des souvenirs et des impressions qui peuvent participer au ressenti global du spectateur devant une œuvre picturale. En étant capable de reproduire et de mélanger les caractéristiques de ces matières, le peintre obtient une liberté qui ne se limite pas au mimétisme de la réalité. C'est au contraire un outil d'émancipation du cadre strict de la reproduction picturale. L'œuvre peinte devient plus qu'une matrice de points d'une couleur définie une fois pour toutes, mais entre en interaction avec l'environnement, les sources lumineuses, la position du spectateur et sa perception visuelle.

1.2 Rôles et intérêts de la reproduction fidèle de la matière

De la même manière, le travail des matières virtuelles présente ces deux aspects : la reproduction fidèle de matières réelles et l'utilisation des modèles de matières mis en place pour imaginer et représenter nos propres matières.

La reproduction fidèle a notamment pour but d'améliorer l'intégration des éléments de la scène entre eux afin qu'ils semblent faire partie naturellement du même univers, du même environnement lumineux, perçu par le même système optique. Les images virtuelles qui ne tiennent pas compte d'un environnement homogène peuvent présenter un aspect « collage ». Pour l'éviter, il est important de créer un repère commun de description des différents éléments de l'image, du modèle de calcul de la réflexion lumineuse jusqu'à celui de la description de la rugosité de la surface.

La compréhension de la scène par le spectateur est généralement améliorée par l'utilisation d'une reproduction fidèle à la réalité. Les matières sont plus aisées à reconnaître si elles réagissent selon les lois du réel. Il est alors plus facile d'identifier leurs rôles et leurs fonctionnements dans le cadre des règles d'un jeu.

L'utilisation de modèles qui ont des caractéristiques physiquement réalistes permet également d'améliorer la création intuitive de matières par l'artiste. Les paramètres et réglages des modèles physiques ont alors une réelle signification qui est parlante pour l'artiste. Cette correspondance entre réalité et modèle virtuel permet d'utiliser des connaissances issues d'un domaine tel que la sculpture ou l'architecture pour concevoir les matières des environnements virtuels. Les simulations numériques des outils des media classiques tels que la peinture (Corel Painter, ArtRage ...) sont par exemple très populaires chez les peintres qui utilisent le numérique. Ces logiciels cherchent à simuler la peinture à l'huile, l'aquarelle ou le pastel en modélisant les interactions entre les pigments, le pinceau et la toile en fonction des gestes du peintre. Les gestes sont capturés par le biais d'une tablette graphique par exemple.

De la même manière, la simulation d'une matière réelle permet de la travailler avec les mêmes types d'outils, de modifier la patine d'une surface, sa rugosité, ses couleurs... Nous pouvons même aller au-delà des matières classiques présentes dans la réalité tout en conservant un lien et une structure, c'est-à-dire un modèle de contrôle qui apporte du sens aux réglages disponibles pour l'utilisateur.

2 Décomposition de l'aspect visuel d'une surface

L'aspect visuel d'une surface dépend de très nombreux paramètres, certains liés à la surface, mais d'autres extérieurs et dépendants de l'environnement proche ou lointain. Le travail de reproduction fidèle nécessite de répertorier, catégoriser et expliquer chacune des interactions qui participent au visuel d'une surface, puis de trouver les règles les plus simples et efficaces possible pour en reproduire les principales caractéristiques.

2.1 Lumière et surface

La surface en elle-même possède des caractéristiques qui permettent de décrire son comportement en réaction à l'environnement extérieur. La majeure partie de ces caractéristiques décrivent la réaction de l'objet à la lumière. C'est en effet par la lumière présente dans la scène que le système visuel prend connaissance de son environnement. La surface elle-même peut produire de la lumière, généralement dénommée « lumière émissive ». Ainsi, un filament chauffé au rouge émettra de la lumière, indépendamment du reste de la scène.

2.1.a La réflexion

L'impact lumineux que notre œil perçoit est en général reflété par un objet, et même souvent par plusieurs objets successifs. Au cours de ce voyage, les composants de la lumière initiale seront filtrés par les différents rebonds sur les objets solides et par les passages à l'intérieur des objets translucides, tels que l'eau ou le brouillard.

Le produit de ce filtrage dépend donc dans un premier temps des caractéristiques de la lumière initiale, telle que la longueur d'onde (c'est-à-dire la couleur), l'intensité, voire même la polarisation dans certains cas extrêmes. Ensuite, chaque objet rencontré filtrera cette lumière selon les pigments qui composent sa matière et la forme de sa surface.

2.1.b Réflexion diffuse et spéculaire

Traditionnellement, les réflexions lumineuses sont réparties en deux catégories : réflexion diffuse et réflexion spéculaire. La première décrit la réflexion de la lumière de manière uniforme dans toutes les directions. La seconde décrit la réflexion de la lumière comme dépendant de l'angle de vue. La lumière est alors reflétée majoritairement dans une direction

particulière, inverse par rapport à la direction de l'impact, en fonction de la normale de la surface.

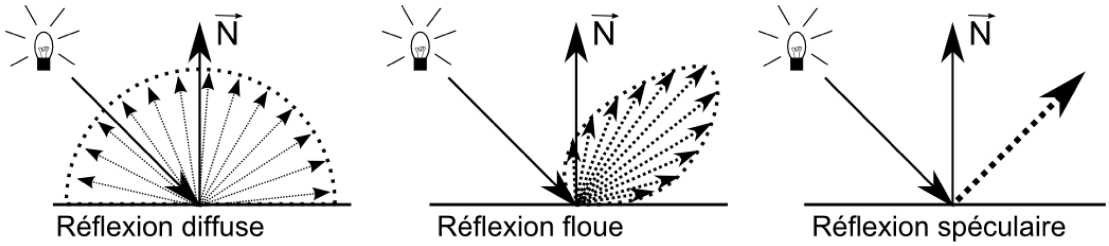


Figure III.3: Les types de réflexion de la lumière.
Dans toutes les directions (à gauche), sur un cône de directions (au centre) et dans une seule direction (à droite).

En réalité, cette division entre lumière diffuse et spéculaire est un outil de simplification, mais ne représente pas deux phénomènes différents, mais deux segments d'un comportement continu. La caractéristique de la matière qui définit principalement le passage d'un comportement diffus à un comportement spéculaire est la rugosité du matériau. Elle représente l'aspect de la surface à une échelle microscopique. Une surface très chaotique aura une rugosité élevée alors qu'une surface lisse aura une faible rugosité. La surface microscopique est généralement modélisée par des microfacettes. Dans ce modèle, chaque microfacette possède une taille, une direction, une hauteur, mais reflète parfaitement la lumière, c'est-à-dire dans une seule direction, reflétée par rapport à la normale de la facette. Les microfacettes ne sont pas représentées individuellement, mais plutôt comme une fonction de probabilité des normales possibles.

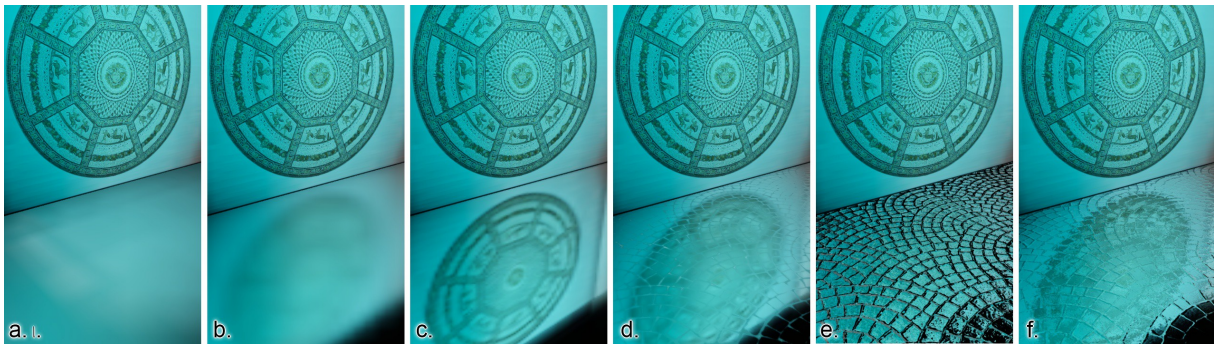


Figure III.4: Les différents types de réflexion en fonction de la rugosité du matériau.
Une réflexion purement diffuse (a.), une réflexion floue (b.) et une réflexion miroir (c.).
Utilisation d'une texture de rugosité (d.), une texture de normale (e.) ou les deux (f.).

L'aspect de la réflexion spéculaire va dépendre de plusieurs éléments. Ainsi, un coefficient appelé « roughness », c'est a dire « rugosité », va déterminer le passage d'une réflexion diffuse à une réflexion spéculaire. En effet, cette valeur détermine la largeur du cône par lequel sera reflétée la lumière arrivant sur l'objet. Plus cet angle est grand, plus la lumière sera diffuse et

floue. À un degré extrême de rugosité, nous obtenons une réflexion purement diffuse. Dans l'autre sens, plus l'angle est petit, plus la réflexion sera nette, jusqu'à obtenir un miroir pur.

Physiquement, certaines propriétés de la réflexion sont effectivement différentes qualitativement selon que la réflexion est spéculaire ou diffuse, notamment la polarisation de la lumière. Les photographes se servent d'ailleurs de filtres polarisants pour réduire la quantité de lumière spéculaire reflétée dans certaines situations, comme pour les scènes maritimes ou lors de la présence de vitres. Certains animaux, notamment des insectes, sont capables de percevoir la différence de polarité de la lumière et utilisent celle-ci pour reconnaître les surfaces d'eau. Dans notre cas précis, néanmoins, la polarisation de la lumière n'est pas importante et les deux phénomènes peuvent être considérés comme similaires.

Il existe néanmoins d'autres types de réflexions qui ne sont ni spéculaires ni diffuses, et toutes les combinaisons de ces types de réflexions peuvent se retrouver au sein d'une même surface.

Ces effets sont liés aux caractéristiques structurelles de la surface (formes de la surface au niveau microscopique, coefficient d'absorption…), ainsi qu'au nombre de couches qui se superposent sur une surface et influencent la lumière. Ainsi la peau est un des éléments les plus complexes à reproduire, car elle est faite de plusieurs couches successives chacune plus ou moins réflexive (chair, épiderme, couche de sueur…). La thèse de Charles De Rousiers[45] propose une très bonne étude des moyens existants pour prendre en compte l'état de surface des matières en vue d'une reproduction « réaliste ».

Nous avons donc besoin d'un système pouvant décrire n'importe quelle forme de réflexion lumineuse. Ce système est celui de la BRDF (Bidirectional Reflectance Distribution Function) ou « fonction bidirectionnelle de distribution de la réflexion » en français. Il s'agit d'une variable à cinq dimensions qui nous permet de connaître la quantité de lumière reflétée dans une direction donnée lorsque la lumière frappe la surface selon une autre direction donnée avec une certaine longueur d'onde. Nous pouvons en général limiter la gestion de la longueur d'onde de la lumière à une séparation en trois canaux de couleur (rouge, vert et bleu). La direction d'entrée de la lumière (ou irradiance) et la direction de sortie (ou radiance) sont chacune décrites par deux variables (coordonnée azimutale et zénithale), formant ainsi un système à quatre dimensions.

45 Charles De rousiers, *Rendu réaliste de matériaux complexes* (Grenoble, 2011), http://www.theses.fr/2011GRENM055.

Si la BRDF en un point donné d'une surface est connue, nous pouvons calculer la lumière réfléchie selon un angle de sortie en calculant l'intégrale des lumières incidentes couplées au calcul de la BRDF pour chacune d'elle.

2.2 La réflexion numérique

Le terme « réflexion » désigne en fait deux phénomènes différents selon le point de vue utilisé pour le décrire. D'un point de vue physique, c'est un « brusque changement de direction d'une onde à l'interface de deux milieux ». Ce point de vue considère chaque rayon lumineux et son parcourt individuellement. Du point de vue visuel, phénoménologique, c'est la copie inversée d'un décor reflété par une surface de type miroir.

Dans le domaine du graphisme assisté par ordinateur, nous sommes habitués à séparer ces deux aspects, car nous utilisons des techniques différentes pour les simuler. La réflexion, en tant que phénomène physique, est en général modélisée en calculant numériquement la relation entre la surface, la lumière et la caméra. Au contraire, la réflexion comme miroir nécessite de produire au préalable une image du reflet du miroir avant de l'utiliser en tant qu'image de réflexion.

2.2.a Les lumières ponctuelles

Il convient de rappeler qu'il est ici question de lumière, et de son transport jusqu'à l'œil ou à la caméra réelle ou virtuelle. Lorsque nous regardons un objet, ce que nous percevons n'est en fait que la réflexion de la lumière provenant d'une source lumineuse qui l'éclaire directement ou indirectement. L'éclairage direct est constitué de la lumière qui part de la source lumineuse et arrive directement sur l'objet. Cette lumière subit alors diverses transformations avant d'arriver jusqu'à notre œil. Tout d'abord, une partie est absorbée par la surface, puis réémise peu de temps après sous forme de lumière ou de chaleur. La lumière peut également être réémise de l'autre côté de la surface, ce qui, si la direction de la lumière n'est pas trop perturbée, sera qualifié de transparence et qui sinon entrera plus dans le domaine de la translucidité. La majeure partie de la lumière qui touche un objet est réémise depuis la surface de manière assez anarchique, c'est-à-dire sans que la direction de réémission dépende réellement de la direction d'absorption. Ce phénomène peut être expliqué par la présence des « microfacettes » dont nous parlions, qui provoquent des réflexions aléatoires. C'est ce que nous appelons la réflexion diffuse, car elle se diffuse dans toutes les directions et que la

position de l'œil n'influe pas sur la quantité de lumière émise dans sa direction. La lumière qui n'est pas suffisamment absorbée par l'objet pour être très perturbée est reflétée directement dans une direction précise. La position de la caméra ou de l'œil va alors fortement influer sur l'intensité lumineuse qui l'atteint, selon qu'elle est ou non dans l'axe de réflexion de la lumière par rapport à la normale de la surface. C'est ce que nous appelons la lumière spéculaire. La couleur de la lumière diffuse va être fortement filtrée par la couleur de l'objet, au contraire la lumière spéculaire ne sera pas filtrée de la même façon et aura donc souvent une autre couleur, ou une couleur plus neutre. Les métaux, en particulier, n'ont pas de réflexion diffuse, et toute la couleur perçue est le résultat de la réflexion spéculaire. Ils ont généralement une réflexion anisotropique, c'est-à-dire dont la puissance spéculaire n'est pas la même selon la direction d'entrée de la lumière par rapport à la surface. C'est le cas des métaux brossés, dont les microfacettes qui constituent l'état de surface sont alignées selon une direction. Dans cette direction de brossage, la réflexion sera fortement spéculaire, mais bien moins dans la direction perpendiculaire, ou les aspérités sont plus nombreuses.

La plupart des matériaux que nous trouvons dans la réalité ne sont ni des miroirs parfaits (donc totalement spéculaires), ni des surfaces parfaitement diffuses, mais un mélange des deux comportements en fonction des caractéristiques de la surface.

Nous avons parlé pour le moment d'une lumière ponctuelle qui éclaire directement l'objet, et dont nous pouvons spécifiquement localiser la position par rapport à l'objet. Dans ce cas précis, ce que nous pouvons appeler « réflexion » se modélise par le biais de la réflexion spéculaire.

2.2.b Les lumières indirectes

En dehors de la lumière directe, les objets sont également éclairés par une lumière indirecte constituée par la multitude des reflets de lumière qui se font sur les objets alentour (reflets diffus, spéculaires, transparences…) et qui atteignent ensuite la surface. Cette source lumineuse est plus délicate à traiter, car elle fait intervenir l'ensemble de la scène. Elle présente même une boucle, puisque chaque objet influence ces voisins qui eux même l'influencent en retour.

Malgré tout, ces sources de lumière indirectes ne se comportent pas différemment des sources ponctuelles même si le résultat visuel peut paraître très différent. Ainsi une partie de ces sources seront réfléchies en lumière diffuse, c'est ce qu'on appelle en infographie la

« radiosité », car chaque objet « irradie » sa couleur diffuse sur son environnement proche. La partie qui, au contraire, sera réfléchie de manière « spéculaire » directement vers la caméra est ce que nous appelons traditionnellement une réflexion.

Il est important de comprendre que ces deux réflexions spéculaires, celle de la lumière directe, et celle de la lumière indirecte, sont bien issues du même phénomène et répondent aux mêmes équations physiques, même si souvent nous utilisons des algorithmes différents, spécifiques à chacune.

2.3 Équations classiques de simulation de BRDF

La simulation informatique de l'aspect des matériaux utilise depuis plusieurs années des équations simplifiées afin de reproduire différents types de BRDF, c'est à dire essentiellement des approximations de la relation entre lumière entrante et sortante d'une surface, en fonction des directions d'entrée (vers la lumière) et de sortie (vers la caméra). Ces approximations sont souvent adaptées à un type précis de matériau, mais échouent à représenter une gamme étendue de matières.

2.3.a Équation de Lambert

La partie diffuse est classiquement modélisée par une équation de Lambert[46] , qui décrit la lumière émise par la surface comme dépendante de l'aire vue depuis la lumière. Cette aire est au maximum lorsque la surface est perpendiculaire à la direction de la lumière, et devient minimale dans la situation de parallélisme entre les deux vecteurs. La réflexion diffuse est alors égale au cosinus de l'angle entre les deux vecteurs, c'est-à-dire le produit scalaire des deux vecteurs normalisés.

$$d = \vec{N} . \vec{L}$$

d : quantité de réflexion diffuse
N: vecteur normal à la surface
L : vecteur direction vers la lumière

Cette équation est simple, intuitive et rapide à calculer. Elle ne dépend absolument pas de la position de la caméra et modélise une matière diffuse « idéale ». Elle présente des variations importantes par rapport à la plupart des matières diffuses réelles.

46 J. H. Lambert, « Photometria sive de mensure de gratibus luminis, colorum umbrae », Eberhard Klett (1760).

2.3.b Équation de Phong

La partie spéculaire de la réflexion est modélisée classiquement par l'équation de Phong[47], une équation empirique qui est très simple, et convaincante pour certaines matières. Par contre, cette équation n'est pas physiquement réaliste. L'équation cherche à quantifier l'adéquation entre la direction du reflet de la lumière en fonction de la normale, et la direction de la caméra. Le réglage de la largeur du cône de réflexion s'opère à l'aide d'un simple exposant.

$$s = max((\vec{R} . \vec{V}), 0)^n$$

s : quantité de réflexion spéculaire
R : vecteur reflété de la lumière selon la normale : $\vec{R} = 2\vec{N}(\vec{L} . \vec{N}) - \vec{L}$
V : vecteur vers la caméra
n : exposant définissant l'amplitude du cône de réflexion

L'utilisation de la fonction « max » permet de limiter le reflet spéculaire à la partie éclairée de la surface, l'utilisation d'un exposant pair pouvant créer un second reflet spéculaire à l'opposé du reflet attendu. Cette discontinuité dans l'équation de Phong est un défaut majeur, créant une discontinuité visuelle dans le reflet spéculaire lui-même.

2.3.c Equation de Blinn-Phong

L'équation de Blinn-Phong[48] améliore le modèle de Phong, notamment en créant un reflet qui s'étire le long d'une surface plane au lieu de former toujours un rond comme l'équation de Phong. Les modèles de Phong et de Blinn-Phong sont pratiquement équivalents sur des surfaces organiques et douces, sans surfaces planes. Le modèle de Blinn-Phong approche mieux la réalité, mais reste une approche empirique, sans être réaliste physiquement.

$$s = (\vec{N} . \vec{H})^n$$

s : quantité de réflexion spéculaire
H : demi-vecteur (normalisé) : $\vec{H} = |\vec{V} - \vec{L}|$
N ; vecteur normal à la surface
n : exposant définissant l'amplitude du cône de réflexion

47 B. Tuong-Phong, *Illumination for Computer-Generated Images*, 1973.

48 J.F. Blinn, « Models of light reflection for computer synthesized pictures », in *ACM SIGGRAPH Computer Graphics*, vol. 11, 1977, 192–198.

2.4 Équation générale des BRDF

La BRDF est une relation à 4 dimensions au moins, nous l'avons vu. Le premier objectif d'une équation de simulation d'une BRDF est de pouvoir reproduire cette relation avec le minimum de calculs et de données stockées en mémoire. Cependant, la relation simulée par une BRDF n'est fixe que pour un certain type de matière. Chaque matière aura ainsi sa propre BRDF qui lie les rayons lumineux arrivant sur la surface à ceux en ressortant. Le véritable challenge d'une équation de simulation de BRDF n'est pas uniquement la reproduction de la BRDF d'une matière précise, mais plutôt d'un ensemble plus ou moins étendu de matériaux présentant des similarités. L'équation de BRDF idéale pourrait ainsi reproduire n'importe quel matériau.

Le nombre de variables et donc de dimensions dans lesquelles une équation de BRDF travaille est en fait bien plus grand que les quatre formées par les angles d'incidence et de réflexion. Les autres dimensions sont les variables qui définissent un certain type de matière, par exemple la rugosité. Une bonne BRDF doit être choisie pour la diversité de matières qu'elle est capable de simuler, mais également pour la facilité de réglage de ses paramètres. Certaines équations (utilisation des polynômes de Zernike[49] par exemple) nécessitent des dizaines de valeurs pour définir un matériau précis, ce qui rend pratiquement impossible tout réglage manuel et oblige à mesurer directement la réflexion des matériaux réels dans toutes les directions, puis d'en extraire les paramètres de l'équation.

Les équations des BRDF peuvent en général être décomposées en plusieurs parties. Selon le modèle des microfacettes, voici l'équation générale de calcul de la relation entre direction d'entrée et de sortie :

$$f(L,V)=\frac{F\times G\times D}{4\times N\cdot L\times N\cdot V}$$

F : réflexion de Fresnel
G : facteur géométrique
D : distribution des normales

L'intérêt de cette équation est de séparer le calcul en trois sections indépendantes, pour lesquelles différentes approximations sont disponibles dans la littérature et peuvent être

49 S. Enrique, « Real-Time Rendering Using CUReT BRDF Materials with Zernike Polynomials », *Topics on Computational Vision and Graphics* (2004).

mélangées entre elles. La fonction F désigne la réflexion de Fresnel et modélise l'augmentation de la réflexion aux angles rasants de la surface, qui produit visuellement un halo autour de l'objet. Nous verrons plus tard l'approximation de Schlick, qui est simple et précise. Le facteur géométrique G désigne la probabilité qu'une microfacette qui suit une direction donnée soit ombrée par les autres facettes, ou invisible depuis la caméra. La distribution des normales D indique la proportion de microfacettes qui ont une direction donnée. Cette distribution permet de définir la taille et la forme du reflet, qu'il soit spéculaire ou diffus. Toutes les fonctions de distributions prennent en compte un paramètre de « rugosité » sous une forme ou une autre, qui contrôle la balance de la réflexion entre spécularité et diffusion.

2.5 Exemples d'équations avancées de BRDF

2.5.a Oren-Nayar :

Cette équation[50] s'occupe de reproduire des matériaux mats, sans spéculaire en tant que telle, mais qui diffusent tout de même plus la lumière lorsque la caméra est alignée avec la direction d'éclairage. Les matières les mieux simulées par cette équation sont le béton, les matières poussiéreuses, l'argile.

Figure III.5: Illustration de la différence entre le modèle de Lambert (à gauche) et de Oren-Nayar (à droite).

50 M. Oren et S.K. Nayar, « Generalization of Lambert's reflectance model », in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, 1994, 239–246.

2.5.b LaFortune :

L'équation de LaFortune[51] est une équation spéculaire très adaptable, qui consiste simplement à superposer plusieurs lobes de réflexion dont nous pouvons fixer la taille sur les 3 axes, afin d'approximer la BRDF désirée. Un modèle de LaFortune à un seul lobe équivaut à un Phong. Trois lobes sont considérés comme suffisants pour la plupart des matériaux. Le modèle de LaFortune peut également servir à calculer un matériau anisotropique. L'inconvénient principal de cette équation est de ne pas avoir d'analogies entre les paramètres de l'équation et les caractéristiques physiques ou visuelles du matériau. Il est donc nécessaire de calculer les valeurs à partir de relevés optiques de matériaux réels ou bien à partir d'une simulation virtuelle qui présente, elle, des paramètres compréhensibles par un humain.

2.5.c Ashikmin-Shirley :

L'équation d'Ashikmin-Shirley est une équation spéculaire complexe et lourde, avec des paramètres très peu intuitifs, mais qui est prévue pour les réflexions anisotropiques. Une partie diffuse est également présentée et possède à peu près les mêmes caractéristiques que l'équation d'oren-nayar.

2.5.d Koenderink :

L'équation de Koendering est issue d'un champ plus large des mathématiques. L'idée est un peu similaire à celle des harmoniques sphériques, c'est-à-dire d'approcher une fonction compliquée par l'addition de fonctions de bases fixes multipliées par un facteur, à la manière d'une décomposition de Fourier. Seuls les facteurs auront ainsi besoin d'être sauvegardés, les fonctions demeurant les mêmes. Ces fonctions ne sont pas ici sur une seule dimension, mais sur deux, et décrivent une demi-sphère. Pour du temps réel, nous nous limitons au second ordre des équations de Koenderink, ce qui donne tout de même 5 fonctions à additionner et donc 5 paramètres à stocker, par composante (rouge, vert et bleu). Pour une reproduction fidèle, l'utilisation des équations jusqu'à l'ordre 8 nécessite 55 paramètres.

51 E. P. F. Lafortune et al., « Non-linear approximation of reflectance functions », in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 1997, 117–126.

2.5.e Cook-Torrance

Le modèle de Cook-Torrance[52] est un modèle qui sera peut-être adopté comme standard dans quelques années dans les applications interactives. Il introduit un facteur géométrique, utilise la distribution des normales de Beckmann[53], et une équation de Fresnel. Ce modèle permet de représenter une grande variété de matières, depuis les plastiques jusqu'aux métaux.

Figure III.6: Modèle de Cook-Torrance

2.6 Équation de Blinn-Phong normalisée

Voici les équations utilisées dans le cadre du calcul de l'impact lumineux d'une lumière ponctuelle, c'est-à-dire dont la lumière provient d'un point unique. C'est ce modèle que nous avons utilisé chez DONTNOD.

Luminosité diffuse :

LumDif = saturate(dot(N, L))
N : Normale de la surface (espace tangent)
L : Direction de la surface vers la lumière (espace tangent)

Luminosité spéculaire :

*LumSpec = ((s + 2) / 8) * saturate(dot(N,H)) ^ s * saturate(dot(N, L))*
*s : puissance spéculaire lié à la rugosité (s = 2 ^ (rugosité * 10 + 1))*
N : Normale de la surface (espace tangent)
H : Demi-vecteur : bissectrice entre les vecteurs lumière et caméra (tangent)
L : Direction de la surface vers la lumière (espace tangent)

52 R. L. Cook et K. E. Torrance, « A reflectance model for computer graphics », *ACM Transactions on Graphics* (1982): 1(1):7-24.

53 P. Beckmann et A. Spizzichino, « The Scattering of Electromagnetic Waves from Rough Surfaces », *The Macmillan Company, New York* (1963).

Approximation de Schlick de l'équation de Fresnel :

*Fresnel = CSpec + (1 - CSpec) * (1.0f - saturate(dot(N, C)) ^ 5*
Cspec : couleur spéculaire de la surface
N : Normale de la surface (espace tangent)
C : Direction de la surface vers la caméra (espace tangent)

Obtention de la couleur finale :

*finale = DiffuseColor * LumDif + LumSpec * Fresnel*

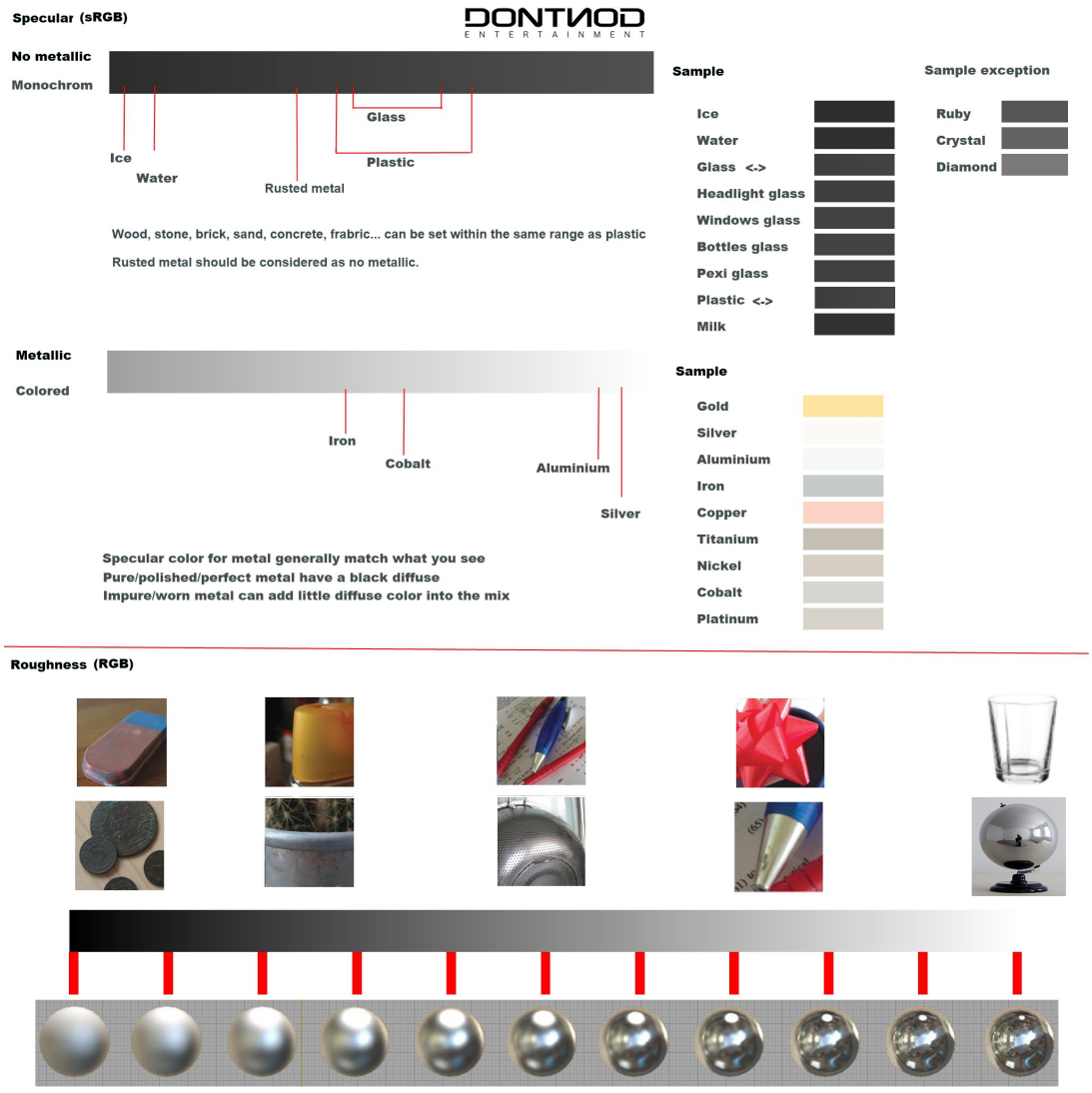


Figure III.7: Tableau utilisé à DONTNOD comme référence des valeurs de couleur diffuse, spéculaire et de rugosité de différents matériaux.
Les artistes peuvent utiliser ce tableau comme base de travail afin de partir sur des bases « physiquement réalistes ».

3 Les paramètres de la surface virtuelle

3.1 Encodage couleur diffuse, émissive, rugosité et couleur spéculaire

Dans le cadre d'un jeu vidéo, nous souhaitons utiliser un modèle de description de surfaces et de matières qui soit le même pour tous les objets. Nous avons déjà expliqué l'intérêt en terme de crédibilité et de cohérence. Il est possible de spécifier les paramètres d'un tel modèle en fonction de l'objet, mais nous souhaitons pouvoir changer de matière au cœur d'un même objet. Cela nous permet par exemple de représenter les différents vêtements (tissus ou métaux) sans devoir diviser un personnage en plusieurs objets séparés par matière. Ainsi, tous les paramètres utiles seront fournis au modèle à partir de textures. Les effets comme l'usure nécessitent également de modifier les caractéristiques d'un matériau de manière pratiquement continue le long de la surface. Seuls quelques types de matières conserveront un modèle spécifique, c'est le cas notamment de la peau et des surfaces d'eau, qui sont trop complexes et différentes du modèle général pour pouvoir y être intégrées facilement, et efficacement en terme de performance.

Les différents paramètres dont nous avons besoin pour décrire notre surface sont alors : une couleur diffuse, une couleur spéculaire, une valeur de rugosité, une couleur d'émissive, une texture de normale, et éventuellement une texture d'ambiance spéculaire. Il a été démontré que l'œil humain était très sensible à la variation de rugosité (qui est liée à la « reflectance ») dans la reconnaissance des matières[54]. Les auteurs de l'étude concernée remarquent que l'estimation visuelle du degré de réflexion d'un matériau est remarquablement stable, quel que soit l'éclairage, tant que celui-ci est calculé en se basant sur la réalité physique.

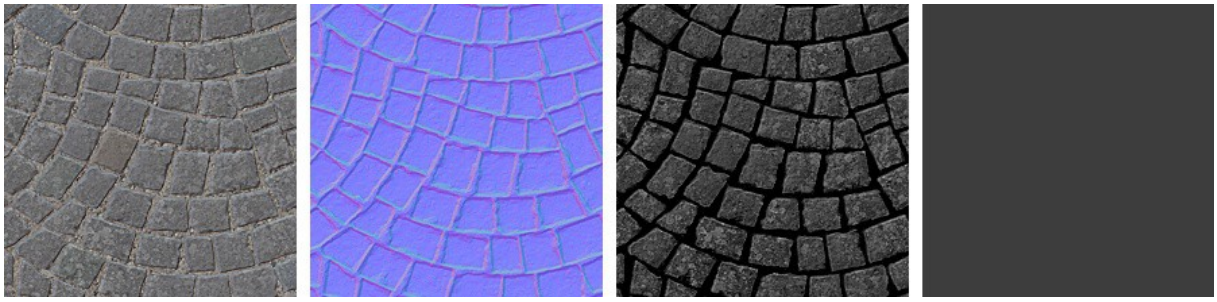


Figure III.8: Exemple de textures utilisées pour la description de la surface.
De gauche à droite, la texture diffuse, la texture de normale, la texture de rugosité et la texture de couleur spéculaire.

54 R. W. Fleming, R. O. Dror, et E. H. Adelson, « Real-world illumination and the perception of surface reflectance properties », *Journal of Vision* 3, n° 5 (2003).

Au niveau des résolutions des différentes textures, il est intéressant de remarquer que les différentes informations nécessitent une fréquence d'échantillonnage différente. Ainsi, la couleur spéculaire participe assez peu au rendu final et présentera peu de variations en dehors des frontières entre deux matériaux complètement différents. Une faible résolution suffira donc en général, et si la matière est la même pour toute la surface, une simple texture d'un pixel de côté sera suffisante. La couleur diffuse est importante, car elle définit la base de la couleur, mais elle accepte une résolution moyenne assez bien. En effet, elle servira uniquement de facteur multiplicateur sans que les calculs du modèle d'éclairage n'accentuent les gradients de couleur de manière non linéaire, ce qui pourrait augmenter la fréquence d'échantillonnage nécessaire pour une bonne qualité. Les informations de normale et de rugosité sont celles qui bénéficient le plus d'une haute fréquence d'échantillonnage, et donc utilisent une grande résolution. La normale et la rugosité servent directement dans les calculs de manière non linéaire, par exemple lors de l'utilisation d'une exponentielle. Ce genre de calculs accentue de faibles différences dans les données de départ, obligeant à échantillonner celles-ci à une plus grande fréquence pour conserver une qualité acceptable.

3.2 Adéquation entre spéculaire et texture de normale

La texture de normale sert à modéliser des variations de la surface à une échelle moyenne, qui serait difficile à modéliser intégralement en géométrie. La texture de rugosité décrit également l'aspect de la surface, mais à une échelle microscopique, trop faible pour être représentée par la texture de normale. Le niveau que nous nommons ici « microscopique » est en fait le niveau du subpixel : les variations qui interviennent à une échelle inférieure à un pixel de l'image finale. Ce niveau est donc dynamique puisque la taille d'une texture à l'écran va dépendre du rapport entre la distance de la caméra et la taille de la surface texturée. Lorsque la distance est faible, la texture de normale est adéquate pour représenter les variations de la surface, mais lorsque la caméra s'éloigne, toutes ces variations passent à des échelles inférieures au pixel, et seraient donc mieux représentées par la texture de rugosité. Il y a donc une certaine liaison entre les deux textures en fonction de la distance à la caméra. Vu de très prés, les détails de la surface apparaissent, alors que vu de loin, ils deviennent invisibles, mais participent alors à la rugosité de la matière. Des approches récentes (Lean Mapping[55], Clean

55 M. Olano et D. Baker, « LEAN mapping », in *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, 2010, 181–188.

Mapping[56] et autres développements[57]) visent à produire ces deux textures au cours d'un même processus, afin de pouvoir créer les « mip-maps » de la texture de rugosité en fonction de la texture de normale. Il s'agit concrètement de préfiltrer[58] la texture de normale et de la convertir en texture de rugosité. En effet, l'utilisation de la technique du mip-mapping permet de créer des versions différentes de la texture en fonction de sa taille finale en pixels sur l'écran. Les niveaux inférieurs des mip-maps de la texture de rugosité vont alors gagner en rugosité aux endroits où la texture de normale est très bruitée, afin de transférer la variation des normales en rugosité lorsque l'échelle est trop petite.

56 Dan Baker, « Spectacular Specular - LEAN and CLEAN specular highlights », *Firaxis Games* (GDC 2011).

57 Stephen Hill, « Specular Showdown in the Wild West », 2011, http://blog.selfshadow.com/2011/07/22/specular-showdown/.

58 E. Bruneton et F. Neyret, « A Survey of Nonlinear Prefiltering Methods for Efficient and Accurate Surface Shading », *IEEE Transactions on Visualization and Computer Graphics* 18, nº 2 (février 2012): 242-260.

4 La réflexion

4.1 Techniques dynamiques / techniques statiques

Les techniques de visualisation 3D peuvent se classer en deux catégories différentes. La partie statique et la partie dynamique. En temps réel, il est souvent utile de combiner ces deux éléments afin de bénéficier de leurs différents avantages.

Imaginons par exemple que nous souhaitons afficher l'image d'une maison sur un écran. Utiliser une technique dynamique signifie alors décrire la maison, ses proportions en trois dimensions, ses couleurs et son éclairage, puis définir un point de vue qui servira à la projection en perspective. Enfin, l'algorithme va calculer la couleur de chacun des pixels de l'image en fonction de ces éléments. Le calcul sera refait intégralement à chaque affichage de l'image, soit en général 30 fois par seconde. Utiliser une technique statique signifie simplement stocker l'image calculée par la méthode dynamique puis l'utiliser lorsque l'on souhaite afficher l'image de la maison. Immédiatement, nous pouvons constater les avantages et les contraintes des deux techniques. La version statique est bien évidemment beaucoup plus rapide, il suffit de parcourir les pixels de l'image et de copier une valeur sur l'écran. Par contre, la méthode dynamique permet de modifier n'importe quel élément de l'image ou l'angle de vue, et l'image sera calculée en conséquence, et sans surcoûts puisque de toute manière l'image était déjà recalculée en continu.

Cette division caricaturale entre statique et dynamique n'est bien évidemment pas aussi franche en général. Ainsi, pour éviter d'avoir à calculer l'influence lumineuse à chaque image, nous pouvons la stocker dans des textures (une pour chaque mur par exemple) afin de pouvoir la retrouver aux images suivantes. Ainsi, nous pouvons faire bouger la caméra de manière dynamique tout en ayant un éclairage statique. Les calculs sont alors bien plus rapides. Par contre, impossible de déplacer une lumière sans devoir recalculer les textures stockant l'influence lumineuse. La quantité de mémoire nécessaire au stockage de l'éclairage peut devenir importante et poser des problèmes de performance liés à l’accès à ces données.

Un algorithme efficace en temps réel est généralement choisi pour son bon compromis entre stockage statique de données précalculées et calculs dynamiques afin de s'adapter aux changements de situation.

Dans ce contexte, la réflexion pose des contraintes importantes. En effet, la réflexion est un phénomène assez global, dont l'aspect change en fonction de la position de la caméra et de l'emplacement de tous les objets de la scène virtuelle. En comparaison, un éclairage diffus, même calculé à base de radiosité, c'est-à-dire en tenant compte des rebonds de la lumière sur les murs, ne change pas lors du déplacement de la caméra. La réflexion dépend de l'environnement proche mais aussi lointain propre à chaque objet. Il est donc difficile de définir les variables qu'il est possible de rendre statique, de précalculer ou bien carrément de supprimer. Pour précalculer un point donné de l'espace, par exemple, il est possible de capturer une image sphérique centrée sur le point et permettant de connaître les influences lumineuses en provenance de n'importe quelle direction. Le calcul de la réflexion vue depuis la caméra consiste alors à calculer une moyenne pondérée des pixels de l'image sphérique en fonction de l'équation de BRDF choisie. Lorsque la position du point change, ou bien lorsque n'importe quel objet environnant bouge, la réflexion n'est plus valable et doit être recalculée, y compris l'image sphérique.

4.2 Le problème du stockage et de la structure

Sans prendre en compte le mouvement des objets, qui invalide les captures de l'environnement, il reste nécessaire de stoker une image sphérique pour un grand nombre de points de la scène afin de pouvoir correctement reproduire les réflexions. Bien évidemment, un grand nombre des informations contenues dans ces images sphériques sont redondantes avec celles des sphères voisines. Il est difficile de créer une structure qui évite ces redondances tout en permettant à un algorithme efficace d'aller chercher et regrouper les informations nécessaires au calcul de la réflexion d'un point en particulier. La structure en mémoire est également importante au niveau des performances. Il faut ainsi s'assurer que les données auxquelles nous accéderons simultanément ou successivement sont proches physiquement dans la mémoire. Les calculs pour obtenir les emplacements à accéder doivent également être les plus simples possible afin d'être effectués en temps réel.

4.3 Les techniques classiques de réflexion

La réflexion étant un phénomène complexe dont le résultat visuel est modifié par de nombreux paramètres, il est intéressant de décomposer les différentes catégories de réflexions que nous pouvons rencontrer afin de définir un algorithme adapté à une catégorie spécifique.

Nous allons voir ici les différentes techniques classiques, chacune adaptée à une situation bien particulière.

4.3.a Réflexions semi-planes dynamiques de type miroir

Technique classique

La première simplification proposée concerne la forme du réflecteur, c'est-à-dire de l'objet qui va refléter l'environnement. Une forme arbitraire nécessiterait un algorithme très complexe et gourmand. Au contraire, si la forme est un plan, les calculs de projection de l'environnement sur la réflexion se simplifient drastiquement, et finissent même par ressembler aux calculs habituels de perspective avec lesquels le matériel est habitué à travailler.

Concrètement, la manière classique d'afficher une réflexion plane est de calculer une image de la scène vue depuis une caméra inversée par rapport à la normale du plan. Les caméras sont ainsi positionnées sous le sol, et pointent généralement vers le ciel. Une fois cette image calculée et stockée dans une texture, il suffit de l'utiliser lors du calcul de la couleur du sol dans l'image normale. La coordonnée de texture à utiliser est très facile à calculer, puisque la caméra de capture est une simple symétrie de la caméra de base. Nous pouvons simplement inverser la coordonnée Y de la position 2D sur l'écran.

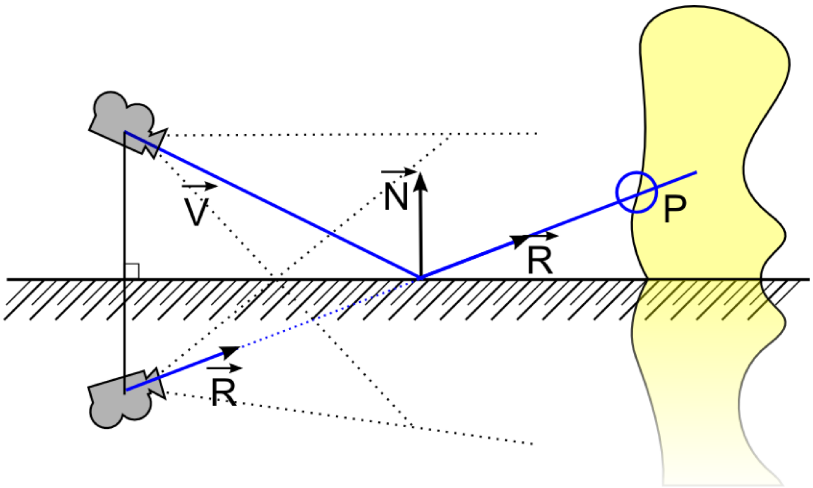


Figure III.9: Réflexion en miroir : une image est calculée grâce à une caméra reflétée par rapport au plan de réflexion.

La seule précaution à prendre concerne les parties du décor qui traverserait ou serait en dessous du plan de réflexion. Dans ces cas, l'image calculée par la caméra reflétée serait encombrée par ces objets qui ne devraient pas se refléter puisqu'ils sont sous le plan de réflexion, mais qui sont tout de même visibles vus depuis la caméra de réflexion. La solution

retenue généralement est d'utiliser un plan de « clipping » qui va éliminer les objets ou les parties d'objets qui sont d'un côté de ce plan. La caméra virtuelle des moteurs temps réel possède en général plusieurs de ces plans afin de n'afficher que les parties de la scène qui entrent dans le champ de la caméra. L'un de ces plans, le « near », est habituellement parallèle à l'image-plan de la caméra et sert à exclure les objets trop proches. Ce plan peut être modifié afin de le faire coïncider avec le plan de réflexion et ainsi éliminer les objets qui sont sous ce plan.

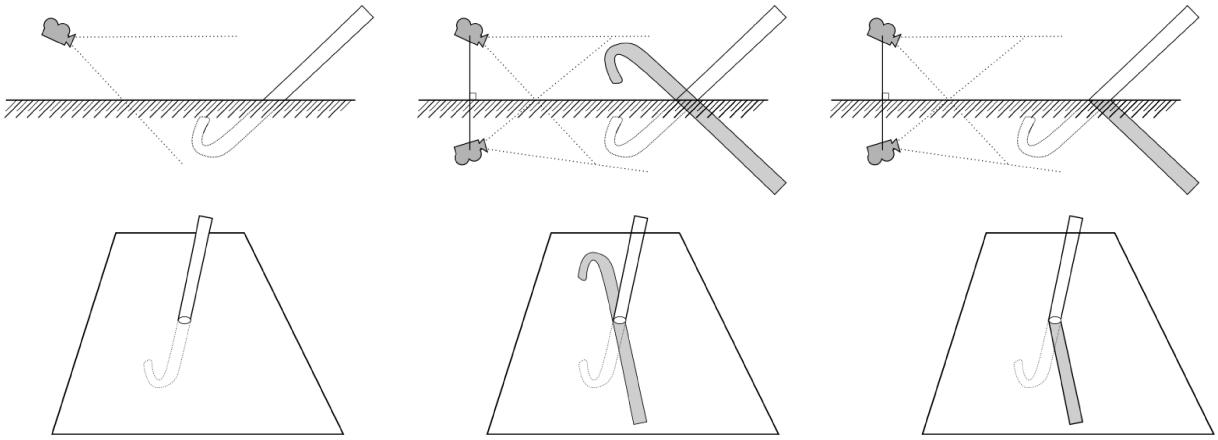


Figure III.10: L'utilité du plan de clipping dans la réflexion.
À gauche, l'objet à refléter traverse le plan de réflexion. Au centre, on effectue un rendu vu depuis une caméra « miroir ». La partie de l'objet sous la surface de réflexion ne devrait pas apparaître dans la réalité. Avec un plan de clipping placé au niveau du plan de réflexion, à droite, cette partie sous la surface est éliminée et la réflexion est correcte.

Cette technique permet de calculer une réflexion parfaitement spéculaire de type miroir. C'est une technique hautement dynamique puisque tout changement d'angle de caméra ou d'objet dans la scène nécessite de recalculer l'image de réflexion. Le coût est assez élevé puisqu'il est nécessaire d'afficher une seconde fois la scène pour calculer l'image de réflexion ce qui, sans optimisations particulières, reviens à doubler le coût de l'affichage de la scène. De plus, si nous souhaitons obtenir plusieurs plans de réflexion, à différentes hauteurs par exemple, il est nécessaire de recalculer la scène pour chacun deux.

Extension aux semi-plans

Nous avons eu besoin, lors du travail sur l'aspect visuel de la simulation de fluide « Fluidz », d'obtenir des réflexions sur l'eau. La simulation se passait à une échelle assez grande, telle que celles des canaux, à laquelle la surface de l'eau peut être assimilée grossièrement à un plan avec de faibles ondulations. Afin d'obtenir des réflexions dynamiques, nous avons modifié l'algorithme habituel pour l'étendre à de faibles variations dans la hauteur du plan de

réflexion. Ainsi, si la surface est effectivement un plan, notre algorithme assure que nous obtiendrons le même résultat qu'avec l'algorithme classique. Par contre, plus la surface s'éloigne du plan, et plus la réflexion se déforme jusqu'à finalement perdre toute crédibilité au-delà d'un certain seuil. L'intérêt de cette technique est de conserver une réflexion correcte aux points de contact entre la réflexion et le décor. Ces contacts sont importants, car ils représentent des indices précis de la hauteur de l'eau. De plus, la correspondance entre un objet et sa réflexion se remarque particulièrement aux endroits de contacts.

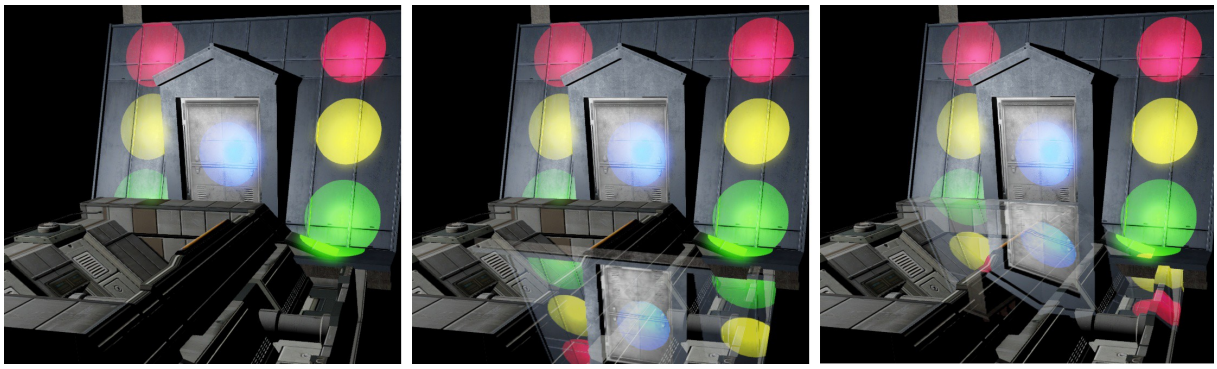


Figure III.11: Importance des points de contact dans la crédibilité de la réflexion.
À gauche, la scène sans réflexion. Au centre, une réflexion planaire est appliquée sur un objet qui n'est pas un simple plan, la réflexion semble « passer à travers » l'objet. À droite, notre technique replace les points de contact entre l'objet et son reflet.

Le principe en jeu ici est tout d'abord de calculer une réflexion plus large que celle de la caméra originelle. Nous capturons l'environnement vu depuis la position reflétée de la caméra, mais avec une ouverture bien plus importante allant pratiquement jusqu'à capturer un hémisphère de l'environnement. Afin de mieux utiliser la résolution de la texture de réflexion, nous réalignons également la caméra en la dirigeant vers le décor à calculer. En effet, dans la majorité des cas, nous avons une caméra orientée parallèlement au plan de l'eau, ce qui induit que la caméra reflétée comporte une moitié vide, car coupée par le plan de réflexion. Cette

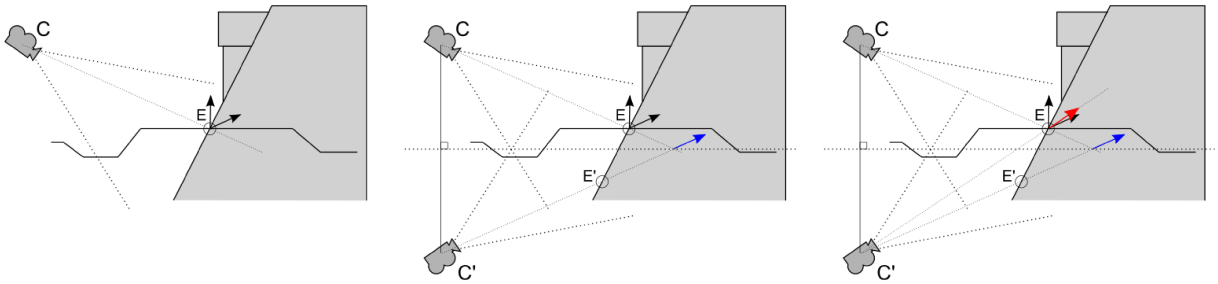


Figure III.12: Gestion d'une réflexion semi-plane.
À gauche, la scène normale. Nous souhaitons calculer la réflexion au point E. L'utilisation d'une réflexion plane pure (au centre) décale les points de contact et nous obtenons une réflexion à partir du point E'. Notre technique (à droite) utilise une réflexion à partir du point E, bien que la direction du reflet soit décalée.

partie de la texture de réflexion serait alors noire. Rediriger l'orientation de la caméra reflétée permet donc d'exploiter cet espace vide au mieux possible, en recadrant sur la portion de la scène au-dessus du sol.

Une fois l'image de réflexion calculée, il s'agit maintenant de la projeter sur la surface de réflexion. Le calcul dans l'algorithme classique est trivial, car l'image finale et l'image de réflexion sont parfaitement alignées, car calculées avec des caméras parfaitement reflétées. Dans notre cas, la situation est plus complexe. Nous utilisons la position dans l'espace de la caméra normale, et nous la projetons par la matrice de la caméra reflétée. Cela nous permet de trouver la position 2D sur l'écran de la caméra reflété à partir de la position 2D sur la caméra normale. Nous utilisons ainsi le vecteur depuis la caméra reflétée vers le point comme direction de réflexion du vecteur depuis la caméra normale vers le point. Cette hypothèse est exacte si le point de la surface appartient au plan de réflexion de la caméra. Ce calcul nous permet effectivement de conserver les surfaces de contacts, mais déforme néanmoins beaucoup le reste de l'image dès que les vagues du plan sont trop grandes. Le plan de « clipping » peut être utilisé en le plaçant au niveau le plus bas que peuvent atteindre les vagues. Les objets isolés, et notamment les objets flottants peuvent utiliser un plan de « clipping » manuel par objet, en communiquant au pixel shader la hauteur de la surface de réflexion à la position de l'objet. Le pixel shader éliminera les pixels inférieurs à cette hauteur, grâce à l'instruction « clip ».

4.3.b Réflexions ambiantes spéculaire et diffuse

L'éclairage d'une surface se compose de deux parties : la première est l'influence directe d'une lumière précise sur la surface, en considérant en général la lumière comme ponctuelle, et la seconde est l'influence de l'environnement lumineux, c'est-à-dire les autres objets, qui reflètent la lumière vers la surface. Cette dernière se nomme éclairage indirect, ou ambiant. Ces deux parties ne sont bien sûr pas séparées dans la réalité, les lumières ponctuelles n'existant pas réellement. Cette séparation est cependant très pratique pour simplifier les calculs et traiter de façon différente les effets qui présentent des caractéristiques visuelles distinctes. Nous pouvons calculer une bonne approximation de la lumière directe de lumières ponctuelles en calculant leurs influences séparément, à partir de leur position, couleur, intensité, puis en les additionnant toutes. L'éclairage indirect, par contre, ne peut se décomposer aisément en lumières ponctuelles de par le nombre de lumières qui serait nécessaire, mais aussi par la difficulté à trouver automatiquement les emplacements et valeurs

de celles-ci. De même, les modèles plus évolués de lumières sont difficiles à simuler en n'utilisant que des lumières ponctuelles. C'est le cas des « area lights », ces lumières n'étant pas ponctuelles, mais au contraire constituées d'une surface éclairante avec une aire non nulle.

Ce que nous dénommons « éclairage ambiant » regroupe ainsi l'éclairage indirect et les lumières trop complexes, mais peut également servir à simplifier l'influence de lumières ponctuelles situées suffisamment loin pour être intégrées à l'éclairage environnant.

À partir de l'éclairage ambiant, nous cherchons à extraire deux informations différentes : l'ambiance diffuse et l'ambiance spéculaire. Nous avons vu que ces deux informations font en réalité partie d'un tout continu et qu'il n'y a pas de barrières entre les deux types de lumières. Néanmoins, il est utile de les différencier, car elles n'ont pas les mêmes caractéristiques statistiques. L'éclairage ambiant diffus est à faible fréquence, il change en général très doucement spatialement, excepté aux zones d'occlusions, c'est-à-dire les murs par exemple. Il est tout de même intéressant de conserver une notion de direction dans l'éclairage ambiant, afin de pouvoir utiliser la texture de normale pour faire varier la couleur de l'éclairage.

Figure III.13: Surface éclairée par deux lumières, une bleue à gauche et une rouge à droite. Une surface plane ne présentera pas beaucoup de variations lumineuses (a.). Si nous ne disposons pas de la direction de la lumière, la normale ne servira qu'à moduler le mélange des deux lumières (b.). En utilisant la direction de la lumière, les couleurs des deux lumières se séparent selon la direction de la normale (c.).

Au contraire, l'éclairage ambiant spéculaire est à plus forte fréquence, non seulement au niveau de la quantité spatiale d'échantillons nécessaires, mais également au niveau de la fréquence d'échantillonnage des données de chaque échantillon. Nous avons besoin d'un grand nombre de sphères d'ambiance, et chacune doit être très précise.

L'éclairage ambiant est pour le moment encore très souvent précalculé, la caméra pouvant ensuite se déplacer librement sans nouveaux calculs. L'éclairage spéculaire est souvent stocké sous la forme de cube-maps.

4.3.c Les méthodes de stockage de l'ambiance lumineuse

Le stockage de l'ambiance lumineuse comporte de nombreuses techniques, chacune avec ses avantages et ses qualités, qui peuvent évoluer en fonction notamment des caractéristiques fréquentielles des données à stocker.

Afin de représenter l'éclairage ambiant d'une scène, nous allons la diviser selon une grille en stockant, pour chaque cellule, une représentation de l'ambiance lumineuse de la scène. Ensuite, les objets devront simplement utiliser la cellule la plus proche pour connaître leur éclairage ambiant, ou bien effectuer une interpolation entre les cellules voisines afin d'obtenir des transitions douces d'une cellule à la suivante.

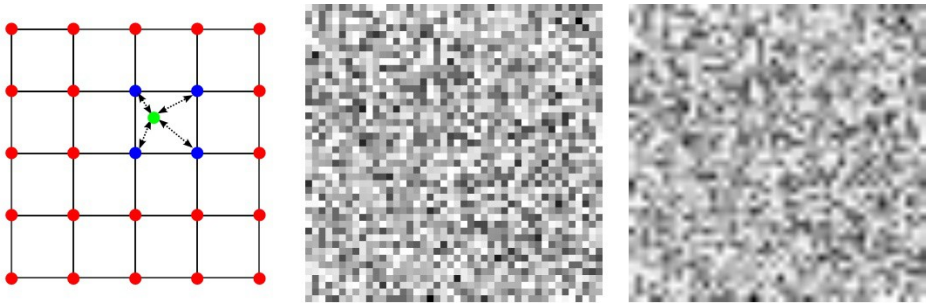


Figure III.14: Interpolation bilinéaire entre les échantillons d'une grille pour obtenir des transitions douces.
À gauche, chaque point rouge est un échantillon. Le calcul de la couleur au point vert se fait par une moyenne pondérée des plus proches voisins, les échantillons en bleu. Au centre, des valeurs aléatoires affichées sans interpolation et à droite avec une interpolation bilinéaire.

Le stockage d'un échantillon unique est la plus simple des solutions. Chaque point de la grille contient une couleur, qui est la moyenne des rayons lumineux qui passent par la cellule de la grille. Cette solution a pour avantage de prendre très peu de place et d'offrir une interpolation aisée entre les cellules en utilisant par exemple une interpolation trilinéaire en utilisant les huit voisins de la grille en 3D. La grille peut être en 2D dans certains cas spécifiques, par exemple en utilisant un seul niveau d'échantillons sur la hauteur, placé à la hauteur moyenne des personnages. L'ambiance lumineuse capturée est ainsi propice à l'éclairage des personnages.

Les solutions plus complexes commencent à ajouter des données directionnelles concernant la provenance de la lumière en un point donné. Ainsi, le développeur « Valve » utilise, dans son moteur « Source », six couleurs pour représenter la lumière ambiante diffuse provenant de chacune des six directions, regroupées sous la forme d'une cube-map de taille 1x1x1. Ces valeurs sont utilisées pour tous les objets dynamiques. Pour les objets statiques, l'ambiance

n'est pas stockée dans une grille, mais dans des textures, chaque échantillon étant associé à un point d'une surface. Grâce à cela, il n'est pas nécessaire de stocker une ambiance sphérique, mais uniquement un hémisphère, et trois valeurs représentant l'ambiance lumineuse dans trois directions autour de la normale de la surface suffisent à l'approximer grossièrement, et à conserver une dépendance entre la normale et la couleur.

La prochaine possibilité est de stocker l'ambiance lumineuse sous la forme d'harmoniques sphériques (ou Spherical Harmonics en anglais). C'est une représentation spectrale de données regroupées sur une sphère. Cette représentation est très pratique, car elle permet de stocker uniquement les basses fréquences de la donnée, qui sont les plus importantes dans l'éclairage ambiant diffus. La balance entre qualité et efficacité peut être réglée facilement en utilisant plus ou moins de bandes des harmoniques sphériques, ce qui donne plus ou moins de valeurs à stocker. Les choix les plus courants sont d'utiliser le niveau 2 qui doit stocker 4 valeurs et le niveau 3 qui doit stocker 9 valeurs. Malheureusement, le nombre de valeurs à stocker devient très vite énorme et l'algorithme n'est donc pas très approprié pour l'éclairage ambiant spéculaire, qui nécessite un échantillonnage à haute fréquence.

La dernière solution proposée ici est d'utiliser directement les échantillons en les stockant dans des cube-maps. Il s'agit d'une technique d'éclairage basée sur des images (Image Based Lighting). L'avantage des cube-maps est de présenter l'ensemble les données de la sphère sous une forme gérée nativement par le matériel. Le nombre de cube-maps qu'il est possible de gérer en temps réel est assez faible, il faut donc accepter une plus faible résolution spatiale au profit d'une meilleure précision directionnelle pour chaque échantillon.

L'interpolation entre plusieurs cube-maps est par contre délicate. En effet, avec un échantillonnage directionnel plus précis, les soucis de parallaxes sous bien plus visibles, et deux cube-maps proches auront des pixels de couleurs très différentes. Prendre simplement une moyenne pondérée des plus proches cube-maps ne donne ainsi pas un bon résultat. La gestion de la parallaxe est un phénomène important dans la réflexion, et nous aurons l'occasion d'y revenir.

4.4 Utilisation des « Cube-maps »

Nous souhaitons stocker l'éclairage qui arrive à un point donné de l'espace depuis toutes les directions. C'est la radiance, qui peut être représentée par une sphère de point lumineux. Cependant, la surface d'une sphère ne pouvant pas aisément être représentée en mémoire, le

choix d'une « cube-map » est plus adéquat. Une cube-map est constituée de six plans formant un cube. Chaque plan va stocker dans une texture la lumière arrivant depuis les directions de son sixième de sphère. Le stockage dans une cube-map considère l'environnement lumineux comme situé à une distance infinie du centre de la cube-map. La parallaxe n'entre ainsi pas en compte. La carte graphique pourra accéder à ses six plans à travers un appel unique.

4.4.a Cube-maps avec niveaux de rugosité :

Dans le cas d'un matériau purement spéculaire, connaître la réflexion de la lumière demandera de faire appel à un seul des échantillons stockés dans la cube-map, en suivant la direction du reflet de la caméra par rapport à la surface pour trouver la collision avec l'une des faces du cube puis les coordonnées de cette collision. Cependant, le matériau est rarement purement spéculaire, mais répond à un certain facteur de rugosité. La lumière n'est ainsi pas reflétée dans une seule direction, mais sur tout un cône, en fonction de la valeur de rugosité de la surface, ou « facteur spéculaire » dans la terminologie de Phong. Ainsi, connaître la réflexion selon un certain angle avec une certaine rugosité nécessite de calculer l'intégrale dans un cône de direction. Plus la rugosité est forte, plus le cône est large, jusqu'à atteindre tout l'hémisphère dans le cas d'une réflexion purement diffuse. Pour calculer la réflexion, nous utilisons les échantillons de la cube-map situés à l'intérieur du cône, que nous multiplions par le facteur issu de la BRDF.

Dans le cas d'une BRDF simple tel que celle de Phong, nous pouvons précalculer ces intégrales et les stocker. Nous remarquons de plus que la fréquence des données diminue avec le facteur spéculaire. En effet, le préfiltrage d'une texture en fonction du facteur spéculaire de Phong ressemble au calcul d'un flou gaussien qui deviendrait de plus en plus gros. Une opération de flou est en réalité un filtre sur les hautes fréquences de l'image, qui sont éliminées, réduisant par la même occasion la fréquence d'échantillonnage nécessaire au stockage. Une image floue à besoin de moins d'échantillons qu'une image nette.

Pour simuler l'influence de la rugosité sur la réflexion, il suffit ainsi de stocker des versions floutées de la cube-map et d'utiliser la bonne version en fonction de la rugosité. Heureusement, une technique simple va permettre de faire cette opération de manière très efficace. Il s'agit du mécanisme des mip-maps. Ce mécanisme permet de stocker une pyramide de texture, chaque niveau de la pyramide voyant sa taille en pixel divisée par deux sur les deux axes. Comme nous cherchons à stocker des versions de plus en plus floues d'une texture, nous allons utiliser ces mip-maps de plus en plus petits pour les stocker. Le flou va

permettre de masquer le faible nombre de pixels utilisé par les mip-maps les plus faibles. Le hardware est ensuite capable d'aller chercher n'importe quel niveau de mip-map très efficacement. De plus, nous pouvons automatiquement faire un mélange entre deux mip-maps successifs afin d'adoucir les transitions d'une réflexion nette à une réflexion floue.

Pour conserver une relation physiquement correcte entre la texture de flou utilisée et le degré de rugosité, nous avons utilisé un outil dédié : « Cubemapgen » d'AMD. Ce logiciel permet de partir d'une cube-map de base capturée en haute résolution dans le moteur de jeu et de calculer chacun des niveaux de mip-maps en fonction du facteur spéculaire. L'outil s'assure également de la concordance au niveau des délimitations entre les faces des cube-maps. Le flou appliqué ne doit en effet pas s'arrêter aux bords entre les faces du cube, mais interpoler entre elles. Une discussion de ce problème spécifique est disponible, par Ignacio Castaño[59]. L'outil « Cubemapgen » a été mis en open source, et devient ainsi améliorable par la communauté des développeurs. Le blog de Sébastien Lagarde[60], développeur chez DONTNOD, fourni des informations détaillées sur l'outil et son utilisation dans le cadre de la réflexion basée sur la physique. Ce type de texture se nomme « Prefiltered mipmapped radiance environment map » (PMREM), soit « texture préfiltrée de stockage de la radiance de l'environnement, sous la forme d'une pyramide de textures ».

4.4.b Adéquation de la technique avec la forme de l'objet

Lorsqu'il est question de réflexion ambiante, nous pouvons séparer les objets en deux groupes qui ne vont pas être sensibles aux mêmes variations de l'environnement.

Objets organiques et complexes

Tout d'abord, nous avons les objets organiques et complexes qui présentent peu de grandes surfaces planes. Ces objets ont l'avantage d'une complexité forte de la relation entre la surface et la réflexion. Il est ainsi difficile de savoir visuellement dans quelle direction va la réflexion et donc quel élément du décor doit se refléter à un endroit précis. Ces objets sont suffisamment chaotiques pour que nous puissions calculer la réflexion spéculaire ambiante en tenant compte uniquement de la direction de la réflexion, et non de la position du pixel à calculer. Ce manque d'influence de la position du pixel élimine tout phénomène de parallaxe

59 Ignacio Castaño, « Seamless Cube Map Filtering », 2012, http://the-witness.net/news/2012/02/seamless-cube-map-filtering/.

60 Sébastien Lagarde, 2012, http://seblagarde.wordpress.com/.

dans la réflexion. Si la parallaxe est imperceptible, une même cube-map peut être utilisée pour tout un objet. Chaque objet va donc se voir assigner une cube-map, la plus proche, et il faudra placer celles-ci aux emplacements les plus représentatifs de chaque ambiance lumineuse.

La technique des cube-maps avec niveaux de flou fonctionne très bien avec les objets organiques et permet de donner une réflexion spéculaire plus ou moins floue en réaction à une ambiance lumineuse fixe.

Par contre, certains cas posent problème. Tout d'abord les objets dynamiques (personnages, véhicules…) qui se déplacent d'un endroit à l'autre du décor ne peuvent pas utiliser ce système d'association fixe d'un objet à la cube-map fixe la plus proche, puisqu'ils se déplacent et donc changent d'environnement lumineux local. Si nous changeons de cube-map en temps réel, la différence de réflexion d'une image à la suivante risque d'être très visible. Utiliser plusieurs cube-maps par objets afin d'obtenir des transitions serait également lourd, au niveau du coût par pixel. Les cube-maps utilisées seraient les plus proches de l'objet, en modifiant leurs poids en fonction de la distance. Le calcul de ces poids et du système de transition sera abordé un peu plus loin.

Les objets plans.

Le second type d'objet présente de grandes surfaces planes, organisées de manière cohérente, telles que le sol ou des murs droits. Ces objets ne permettent pas les mêmes approximations que les objets organiques, car nous percevons immédiatement lorsque la réflexion n'est pas correctement localisée, et que la sensation de parallaxe n'est pas respectée au cours du mouvement de la caméra. La parallaxe est un phénomène lié au rapport entre la position d'un observateur et l'éloignement des objets qu'il observe. Lors du déplacement du point d'observation, les objets proches vont très rapidement changer de position visuelle, alors que les objets très lointains ne bougeront pratiquement pas. Ce décalage qui s'opère entre les différents plans selon la distance s'observe également dans la réflexion. La position du point de réflexion a une plus grande importance lorsque les objets reflétés sont proches. C'est pour cette raison qu'une cube-map, qui stocke la réflexion en fonction d'une direction, considère l'environnement comme situé à une distance infinie, et donc insensible à la parallaxe.

La technique des cube-maps est ainsi plus difficile à utiliser dans le cas de ces objets du fait de ce manque de parallaxe. L'environnement lumineux d'une grande surface plane est également difficile à reproduire avec une seule cube-map locale, puisque la surface est assez

grande pour traverser plusieurs ambiances lumineuses bien distinctes. Faire la transition entre plusieurs cube-maps pour une même surface n'est pas trivial. En contrepartie, ces objets sont plus simples, constitués de plans réguliers, et nous pouvons donc utiliser des algorithmes qui exploitent cette simplicité. Nous avons déjà évoqué une technique relativement efficace pour calculer dynamiquement la réflexion d'un plan unique, tel que le sol.

Les objets constitués de grandes surfaces planes ne permettent pas de résumer le calcul de leur réflexion comme dépendant uniquement de la direction, nous devons prendre en compte également la position du pixel à calculer afin de reproduire la parallaxe.

Figure III.15: Importance de la parallaxe pour la réflexion des objets non organiques. À gauche, un mur planaire avec réflexion sans parallaxe puis avec. À droite, un objet organique sans puis avec parallaxe. La réflexion de l'objet organique n'a pas changée qualitativement alors que celle du mur semble complètement différente.

5 Techniques modernes récentes

Nous allons voir plusieurs techniques récentes qui tentent de résoudre le problème de la lumière ambiante diffuse, mais surtout spéculaire.

5.1 Cone tracing global illumination

La technique que nous allons étudier est très prometteuse et généraliste. Elle semble être la voie à suivre dans les prochaines années pour s'approcher de la qualité du rendu précalculé. Elle a été présentée par Cyril Crassin dans une publication[61] et dans sa thèse[62]. Cette technique se base principalement sur une structure spatiale complexe qui va stocker les données d'illumination. L'étape suivante de l'algorithme est de décomposer la BRDF d'un point d'une surface en un petit nombre de « lancés de rayons coniques » à travers cette structure.

Figure III.16: Le « Cone Tracing » en action.

L'algorithme se décompose en 3 phases :

- injection des lumières directes dans la structure (soleil, lumières ponctuelles…)
- propagation des lumières dans la structure en fonction de la géométrie et création des « mip-maps »
- échantillonnage de la structure pour chaque pixel de l'image finale

La structure contient préalablement les informations de géométrie permettant, au moment de la propagation, d'éviter de laisser passer la lumière à travers les murs. Nous pouvons ainsi obtenir une occlusion dans l'éclairage secondaire, et même plusieurs rebonds d'éclairage. Le

61 C. Crassin et al., « Interactive Indirect Illumination Using Voxel Cone Tracing », in *Computer Graphics Forum*, vol. 30, 2011, 1921–1930.

62 Cyril Crassin, « GigaVoxels: A Voxel-Based Rendering Pipeline For Efficient Exploration Of Large And Detailed Scenes », *Université de Grenoble*, Thèse (2011).

décor statique est inséré dans la structure une seule fois. Les objets en mouvement sont « voxélisés » à chaque image puis insérés dans une copie de la structure des objets statiques.

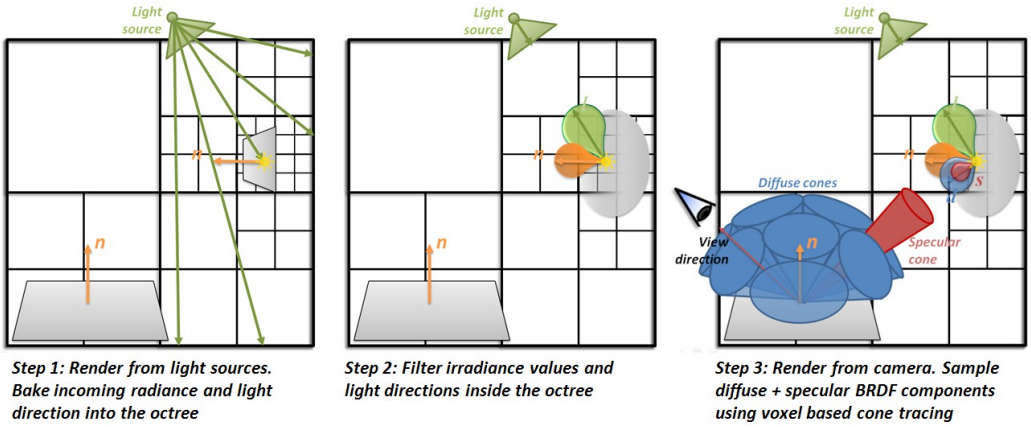


Figure III.17: Les trois étapes de l'algorithme de « Cone Tracing »

La structure est un « sparse voxel octree », c'est-à-dire une structure hiérarchique de stockage de points, qui a l'avantage de stocker uniquement les points utiles, par exemple les points proches du décor. La structure applique successivement une décomposition qui divise l'espace 3D selon les trois axes, ce qui produit huit subdivisions à chaque niveau. Les cellules de l'octree qui ne sont pas occupées par des objets n'ont pas besoin d'être subdivisées et ne prendrons ainsi pas de place en mémoire. C'est une structure complexe à gérer et plutôt nouvelle sur la carte graphique. Chaque échantillon de la structure est constitué de 6 couleurs, chacune représentant la lumière passant par le cube selon une direction donnée parmi les trois axes (X, Y et Z) dans les deux sens, formant effectivement six directions.

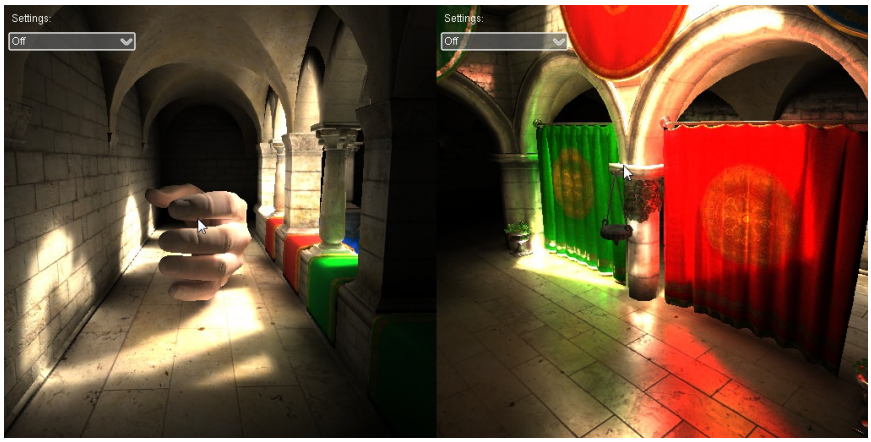


Figure III.18: Le résultat final du "Cone Tracing".

Les RSM (« Reflective Shadow Map ») sont des textures qui représentent la scène vue depuis une lumière. La particularité de ces textures est de regrouper toutes les informations nécessaires pour calculer les réflexions secondaires produites par cette lumière. Ainsi, à partir de ces textures, nous connaissons les points dans l'espace 3D qui sont touchés par cette

lumière ainsi que les directions et couleurs des reflets une fois filtrés par la surface. Les RSM contiennent généralement ces trois informations dans plusieurs textures : position, direction et couleur.

Une fois les RSM calculées pour chaque lumière, celles-ci sont injectées dans la structure, en allant chercher pour chaque point de la structure sa position associée dans la RSM. Nous pouvons ainsi savoir si une réflexion lumineuse est présente en un point ou non, ainsi que ses caractéristiques. Après l'injection, les informations lumineuses sont propagées aux niveaux hiérarchiques supérieurs, ce qui permettra d'obtenir efficacement l'influence lumineuse de n'importe quel volume, en choisissant le niveau hiérarchique à utiliser en fonction de la taille du volume.

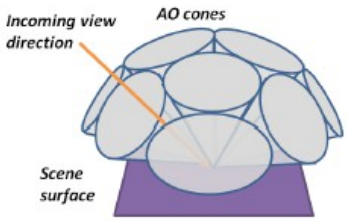


Figure III.19: Cônes d'approximation de la diffuse

L'intégration de la BRDF est alors approximée par plusieurs cônes. Chaque cône permet de calculer la quantité de lumière arrivant selon un certain angle et selon une certaine ouverture. Un éclairage de Phong est séparé en deux parties, une partie spéculaire constituée d'un cône orienté selon la réflexion par rapport à la surface et une partie diffuse découpée en une dizaine de cônes répartis sur l'hémisphère.

Le calcul de la lumière selon un cône peut se faire en décomposant le cône en échantillonnages de sphères de plus en plus grandes, et donc de plus en plus éloignées les unes des autres. La structure hiérarchique nous permet de calculer la lumière dans chacune de ces sphères efficacement.

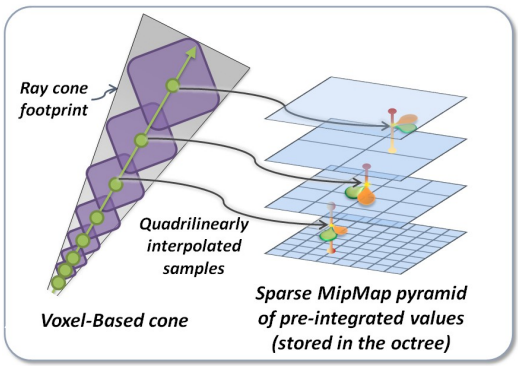


Figure III.20: Division d'un cône en échantillons sphériques

Les résultats sont excellents, mais l'algorithme est complexe et nécessite un matériel haut de gamme ainsi que des langages évolués tels que CUDA. La quantité de données à stocker est assez importante, la gestion d'objets dynamiques, bien que possible, risque d'être très coûteuse dans un cas d'utilisation réelle, tel que celui d'un jeu. Néanmoins, cette technique est une réelle avancée vers une réflexion ambiante spéculaire en temps réel.

5.2 Light Propagation Volume pour le spéculaire

Crytek a présenté l'un des premiers algorithmes d'illumination globale diffuse[63] pouvant tourner efficacement sur les consoles actuelles (PS3/360). Les développeurs ont également introduit une technique de réflexion spéculaire utilisant le même système.

Figure III.21: Le « Light Propagation Volume » pour la diffuse et le brouillard.

L'algorithme se présente en trois étapes de la même manière que celui du « Cone Tracing » :

- Injection des lumières dans la structure (soleil, lampes…)
- propagation des lumières dans la structure
- échantillonnage de la structure pour chaque pixel de l'image finale

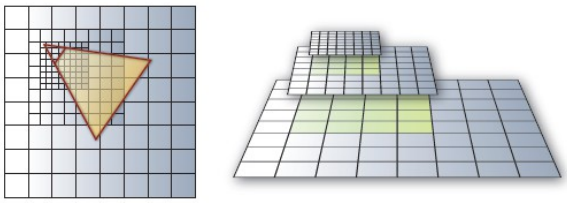


Figure III.22: Structure hiérarchique des trois grilles

Les différences entre les deux algorithmes se présentent néanmoins dans tous les détails, rendant cette technique bien moins généraliste, mais beaucoup plus rapide et efficace, tout en conservant une qualité raisonnable. La structure, tout d'abord, n'est plus un « sparse voxel octree », mais une simple grille en trois dimensions, chaque point de la grille stockant les échanges lumineux sous la forme d'harmoniques sphériques du niveau deux, c'est à dire quatre couleurs. Afin de prendre en compte de grandes échelles tout en conservant une qualité plus importante près du joueur, plusieurs grilles indépendantes sont utilisées. Dans leur cas, trois grilles sont utilisées, la première permet d'avoir des détails très proches du joueur, la seconde capte les effets à moyenne distance et la dernière regroupe les effets à grandes échelles, telles que celle des bâtiments.

L'étape de diffusion n'est plus hiérarchique, mais itérative. L'algorithme ressemble un peu à celui d'un fluide. Il s'agit de propager les valeurs de luminosité des cellules en fonction des

63 A. Kaplanyan et C. Dachsbacher, « Cascaded light propagation volumes for real-time indirect illumination », in *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, 2010, 99–107.

directions liées aux échantillons lumineux. Chaque itération de l'algorithme va propager un échantillon de la grille sur ces voisins, au nombre de 6 en 3D. Une fois un nombre suffisant d'itérations calculées, la lumière a été propagée dans tout l'espace. Au cours de cette phase, il est possible d'utiliser les occlusions secondaires afin de stopper la propagation de la lumière en cas d'obstacle entre deux cellules.

Figure III.23: Avec (en haut) et sans (en bas) illumination indirecte

Le principe de la dernière étape est simple et rapide : pour chaque pixel, il suffit d'interpoler entre les échantillons de la grille qui entoure la position 3D du pixel. Nous obtenons un éclairage diffus secondaire. Pour la partie spéculaire, la proposition faite par les développeurs de chez Crytek est d'accumuler les échantillons le long d'un rayon depuis la surface dans la direction du reflet. Dans l'idéal, le rayon devrait aller jusqu'à toucher un obstacle. Dans la pratique, un nombre réduit de cellules parcourues suffit à obtenir l'effet d'une réflexion floue. Plus le nombre d'échantillons accumulés le long du rayon est important et plus la réflexion sera précise et nette, jusqu'à un maximum de netteté fixé par la résolution de la grille. Les développeurs de Crytek utilisent quatre échantillons dans leurs tests.

Figure III.24: Réflexion spéculaire floue

La technique est très impressionnante par son efficacité, même si elle a fini par être désactivée sur consoles pour gagner en performance et obtenir un jeu plus fluide. L'équipe de développement de Crytek a clairement prouvé que ce type de technique est possible sur la génération de console actuelle. Au niveau qualité, de nombreux artefacts sont visibles dans certaines conditions, comme de la lumière passant à travers les murs. De plus, l'augmentation de la résolution de la grille 3D produit une explosion du nombre de calculs, limitant la grille à de petites résolutions (32^3 par exemple). L'effet apporte néanmoins une belle ambiance lumineuse diffuse, et à un bon potentiel également pour les réflexions spéculaires.

5.3 Real Time Local reflection

Tout d'abord présentée par « Graham » sur le forum « beyond3D »[64], l'idée d'afficher des réflexions en allant simplement chercher dans l'espace-écran de l'image a ensuite été formalisée et implémentée par les développeurs de Crytek pour la version PC du jeu « Crysis 2 »[65].

Figure III.25: Sans (gauche) et avec (droite) Real Time Local Reflection

En effet, dans de nombreux cas, la surface à refléter est présente à l'écran en même temps que la surface réflexive. C'est notamment le cas pour les sols et les réflexions de contacts, qui sont très importantes visuellement. Les objets reflétés peuvent être retrouvés dans l'image normale de la scène, qui a déjà été calculée et donc ne coûte plus rien. Pour ce faire, nous avons besoin de la couleur de l'image, bien sûr, mais également de la texture de profondeur, indiquant la distance entre la caméra et chaque pixel. Le calcul de la réflexion d'un pixel de l'image selon une certaine direction est effectué par « ray marching » de la texture de profondeur. Concrètement, il s'agit de parcourir les pixels situés sur la trajectoire 2D de la réflexion, jusqu'à trouver un pixel dont la profondeur le place à proximité de cette trajectoire en 3D. Ce

64 Graham, « Screen space reflections », 2010, http://forum.beyond3d.com/showthread.php?t=56095.

65 Pierre-Yves Donzallaz et Tiago Sousa, « Lighting in Crysis 2 », Conférence GDC 2011.

pixel est ainsi une collision du vecteur de réflexion avec la texture de profondeur du décor, et sa couleur est utilisée en tant que couleur de réflexion.

Malheureusement, il n'est pas toujours possible de trouver un tel pixel. Si la trajectoire de réflexion atteint les bords de l'image sans avoir trouvé de collision, aucune couleur de réflexion ne pourra être déterminée. Pour adoucir la transition entre les pixels qui ont une réflexion et ceux qui n'en ont pas, toute la magie va consister à trouver des heuristiques intéressantes afin d'anticiper la présence de réflexion. Crytek utilise la proximité du pixel avec les bords de l'écran, ainsi que l'angle entre la réflexion de la surface et la direction de la caméra. En effet, si la réflexion de la surface pointe vers la caméra, peu de pixels seront traversés avant que la trajectoire ne passe derrière la caméra, rendant impossible toute collision. L'utilisation de ces deux paramètres permet de limiter les discontinuités dans la réflexion en l'atténuant lorsqu'elle s'approche des cas qui ne sont pas gérés. La complexité de l'algorithme est assez élevée. Le coût de la recherche de collisions avec la texture de profondeur est prohibitif sur console. Pour alléger les calculs et améliorer la qualité, la recherche se fait sur une texture en plus petite résolution et plus floue. La faible résolution adoucit les discontinuités entre les pixels voisins.

Figure III.26: Sans (gauche) et avec (droite) Real Time Local Reflection

Cette technique est intéressante notamment pour le sol, que le joueur regarde en général en même temps que le décor. Il est néanmoins important de n'appliquer cette réflexion que légèrement, comme un plus, afin de ne pas mettre en avant ses limites. Dans « Crysis 2 », elle est utilisée en addition à une cube-map classique.

5.4 Raytracing de plans au pixel

La vidéo temps réel « Samaritan »[66] de présentation de la version DirectX 11 du moteur Unreal Engine 3 est impressionnante à plus d'un titre et notamment pour sa gestion des réflexions floues sur l'ensemble du décor.

En effet, le moteur utilise une réflexion à base de quadrilatères texturés qui forment des « proxies » du décor. Un proxy sert à remplacer un objet 3D par un autre, plus simple, ici un quadrilatère. Les éléments importants de la scène 3D voient leur image capturée chacune dans une texture, selon un plan qui englobe au mieux l'élément. Tous les objets de la scène peuvent alors accéder, sur le processeur graphique, à la liste de ces textures, accompagnées de la description du quadrilatère associé. Un calcul analytique permet d'effectuer un lancer de rayon au travers de ces plans afin de sélectionner la collision la plus proche dans la direction du reflet. Ainsi, tous les objets sont capables de refléter ces proxies en fonction de la normale et de la position du point de la surface.

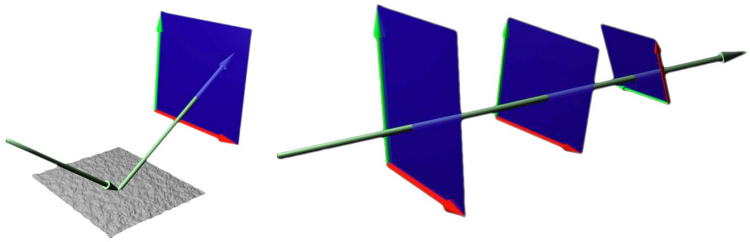


Figure III.27: Collision du rayon réfléchi avec les quadrilatères « proxies »

Les objets peuvent avoir n'importe quelle forme, organique ou constituée de grand plan, sans poser de problèmes. La parallaxe est gérée correctement. La seule limitation est de réduire le décor à une série de plans, dont le nombre est le principal facteur de complexité. La réflexion peut être rendue floue en utilisant la technique du mip-mapping sur les textures capturées. Les proxies peuvent également être utilisés pour approximer la réflexion de lumières ponctuelles, en remplacement du calcul de l'équation complète de la réflexion de chaque lumière par pixel.

Les développeurs ont également mis en place une technique d'anisotropie pour la réflexion. L'anisotropie est le phénomène d'étirement de la réflexion lorsque la surface est rugueuse et l'angle de vue rasant.

66 Mittring et Dudash, « The Technology Behind theDirectX 11 Unreal Engine“Samaritan” Demo ».

Figure III.28: La réflexion à base de proxies d'après Epic,

En plus de cette technique, utilisée principalement pour figurer les objets très lumineux, une technique de « ray marching » dans un « signed distance field » est utilisée. Un « ray marching », comme nous l'avons déjà précisé, est cette opération de parcourt des échantillons dans une direction donnée afin de trouver les collisions. Les cellules font ici partie d'un volume tridimensionnel et sont donc des « voxels » et non des pixels comme en deux dimensions. Le « signed distance field », ou « champs de distances signées » est une technique d'accélération du « ray marching » qui permet de trouver rapidement la collision d'un rayon avec une surface. Le volume du « signed distance field » est capturé une seule fois, à partir du décor statique. Cette technique permet de prendre en compte très rapidement les occlusions des proxies par le décor fixe. Ces occlusions ne peuvent produire qu'un reflet totalement noir, obscurcissant les autres réflexions, car le volume de collision ne contient qu'une information binaire indiquant si la cellule est vide ou non. Si le volume contenait également les couleurs du décor, la réflexion pourrait être colorée. L'utilisation de ce volume permet de représenter grossièrement l'occlusion du décor et de limiter l'usage des proxies aux objets lumineux importants.

Figure III.29: Sans (gauche) et avec (droite) l'occlusion par le volume 3D

Les objets dynamiques complexes, tels que les personnages, sont ajoutés par une technique classique de réflexion ne pouvant être utilisée que sur le sol, c'est-à-dire un plan placé à une hauteur fixe.

Cette vidéo nous permet d'observer pratiquement pour la première fois un décor en temps réel avec des réflexions floues sur tout le décor. Malheureusement, la technique est lourde, réservée aux dernières cartes graphiques, et nécessite beaucoup d'interventions manuelles dans la création des proxies. Cette technique basée sur des proxies est très proche de notre méthode que nous allons présenter plus loin. Nous nous intéresserons d'ailleurs aux manières de faciliter la création des proxies. Nous étendrons également la technique à la réflexion de maillages convexes, mais nous limiterons par contre la réflexion à un seul plan, le sol. Notre technique à l'avantage d'être légère, et compatible avec le matériel des consoles actuelles.

6 Création de techniques personnelles

6.1 Réflexions semi-planes précalculées

Au cours du développement du jeu, nous avons eu besoin de présenter au joueur des décors sous la pluie, ou juste après une pluie. Dans ces cas là, le sol présente une forte réflexivité, un aspect mouillé qu'il n'est pas aisé d'obtenir par les techniques courantes en temps réel.

6.1.a Contexte d'utilisation

L'objet le plus important dans cet aspect est le sol, un objet en général plat à une échelle de quelques mètres, mais avec des dénivelés à une échelle plus grande. La technique classique de réflexion par un second rendu de la scène vue depuis une caméra reflétée présente plusieurs inconvénients qui l'ont rendu impraticable dans notre cas. Notamment, rendre une seconde fois la scène par plan de réflexion pose beaucoup de contraintes pour un jeu vidéo qui consacre déjà toute sa puissance à rendre la scène une première fois. De plus, la prise en compte des différences d'élévations du sol, à petite ou grande échelle, n'est pas triviale. D'un point de vue artistique, nous souhaitions également des réflexions moins précises et plus floues, plus proches de l'aspect mouillé d'une rue sous la pluie. Nous cherchions un algorithme dont le coût soit faible pour la carte graphique et négligeable pour le processeur principal. La réflexion pourra ainsi être utilisée librement sans ajouter de contraintes de performances.

Les décors du jeu sont principalement statiques, ce qui nous oriente immédiatement vers les techniques basées sur un précalcul. Deux aspects sont primordiaux dans un tel cas, tout d'abord la structure de stockage des données précalculées et ensuite l'algorithme qui va lier cette structure aux données nécessaires pour calculer la réflexion d'un pixel donné. La structure de stockage doit être la plus légère possible tout en regroupant au mieux les informations qui sont pertinentes. L'algorithme doit être capable de retrouver rapidement les données concernant un pixel en fonction de sa position et de la direction de réflexion. Ces deux aspects partagent de nombreuses contraintes, et nous pouvons observer une sorte de balance entre la taille de la structure et l'efficacité de l'algorithme. Améliorer l'un détériore généralement l'autre.

6.1.b Les paramètres du format de texture

Au moment du précalcul, la seule variable que nous ne connaissons pas est la position de la caméra, qui sera modifiée à chaque image. La première étape dans la simplification du problème a été de réduire les paramètres de la réflexion à une simple direction et à une distance, en négligeant la position réelle du pixel. La différence est subtile, mais très importante. En effet, nous considérons que la couleur de réflexion d'un point ne changera qu'en fonction de la direction d'où la caméra l'observe et de la position sur le plan perpendiculaire à cette direction, sans prendre en compte la distance de la caméra à ce plan, et donc sa position réelle.

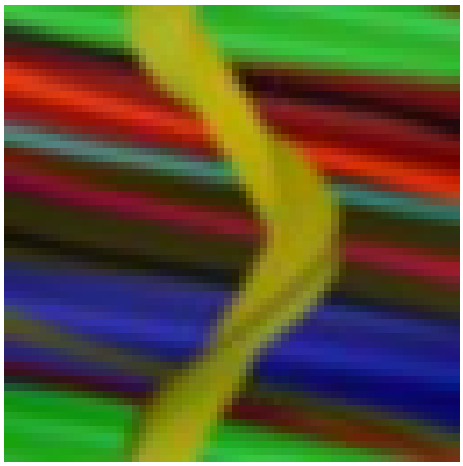


Figure III.30: Exemple de texture statique de réflexion précalculée.
La scène est située à l'intérieur d'un cube aux faces colorées, avec un unique cube jaune placé au centre. En Y nous avons la direction de capture qui varie, en X nous avons à chaque fois une coupe de la scène. La structure en spirale provient de la rotation de la direction de capture.

Cette direction est également ramenée à une direction sur un plan, donc une direction bidimensionnelle. Cette simplification se justifie, car nous nous intéressons principalement à la réflexion liée à l'effet de Fresnel d'un objet de forme globalement plane. Cette réflexion apparaît à des angles rasants entre la surface et la caméra. Si la caméra est plus haute, la couleur ne changera pas beaucoup, mais commencera à disparaître. La couleur dépend donc essentiellement de la direction perpendiculaire à l'axe normal du plan. Une direction sur un plan bidimensionnel peut se résumer à une seule valeur, qui est l'angle par rapport à l'un des deux axes.

Grâce à ces deux simplifications, les données à stocker commencent à être moins nombreuses. Le premier paramètre est l'angle de réflexion, et le second est une valeur de décalage le long du vecteur perpendiculaire à l'axe de réflexion. De cette manière, nous allons stocker des lignes passant par un point et tournant à 360 degrés autour de ce point. Chaque ligne est associée à une direction, et chaque pixel de cette ligne à un décalage perpendiculaire à cette ligne. Concrètement, nous avons choisi de regrouper les informations, non par position

spatiale sur un plan, mais par un angle et une distance. Pour chaque angle, nous stockons une bande complète de l'ambiance lumineuse dans cette direction. La façon la plus simple de stocker ensemble ces « coupes » est une texture en deux dimensions, avec chaque coupe placée l'une au-dessus de l'autre. Ainsi, les données tiennent dans une texture 2D et nous pouvons concentrer la résolution sur la variation de luminosité parallèlement à la direction de réflexion au lieu de gâcher de la mémoire sur les variations spatiales le long de celle-ci.

6.1.c Le calcul de l'interpolation

Une seconde contrainte vient s'ajouter à celle de la structure de stockage, il s'agit de la capacité d'interpolation. En effet, nous stockons peu d'informations pour calculer la réflexion, mais nous souhaitons qu'elle évolue doucement, sans sensation de crénelage ni de discontinuités. Ainsi, la faible résolution va nous apporter une réflexion floue, mais pas pixelisée. Grâce à notre structure de stockage, les échantillons dont la direction et la distance parallèle sont proches le sont également dans la texture 2D. C'est cette caractéristique qui permet d'interpoler entre un point et ses voisins afin d'obtenir des transitions douces.

La rotation de la direction de réflexion autour du point central va provoquer une forme de spirale dans la texture 2D, ce qui signifie que des couleurs qui sont proches vont se retrouver légèrement orientées dans une direction, mais cet effet n'est pas gênant tant que la résolution est suffisante. Afin de compenser la faible résolution de la texture et d'obtenir une réflexion très douce, nous appliquons un flou gaussien sur la texture 2D.

La technique de l'interpolation bilinéaire matérielle des textures 2D est idéale pour assurer l'interpolation entre pixels. En effet, celle-ci va utiliser les quatre échantillons les plus proches afin d'obtenir une valeur évoluant doucement entre deux points spatialement proches ou deux directions adjacentes. De plus, le « bouclage » de la coordonnée y, représentant l'angle de réflexion, peut être géré simplement. Une fois parcouru tout le périmètre du disque représentant les angles de réflexions possibles, nous revenons en effet au point de départ et nous souhaiterions pouvoir interpoler entre ces deux points, le premier et le dernier. Ce bouclage peut en théorie être géré directement par la carte graphique, qui assurerait une interpolation correcte comme dans le cas d'une texture qui se répète. Malheureusement, notre algorithme ne boucle pas à travers une répétition. Nos coordonnées de textures passent d'un coup de un à zéro lors de la boucle, alors que lors d'une répétition, les valeurs au-dessus de un sont utilisées afin d'indiquer le bouclage.

La carte graphique présente alors plusieurs artefacts, car elle ne sait pas gérer cette discontinuité dans les coordonnées de texture. Le mécanisme de « mip-mapping » de la texture, notamment, va sélectionner automatiquement une version très peu détaillée de la texture sur les pixels du bord, car il base son calcul sur le gradient des coordonnées de textures, c'est-a-dire leur vitesse de variation. La solution la plus simple est de désactiver l'utilisation des « mip-maps », mais s'ils sont indispensables, nous pouvons passer manuellement le numéro du niveau de « mip-map » à utiliser, en se basant directement sur les coordonnées de textures originales de la surface au lieu de celles que nous calculons. La fonction HLSL à utiliser dans ce cas est « tex2Dlod », au lieu de « tex2D ».

6.1.d Code HLSL

Voici le code HLSL permettant d'utiliser notre format de texture. Le paramètre « Size » est un facteur qui dépend de la taille de la surface de réflexion et qui agrandit ou rétrécit l'image de réflexion en conséquence.

```
// facteur de taille de la réflexion
float Size = 0.75f;

// Direction du point de la surface vers la caméra
float3 View = normalize(ViewPosition – PointPosition);

// Facteur de Fresnel, limitant la réflexion aux angles rasants
float Fresnel = 1.0f-saturate(pow(View.y, 5.0f));

// Direction de la caméra ramenée sur le plan 2D XZ
float2 View2D = normalize(CamVector.xz);

// Calcul du décalage perpendiculairement à la direction
// de la caméra, L'inversion des axes X et Y de View2D
// effectue une rotation en 2D
float OffsetX = saturate(dot(uv - 0.5f, -View2D.yx) * Size +0.5f);

// Calcul de l'angle absolu entre la direction de la caméra et
// l'axe Y du monde
float Angle= clamp(acos(-View2D.y) / PI, 0.0f, 0.99f);

// L'angle récupère un signe et il est ramené entre 0 et 1
Angle = 0.5f + sign(View2D.x) * DirAngle * 0.5f;

// Utilisation des valeurs calculées pour échantillonner la
// texture de réflexion
// Le facteur de Fresnel sert simplement de multiplicateur
float4 RefColor = tex2D(Stroke, float2(OffsetX, Angle)) * Fresnel;
```

6.1.e Contraintes de l'algorithme simple

Le problème principal de cette technique concerne les objets placés à l'intérieur de la surface de réflexion. En effet, si nous représentons le sol par un carré, le résultat fonctionne très bien tant que l'ambiance lumineuse provient de l'extérieur de ce carré. Par contre, étant donné que nous ne possédons qu'une seule bande de valeurs par direction, nous ne pouvons pas gérer des objets à l'intérieur du carré. Pour cela, nous aurions besoin de stocker plusieurs valeurs dans une direction, les collisions du plan avec chaque objet.

La solution serait de stocker effectivement plusieurs bandes par directions, mais le nombre nécessaire dépend de la complexité de la scène. Des subdivisions régulières reviendraient à utiliser une texture 3D, ce qui augmenterait fortement la quantité de données à stocker.

Un second problème est également gênant. La résolution du format n'est pas homogène sur tout le plan du sol. En effet, la distance spatiale entre deux coupes augmente au fur et à mesure que nous nous éloignons du centre. Les zones à proximité du point de rotation de l'angle de réflexion vont être capturées par beaucoup de « coupes ». Par contre, les zones éloignées, près des bords du carré, seront traversées par beaucoup moins de « coupes ». Cette variation du taux d'échantillonnage n'est pas idéale et produit une qualité variable en fonction des endroits.

6.1.f Décomposition de la surface de réflexion selon une grille

Pour contrer ces deux problèmes, notre choix s'est porté sur la décomposition de la surface de réflexion en plusieurs sous plans. En effet, l'algorithme de base utilise un centre de rotation des coupes de direction qui est unique pour toute la surface, quelle que soit sa taille. Nous

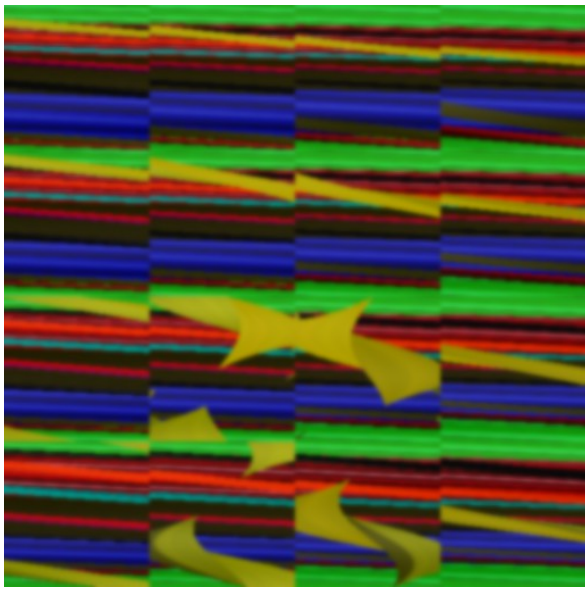


Figure III.31: Une texture tirée de la même scène que précédemment.
Cette fois, la texture est découpée en 4x4 zones. La résolution est mieux répartie et les objets à l'intérieur des zones peuvent être reflétés.

découpons donc la zone de capture en plusieurs carrés, chacun possédant son propre centre de capture. Ainsi, si des objets sont présents dans un des carrés, sa réflexion sera visible dans les autres carrés. Nous avons donc résolu le premier problème dans une certaine mesure. La résolution est également mieux répartie sur la surface du sol.

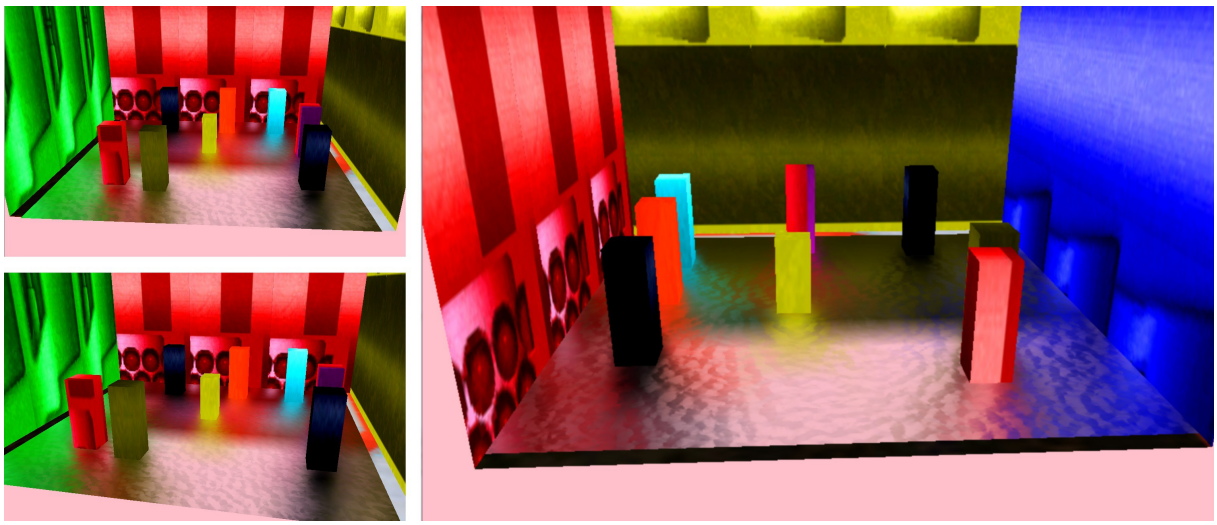


Figure III.32: Trois vues d'une scène de test. La parallaxe dans la réflexion est bien présente, les objets à l'intérieur sont correctement reflétés.

Une fois la réflexion découpée en une grille de carrés, la quantité de mémoire nécessaire n'a pas changé. Par contre, des discontinuités vont apparaître sous la forme de coupures dans la réflexion lorsque nous passons d'un carré à un autre sur la surface. La solution est de complexifier l'algorithme de rassemblement des données afin d'aller chercher la réflexion dans les quatre carrés qui entourent un point donné. Ainsi, nous pouvons faire une interpolation bilinéaire entre les résultats des quatre textures de réflexion et avoir une transition douce tout en obtenant une prise en compte locale des occlusions internes. Le coût de l'algorithme est quant à lui plus élevé. En optimisant correctement et en rassemblant les

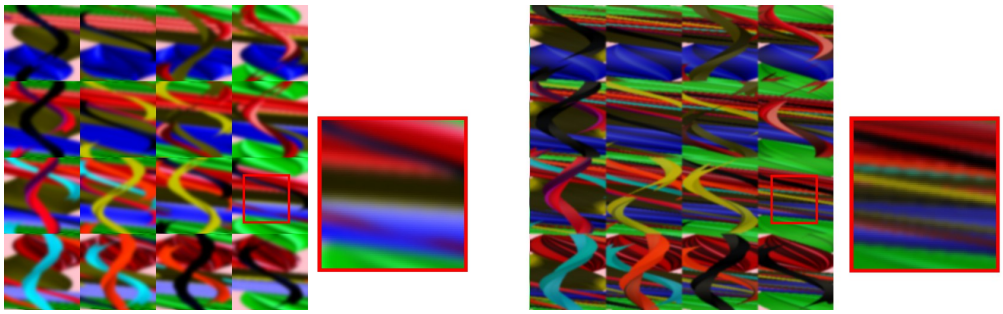


Figure III.33: Deux exemples de textures de réflexion précalculées tirés de scènes similaires.
La première texture est suffisamment floue pour donner un résultat doux, la seconde présente trop de détails et produira une réflexion crénelée et donc moins crédible.

calculs similaires afin d'utiliser les capacités en calculs « vectorisés » de la carte graphique, nous pouvons obtenir un algorithme environ 2 fois plus gourmand que l'algorithme simple.

Quelques détails sont à prendre en compte afin d'obtenir un résultat correct avec cette technique. Les différents plans de la surface étant stockés sous la forme d'une grille dans la texture, nous ne pouvons plus compter sur l'interpolation de la carte graphique pour assurer le bouclage de la texture entre le premier et le dernier pixel d'une rotation complète. Pour pallier cela, il suffit de s'assurer que la première et la dernière coupe de chaque rotation sont exactement les mêmes au niveau des couleurs. Ainsi, l'interpolation parviendra au même résultat des deux côtés de la coupure. Cette technique est appelée « padding », elle consiste à dupliquer une ligne pour éviter l'interpolation à certains endroits. Les coordonnées de textures que nous choisissons doivent également être ajustées afin de ne jamais dépasser les limites du carré associé.

Au niveau de l'application du flou sur la texture, le même problème se pose. Le flou doit boucler en utilisant les valeurs des dernières coupes lors du calcul du flou des premières. Le flou sera ainsi doux au lieu de laisser une marque nette lors du bouclage. Il est également nécessaire de limiter les échantillons utilisés au carré de chacun des plans. Les carrés sont floutés séparément sans déborder des uns sur les autres.

6.1.g La capture de la scène

La capture initiale des coupes de la scène environnante peut être accomplie en utilisant une caméra orthographique, c'est à dire avec une perspective isométrique. Cette caméra tournera autour du centre du plan de réflexion. Le résultat de chacune de ces captures est une bande horizontale de un pixel de haut. Afin de mieux représenter l'ensemble de l'ambiance lumineuse et non une seule bande, nous rendons d'abord une image plus haute, qui capture environ ¼ de la scène. Chaque colonne de la bande est alors utilisée pour calculer une valeur unique, soit en utilisant une moyenne, soit en prenant la valeur maximum. Le résultat est donc une bande de un pixel de haut, chaque pixel représentant toute sa colonne. Pour améliorer l'échantillonnage, la texture intermédiaire peut être calculée dans une résolution plus élevée, puis réduite au moment de l'insertion de la bande dans la texture finale. La résolution intermédiaire n'a pas d'importance sur le calcul temps réel final, puisqu'il s'agit d'un pré-calcul.

La dernière phase dans la création de cette texture est le flou. En effet, sans flou la réflexion calculée est trop pixelisée et elle n'est pas stable lors des mouvements de caméra. La texture présente des fréquences trop grandes qui ne pourront pas être correctement échantillonnées ensuite. La solution est donc de supprimer les hautes fréquences, c'est-à-dire de faire un flou. Ce flou est appliqué au moment de la création de la texture, en précalcul, et n'a donc pas d'incidence sur les performances de l'algorithme.

Le flou peut être appliqué à plusieurs moments, par exemple lors du calcul de la texture intermédiaire à chaque coupe ou bien tout à la fin, une fois la texture complètement calculée. C'est cette solution qui a été choisie, car elle donnait les meilleurs résultats visuels en floutant à la fois sur l'axe X (distance spatiale) et sur l'axe Y (direction). Un flou gaussien de quelques pixels suffit à rendre la réflexion résultante beaucoup plus propre et crédible.

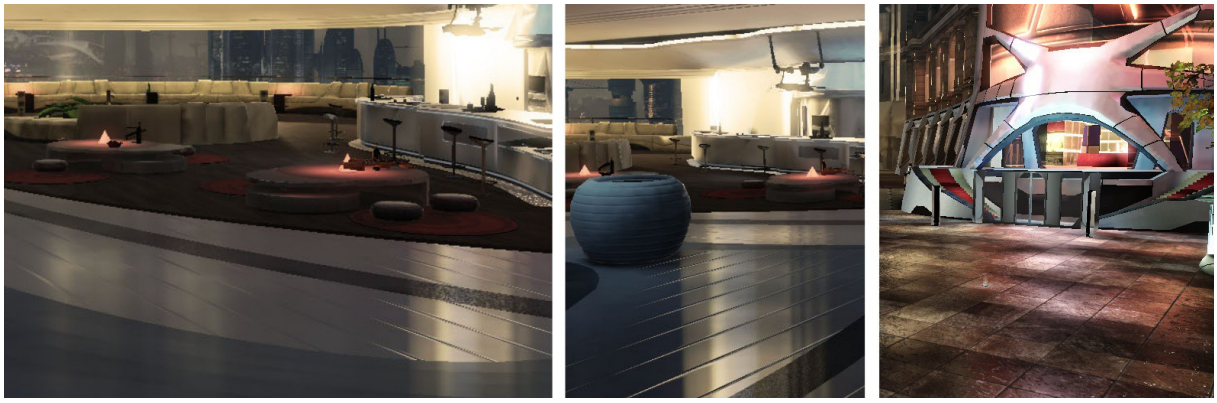


Figure III.34: Intégration des réflexions statiques précalculées dans un prototype plus évolué.

6.1.h Conclusion

En conclusion, cet algorithme est intéressant à plusieurs points de vue. Du point de vue de la « recherche », il propose une réponse à la question du stockage des données de réflexions par un format compact des données essentielles. Du point de vue de la « production », l'algorithme est peu gourmand en mémoire (une texture de 256x256 pour une place par exemple). Le coût au pixel est élevé, mais prédictible et dépendant directement du nombre de pixels utilisant la réflexion, sans coûts fixes.

Les résultats visuels sont variables, certaines scènes fonctionnent très bien, notamment celles avec de grandes plages d'une couleur assez unie. L'aspect allongé de la réflexion est particulièrement adapté à la réflexion aux angles rasants. La normale du sol peut être utilisée pour faire varier l'angle de réflexion d'une manière crédible. Les scènes avec beaucoup de petits objets, et donc une ambiance lumineuse à très haute fréquence, ne présenteront pas une réflexion très crédible avec cette technique.

Le processus de création que nous avons utilisé lors du développement de cette technique a été très intéressant, car particulièrement axé sur l'innovation. La solution n'est pas venue de l'évolution d'une technique largement connue, mais plutôt d'une réflexion sur la théorie, sur les données qui sont nécessaires à la réflexion et sur la manière de les échantillonner pour répartir la résolution aux endroits où elle est la plus utile. Cette technologie sera finalement abandonnée au profit de celle que nous présenterons plus bas. Ce choix est principalement lié à la difficulté de séparer le décor en différentes zones avec chacune un plan de réflexion. La qualité de réflexion n'est pas toujours assez bonne, et la technique permettant la gestion d'occlusions internes est un peu lourde au niveau du coût par pixel.

6.2 Utilisation de cube-maps d'environnement statiques

Nous avons vu précédemment qu'il est possible de précalculer des échantillons de réflexions, vus depuis des positions spatiales précises, par l'intermédiaire de cube-maps. Pour calculer la réflexion d'un point précis, il est nécessaire de décider de la cube-map à utiliser ainsi que de la manière dont se fera la transition entre les cube-maps voisines. En effet, s'il est possible pour les objets fixes du décor de choisir une fois pour toutes une cube-map (par exemple la plus proche), un objet dynamique devra changer de cube-map au cours de ses déplacements. De plus, un élément très grand du décor, comme un sol, devra lui aussi changer de cube-map selon les parties de l'objet afin de refléter l'ambiance locale à chacune d'elle.

6.2.a Création d'un mélange de cube-maps unique

Chaque objet devrait, dans le cas idéal, calculer son propre mélange des cube-maps voisines. Pour simplifier et améliorer les performances, il est possible de calculer une seule cube-map qui sera utilisée par tous les objets dynamiques, et même les objets statiques. Cette nouvelle texture d'environnement va être un mélange de plusieurs cube-maps, en utilisant la position de la caméra comme référence dans le choix des cube-maps voisines à utiliser. Pour ce faire, nous allons devoir trouver un moyen de sélectionner puis de mélanger les cube-maps les plus adéquates. Le but est d'obtenir les réflexions les plus fidèles possible tout en assurant une continuité temporelle et spatiale de la réflexion lors des déplacements de la caméra.

Utiliser cette cube-map unique facilite énormément l'application sur les objets. Ainsi, au lieu de devoir appliquer l'algorithme de sélection et de mélange pour chaque pixel, chaque sommet ou chaque objet, nous l'appliquons une seule fois pour toute la scène. Si les objets proches de

la caméra voient leur ambiance lumineuse locale correctement reproduite, ceux qui sont loin de la caméra n'utilisent pas les cube-maps proches d'eux, mais celles proches de la caméra. Cet effet est notamment visible lors des transitions d'une zone très lumineuse à une zone très sombre. Lorsque la caméra est dans la lumière, tous les objets et même ceux dans l'ombre, se mettent à refléter une ambiance très lumineuse et donc à briller fortement. Pour compenser ce problème, nous avons deux outils à disposition. Tout d'abord, il est possible de choisir pour chaque objet une cube-map unique qui ne changera pas. Ainsi, les objets statiques peuvent utiliser directement l'ambiance locale. Pour tout le reste, nous allons artificiellement réduire l'influence de la réflexion secondaire (la cube-map) en fonction de la distance à la caméra afin de la faire disparaître au bout de quelques dizaines de mètres. Ce choix va également dans le sens d'une simplification des calculs nécessaires pour afficher les objets loin de la caméra (gestion du niveau de détail - LOD), ce qui améliore également les performances. Au-delà d'une certaine distance, l'atténuation de la cube-map est totale, et le shader utilisé peut dès lors être simplifié. Les objets lointains n'ont de toute manière pas vraiment besoin de l'ambiance lumineuse secondaire pour ajouter du détail, car à cette distance la simple forme de l'objet suffit à lui procurer un caractère identifiable. L'influence de la réflexion spéculaire est, de plus, principalement visible lorsque la caméra tourne autour d'un élément. Les objets loin de la caméra verront donc leur aspect spéculaire se modifier beaucoup plus lentement, car la caméra ne pourra pas tourner rapidement autour d'eux.

Les techniques pour simplifier le shader sont diverses. Il est possible d'avoir plusieurs versions du code du shader en fonction de la distance et de basculer entre elles. La quantité de mémoire nécessaire au stockage des différentes versions du shader peut par contre devenir critique, de même que le temps de compilation des différentes versions au cours du développement du jeu. Une compilation complète des versions de tous les matériaux d'un jeu vidéo peut facilement prendre plusieurs heures. Il est également possible d'utiliser le branchement dynamique dans le shader afin de désactiver certaines parties sans pour autant avoir besoin d'un nouveau shader. Le coût d'utilisation du branchement dynamique sur les performances reste important sur console.

Finalement, une solution très simple est de remplacer la texture de cube-map par une texture générique de taille 1x1 pour chaque face. Ainsi, les calculs complets restent exécutés quelque soit la distance, mais au-delà d'un certain seuil, la texture est remplacée par une version simple qui élimine tous les problèmes d'accès mémoire et notamment les fameux « cache-

miss » qui interviennent lorsque la texture ne peut pas être intégralement stockée en mémoire cache. Afin de masquer la transition entre les deux cube-maps, l'influence de celle-ci sur le rendu est diminuée avec la distance et devient nulle avant le remplacement par la cube-map vide.

6.2.b Techniques de mélange de cube-maps

Le passage d'une cube-map à une autre peut se faire de plusieurs manières. Tout d'abord, deux catégories existent, la transition spatiale et la transition temporelle. Dans le cas de la transition spatiale, le poids d'une cube-map changera uniquement en fonction de la position dans l'espace 3D, et non du temps. C'est ce type de transition que nous allons principalement étudier et utiliser. Au contraire, la transition temporelle va modifier le poids de la cube-map en fonction du temps. Le passage d'un objet d'une zone à la suivante déclenche une transition entre les cube-maps associées à ces zones. Cette transition se produit sur une durée fixe, quelle que soit la position de l'objet tant qu'il ne quitte pas la nouvelle zone. La transition temporelle a l'avantage d'être très simple à mettre en place en production, car seule la zone de chaque cube-map doit être spécifiée, sans besoin de zones de transitions (« falloff »). Ainsi, n'importe quel événement ou volume peut être choisi pour déclencher la transition. Par contre, la transition temporelle est moins crédible, car elle introduit un changement indépendant du déplacement du personnage. Ainsi, si la caméra s’arrête juste après l'activation d'une transition, la cube-map changera d'aspect bien que la caméra soit statique. Ce type d'effet est très visible et semble artificiel du point de vue des joueurs. La transition temporelle nécessite également de choisir une durée de transition qui dépend de la taille de chaque zone et de la vitesse de déplacement du personnage. Le but est de rendre la transition la plus invisible possible tout en étant suffisamment rapide pour éviter que le joueur ne sorte de la zone avant la fin de la transition. Une idée qui pourrait être à étudier est de modifier en temps réel la vitesse de transition en fonction de la vitesse de déplacement du personnage. Ainsi, s'il ne bouge pas, la transition s’arrête.

Les développeurs de chez « Crytek » ont utilisé sur « Crysis 2 » une technique d'application de la réflexion des cube-maps qui utilise la technologie du « deffered shading ». Avec cette technique, l'influence de chaque cube-map est simplement ajoutée sur les pixels de l'image qui sont concernés. Le nombre de cube-maps qui influencent un point n'est donc pas limité, et il n'est pas nécessaire de créer une cube-map unique. Cette technique est néanmoins assez lourde et nécessite l'utilisation du « deffered shading » sur tout le jeu.

6.2.c L'outil de contrôle du mélange de cube-maps

Le but principal de l'outil dont nous avons besoin est de permettre au graphiste de décider du volume que chaque cube-map va représenter, mais aussi d'avoir un certain contrôle sur les transitions d'un volume au volume suivant. Lorsque la caméra est dans le volume d'une cube-map, celle-ci peut être utilisée lors du mélange. Cette association d'une cube-map à un volume permet principalement de limiter son influence et de respecter les obstacles entre les différentes ambiances lumineuses. Grâce à ce procédé, nous pouvons éviter de prendre en compte les cube-maps qui sont situées derrière un mur, et ne devraient donc pas avoir d'influence sur l'éclairage local.

La difficulté du choix de la méthode de mélange tient à la balance entre la facilité de réglage et le degré de contrôle désiré. Si nous utilisons une technique automatisée qui permet de passer peu de temps sur le réglage, nous risquons de perdre en degré de contrôle et de nous retrouver avec des cas particuliers impossibles à gérer. Au contraire, un système trop complexe demanderait des temps de réglages manuels rédhibitoires lors de la phase de production du jeu, où le temps alloué à la création de l'éclairage de chaque niveau de jeu est limité. Il s'agit donc de trouver le bon compromis qui fournit un résultat très rapidement tout en permettant de revenir travailler en détail sur les endroits qui ne fonctionnent pas avec le système simple.

6.2.d Volumes sphériques réglés par deux valeurs : « range » et « falloff »

Le système que nous avons mis en place comporte simplement des cube-maps, accompagnées de deux valeurs : le « range » et le « falloff ». Le facteur déterminant est la distance entre la caméra et le centre de la cube-map ce qui forme une sphère d'influence. Cette sphère est

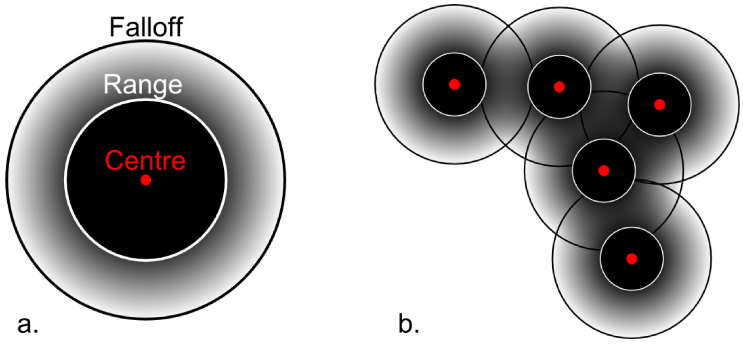


Figure III.35: Calcul de l'influence des échantillons. a. L'influence sphérique d'un échantillon, en noir, en fonction des distances de réglages (range et falloff). b. Cumul de cinq volumes pour former une zone d'influence complexe.

séparée en plusieurs zones en fonction de la distance. En deçà de la distance du « range », la cube-map a une influence totale, au-delà de la distance de « falloff », elle n'en a plus du tout. Entre ces deux limites, l'influence est plus ou moins grande. Il nous reste à décider d'une manière de calculer et de mélanger les influences des cube-maps dont le volume touche la caméra.

Avoir simplement deux valeurs à régler par cube-map permet une mise en place rapide et intuitive de la réflexion. Les deux zones sont représentées visuellement par des sphères lors de la phase de réglage. Pour obtenir un contrôle plus poussé dans les cas délicats, notamment lorsque la topologie du décor est complexe, une possibilité est de simplement rajouter de nouvelles cube-maps pour affiner les transitions. Étant donné que chaque nouvelle cube-map coûte cher en mémoire, nous avons mis en place un système de cube-maps secondaires, qui font référence à une véritable cube-map au lieu d'en stocker une nouvelle. Il s'agit donc simplement de points que nous pouvons rajouter pour contrôler le mélange et sculpter le volume que représente une cube-map réelle. Ces cube-maps secondaires ont le même système de « range » et de « falloff » et pour connaître le facteur d'une cube-map réelle, il suffit de prendre le maximum entre son facteur et ceux des cube-maps secondaires associées. L'attribution du volume d'un échantillon lumineux se fait ainsi par l'addition de sphères pouvant suivre les formes du décor tout en procurant un moyen simple d'obtenir un paramètre d'atténuation, grâce aux deux distances, « range » et « fallof ».

6.2.e Calcul du poids des cube-maps

Nous allons voir quelques algorithmes classiques permettant de calculer les poids relatifs des différentes cube-maps qui ont été sélectionnées.

Voici une représentation des poids assignés à trois échantillons (ou cube-maps) différents par chacun des algorithmes présentés par la suite. Le poids de chaque échantillon est représenté par la valeur d'une des couleurs primaires (rouge, vert et bleu). La somme est normalisée, c'est-à-dire que l'addition des trois valeurs est toujours égale à un. Ainsi, quand la couleur est primaire pure, seule une des cube-maps sera visible, alors que quand la couleur est « sale », c'est que plusieurs cube-maps sont utilisées.

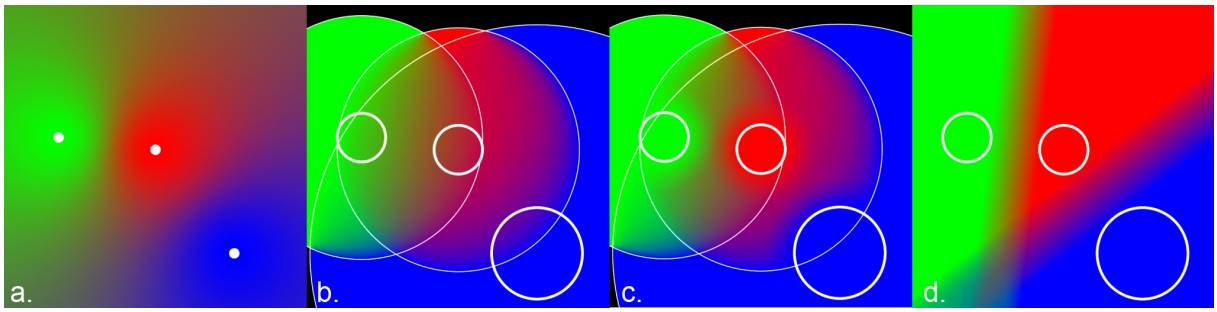


Figure III.36: Algorithmes de mélanges appliqués à trois échantillons.
a. Inverse de la distance au carré
b. 1 - Distance normalisée entre une distance minimum et une distance maximum
c. (1 - Distance normalisée) multiplié par inverse de la distance
d. Séparation par la triangulation de Delaunay

Inverse de la distance au carré :

L'idée la plus simple pour calculer les poids est de sélectionner les « n » cube-maps les plus proches, puis de calculer leur facteur de mélange en utilisant l'inverse de la distance au carré entre la camera et le centre de la cube-map. Pour prendre en compte le « range », il est possible de soustraire cette valeur à la distance, en veillant à rester dans les nombres positifs ou nuls, de monter le tout au carré et d'en prendre l'inverse.

L'équation simple est donc :

$$facteur1 = 1 / distance^2$$

Et l'équation avec prise en compte du « range » est :

$$facteur2 = 1 / max(distance - range , 0.001)^2$$

Les valeurs des différentes cube-maps sont ensuite normalisées afin que la somme de tous les facteurs pris en compte soit de 1. Le calcul est très simple, chaque poids est divisé par la somme totale des poids. Cette normalisation a deux avantages. Tout d'abord, elle permet d'éviter que la somme des facteurs soit supérieure ou inférieure à 1 et donc que la réflexion apparaisse plus claire ou plus foncée par endroits. Elle permet également d'assurer que si la caméra est dans la zone de « falloff » d'une seule des cube-maps, celle-ci sera utilisée pleinement au lieu de devenir de plus en plus sombre en fonction de la distance. En effet, nous cherchons à calculer les transitions entre les échantillons, mais les zones sans échantillons ne

nous intéressent pas. Nous considérons que la caméra est toujours dans la zone d'au moins un échantillon.

Cette technique, inspirée des calculs de la gravité newtonienne, est très simple et a prouvé son efficacité dans le mélange d'échantillons lumineux. Malheureusement, dans le cas du mélange de cube-maps, ce calcul n'est pas très adapté pour deux raisons. La première provient du fait que l'inverse de la distance au carré ne devient jamais nul. Chaque cube-map aura donc toujours une influence, même minime, ce qui va causer des heurts à chaque fois que la caméra sort de la zone de « falloff » d'une cube-map. Au-delà de cette zone, la cube-map ne sera plus utilisée, alors que son influence était faible, mais pas nulle. Dans le cadre de cette technique, la valeur de « falloff » n'est utilisée que pour limiter le nombre de cube-maps à envisager en un emplacement précis. La seconde raison est visible lorsque la caméra s'éloigne de toutes les cube-maps. Elle tombe alors très vite dans une zone « grise » où plusieurs cube-maps ont plus ou moins la même intensité et aucune ne prend vraiment le pas sur les autres. Dans le cas du mélange d'une valeur simple, tel qu'une couleur ou une intensité lumineuse, cela ne pose pas forcément de problème. La couleur risque néanmoins d'être « sale » comme le serait le résultat d'un mélange de toutes les couleurs, c'est-à-dire un gris « sale ». Dans le cas du mélange d'images, ici de cube-maps, il est préférable d'avoir le minimum de période de transition et d'avoir le moins d'images différentes à mélanger. En effet, le mélange de deux images ne produit pas vraiment un résultat compréhensible. Les deux images se superposent et donnent un aspect fantomatique aux détails qui deviennent à moitié transparents. Cette technique n'est donc pas idéale dans le cas du mélange d'images.

Distance au « range » divisée par le « falloff » :

Une seconde possibilité pour calculer le facteur de mélange d'une cube-map est de calculer la distance au centre, à laquelle nous retirons la valeur de « range » et que nous divisons par le « falloff ». Les facteurs sont une nouvelle fois normalisés pour s'assurer que leur somme est unitaire. L'avantage de cette technique est de prévenir les heurts en sortie de la zone de « falloff ». Elle garantit que le facteur d'une cube-map sera à zéro lorsque la caméra s'apprête à sortir de la zone de « falloff ». La transition d'une cube-map à la suivante est définie par la zone de recouvrement des deux cercles de « falloff ».

facteur3 = 1 – saturate((distance – range) / (falloff – range))

où saturate sert à limiter la valeur entre 0 et 1 (équivalent de : max(0, min(1, x)))

Le calcul peut être simplifié en :
facteur3 = saturate((falloff – distance) / (falloff – range))

Cette méthode ne garantit pas que le facteur, une fois normalisé avec les autres cube-maps, sera à 1 lorsque la caméra arrive près de la zone de « range » d'une cube-map. Au contraire, dans le cas où la zone de « range » est recouverte par la zone de « falloff » d'une autre cube-map, cette zone de « range » sera « grise », c'est-à-dire un mélange de plusieurs cube-maps. Pourtant cette zone de « range » est censée être le mieux représentée par la cube-map associée, qui a été calculée pratiquement à cet endroit exact.

Pour remédier à ce problème, nous pouvons mélanger cet algorithme avec le premier présenté, c'est-à-dire ajouter un facteur de l'inverse de la distance (au carré ou non) afin de forcer l'influence d'une cube-map à devenir infinie à proximité de sa zone de « range ». Une autre possibilité est tout simplement d'appliquer cet algorithme deux fois, avec un « falloff » plus petit la seconde fois, mais une influence plus importante.

$$facteur4 = \frac{facteur3}{saturate\left(\left(distance - range\right) \times 10\right)}$$

la valeur 10 diminue l'aire d'effet de l'inverse de la distance

Utilisation de la triangulation de Delaunay

Les techniques que nous avons présentées jusqu'à maintenant présentent un inconvénient majeur. Chaque point est pris en compte individuellement et les résultats sont ensuite mélangés entre eux avec une simple normalisation. L'utilisation de la triangulation de Delaunay va permettre de créer une structure à partir des points afin d'obtenir les zones de transition les plus logiques possible entre les points les plus adéquats.

> « La triangulation de Delaunay d'un ensemble P de points du plan est une triangulation telle qu'aucun point n'est à l'intérieur du cercle circonscrit d'un des triangles. » Wikipedia

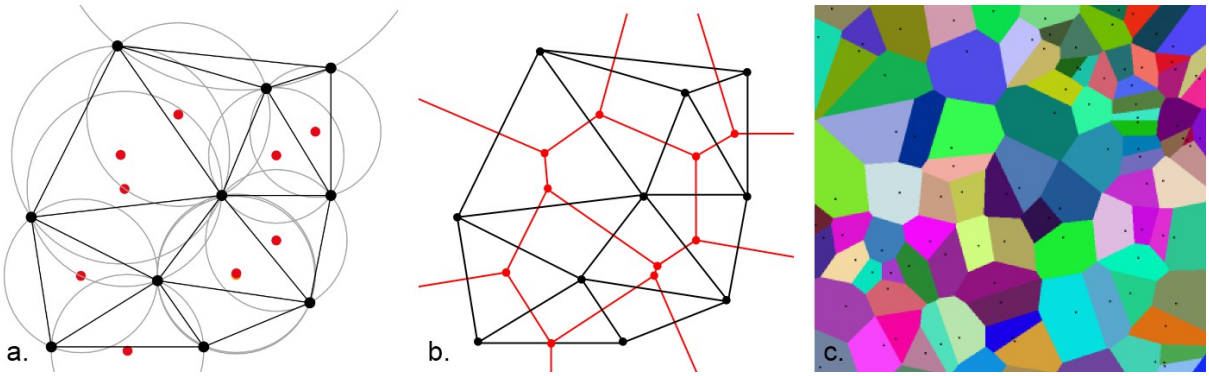


Figure III.37: La triangulation de Delaunay.
a. Triangulation de Delaunay des points noirs et les centres des cercles inscrits (en rouge)
b. Diagramme de Voronoï (en rouge) et triangulation de Delaunay (en noir)
c. Diagramme de Voronoï liant chaque zone à une couleur

Images Wikipedia

Le diagramme de Voronoï est une structure de division de l'espace qui utilise une liste de points et leur associe l'espace qui est plus proche d'eux que de n'importe quel autre point. La triangulation de Delaunay est associée au diagramme de Voronoï, car elle consiste à relier les points entre eux s'ils sont voisins dans le diagramme de Voronoï.

Pour bien illustrer notre propos, nous pouvons faire une comparaison avec la technique d'éclairage 3D dénommée « ombrage de Gouraud ». Cette technique permet, à partir d'un maillage de triangles, d'interpoler les valeurs des sommets (ici une valeur de luminosité) sur la surface des triangles. Ainsi, la valeur d'un point donné de la surface va dépendre uniquement des trois points qui forment le triangle qui le supporte. Les valeurs sont interpolées sur la surface par le calcul des coordonnées barycentriques. Nous souhaitons obtenir un résultat similaire, chaque sommet étant une cube-map et les coordonnées barycentriques nous fournissant les poids. La triangulation de Delaunay va intervenir dans le choix des sommets à relier pour former chaque triangle. Cette technique de triangulation permet en effet d'équilibrer au mieux le maillage formé, en assurant notamment une bonne répartition des angles des triangles, qui seront les moins allongés possible.

La triangulation de Delaunay fonctionne sur une surface en utilisant trois points, mais elle fonctionne également en trois dimensions en cherchant à calculer des tétraèdres, c'est-à-dire quatre points qui vérifient les mêmes propriétés que ceux de la triangulation de Delaunay, à

savoir que chaque sommet d'un tétraèdre n'a pas d'autre sommet plus proche de lui que ceux du même tétraèdre. L'algorithme en 3D est similaire à celui en 2D, mais va demander plus de calculs et plus de mémoire, car le nombre de connexions en 3D est plus important qu'en 2D.

Les développeurs du moteur « Unity » ont présenté récemment leur algorithme de mélange d'échantillons de lumières précalculés par la construction de tétraèdres[67], le résultat semble idéal, mais requiert au préalable la construction et le stockage des tétraèdres et d'autres informations permettant une recherche rapide. Par contre, ils ne prennent pas en compte de zones de « range » et « falloff ».

La triangulation de Delaunay demande un algorithme assez complexe, qui a notamment besoin de calculer certaines données intermédiaires comme les arêtes des triangles, afin de les modifier au cours de la progression de l'algorithme. S'il s'agissait de connaître les poids d'un grand nombre de points, il serait alors légitime d'utiliser cet algorithme, car une fois la triangulation complète calculée, il suffit de peu de calculs pour chaque point. Néanmoins, s'il s'agit uniquement de connaître la valeur d'un seul point de l'espace, dans notre cas la caméra, Il est alors bien lourd de chercher à trianguler l'ensemble de l'espace. Nous allons voir qu'il est possible de se passer de cette triangulation pour calculer rapidement les valeurs d'un point unique.

Nous avons donc développé un algorithme qui obéit aux mêmes exigences que le diagramme de Voronoï sans pour autant avoir à le construire explicitement. Pour cela, nous avons suivi à la lettre la définition même du diagramme de Voronoï, c'est-à-dire que chaque sommet va influencer les points de l'espace qui sont plus proches de lui que des autres sommets. Pour calculer l'influence d'un sommet sur un point, notre algorithme va calculer le dégradé du poids entre ce sommet et chacun des autres sommets proche, en fonction de la distance et en respectant les valeurs de « falloff » et de « range ». L'étape suivante est de soustraire toutes ces valeurs à 1 ce qui donne le poids final du sommet. Une fois cette opération effectuée pour tous les sommets, une simple normalisation des poids assurera un comportement cohérent en toutes circonstances. En effet, cet algorithme n'est pas parfait, à cause notamment du « range ». L'algorithme ne s'occupe pas vraiment de la situation dans laquelle plusieurs zones de « range » se recouvrent, mais uniquement du recouvrement des zones de « falloff ». Le croisement de plusieurs zones de dégradés n'est pas non plus parfait, et il est nécessaire de renormaliser pour que le résultat soit utilisable.

67 Robert Cupisz, « Light probe interpolation using tetrahedral tessellations », *Conférence GDC 2012* (2012).

Un avantage secondaire de l'utilisation du Diagramme de Voronoy est de pouvoir prendre en compte aisément les collisions du décor afin d'éviter que les cube-maps ne puissent influencer les objets à travers les murs. Pour cela, des tests de collisions peuvent être précalculés entre chacune des cube-maps. Alternativement, ajouter des sommets près des obstacles suffira à éviter l'interpolation d'un côté sur l'autre.

Pour conclure, la solution que nous avons utilisé est celle de la distance au « range » divisé par le « falloff ». En effet, nous n'avons effectué les recherches sur la triangulation de Delaunay qu'une fois le projet de DONTNOD déjà très avancé. Nous avons donc conservé la technique que nous utilisions depuis le début, afin de préserver les réglages déjà effectués et les procédures de travail déjà intégrées par les artistes des lumières. La technique par triangulation de Delaunay est également moins performante et nécessite un calcul à la complexité $O(n^2)$, car le poids de chaque sommet est calculé grâce à la prise en compte de chacun des autres sommets.

6.3 La simulation de la parallaxe

L'utilisation de cube-maps permet de calculer facilement une ambiance lumineuse locale. Cette ambiance fonctionnera très bien pour des petits objets proches du point de capture de la cube-map et qui ne présentent pas de surfaces planes trop importantes. Au contraire, les murs et les sols sont de grands objets constitués de grandes surfaces planes. Dans ces cas-là, notre

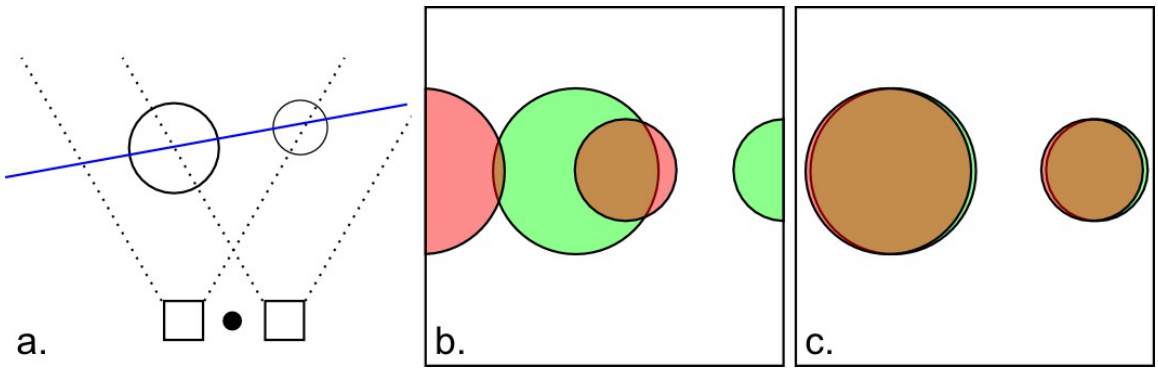


Figure III.38: Le problème du mélange en situation de parallaxe.
a. Scène 3D vue du dessus, constituée de deux sphères et de deux cube-maps. Les lignes pointillées désignent les vues d'une face de chaque cube-map.
b. Superposition directe des images des cube-maps de gauche (vert) et de droite (rouge). Les sphères ne sont pas du tout alignées, le mélange est peu crédible.
c. Superposition avec prise en compte de la parallaxe. Les deux sphères sont pratiquement superposées et le mélange s'opère aisément. Le plan de parallaxe est représenté par une ligne bleue et la position de la caméra par un point noir sur la vue a.

technique de la cube-map unique ne fonctionne pas très bien, car elle ne prend absolument pas en compte la parallaxe.

Il s'agit d'adapter la cube-map en fonction de la position du pixel à calculer afin d'obtenir une parallaxe et ainsi pouvoir afficher un espace assez grand sans artefacts trop visibles. De plus, la prise en compte de la parallaxe permet d'améliorer les transitions et le mélange entre deux cube-maps différentes. Habituellement, deux textures cubiques voisines présentent des ambiances assez similaires, mais des différences liées justement à la parallaxe. Avec une technique qui reproduit une parallaxe, la transition est alors plus douce, car les éléments présents dans les deux cube-maps sont replacés au même endroit par la parallaxe et ainsi mélangés ensemble. Nous allons voir quelques techniques que nous avons envisagées pour redonner une parallaxe à une cube-map.

6.3.a Raytracing de cube-map au pixel

Une cube-map contient des informations sur l'environnement à 360 degrés. Cependant, nous n'avons pas de notion de distance concernant l'environnement, qui est considéré comme infiniment lointain. Afin de redonner une parallaxe, nous allons devoir redonner également une distance à chaque pixel de l'environnement, et utiliser cette distance pour calculer son aspect vu depuis un point quelconque de l'espace.

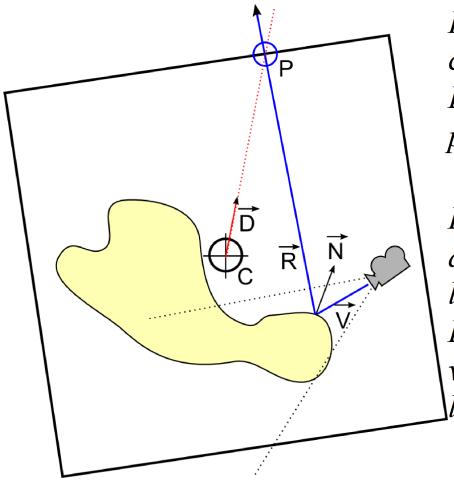


Figure III.39: Intersection entre le rayon de réflexion et le cube de projection de la cube-map.
Le vecteur $\vec{R}$ est la réflexion du vecteur $\vec{V}$ en provenance de la caméra par la normale de la surface $\vec{N}$.
Le point P est l'intersection du vecteur $\vec{R}$ avec le cube de projection. Le centre C du cube est également l'emplacement depuis lequel la cube-map a été calculée.
Le vecteur $\vec{D}$ est la direction entre les points C et P. Ce vecteur sera notre nouvelle direction d'échantillonnage de la cube-map.

Cette distance peut être approximée par une primitive, telle qu'un cube, ce qui facilite énormément les calculs. La forme cubique n'est pas forcément la meilleure, mais fonctionne bien dans beaucoup de cas et son extrême simplicité en fait un choix privilégié. La sphère est également intéressante, mais représente très mal les murs. Le cube peut éventuellement subir diverses transformations afin d'en faire un prisme ou une pyramide. L'avantage du cube est d'être une forme simple dont nous pouvons calculer analytiquement l'intersection des faces

avec un rayon. La technique de projection sur un cube a été présentée sur le forum du site GameDev.net par l'utilisateur Bartosz Czuba[68]. Une technique similaire avec projection sur une sphère existe également[69].

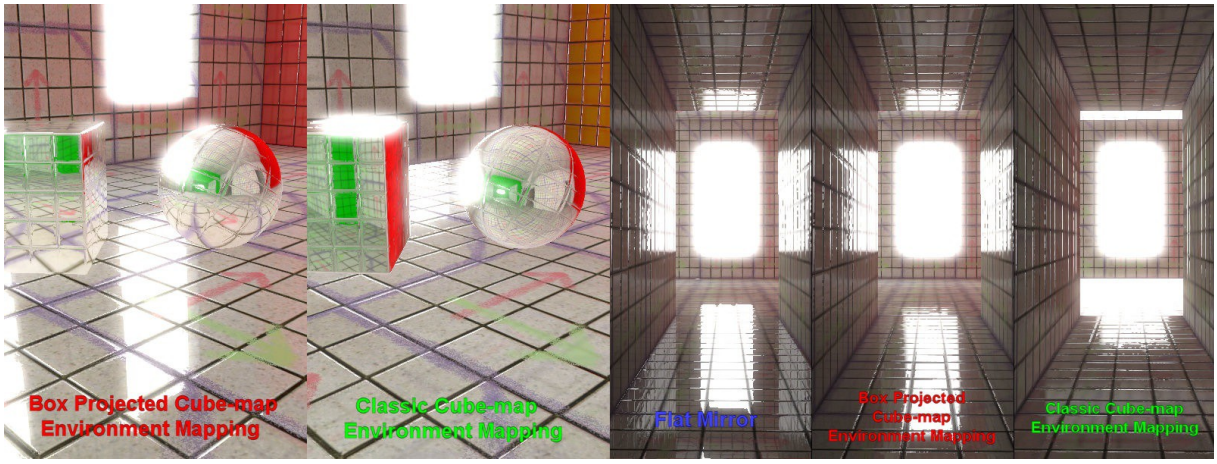


Figure III.40: Images du box projected cube-map par l'utilisateur Behc.

Le test de collision entre un rayon et les six plans d'un cube est pratiquement aussi rapide qu'avec un seul plan. La carte graphique peut effectuer des calculs dédiés sur des vecteurs, qui sont aussi rapides que s'ils étaient effectués sur une seule valeur. Les plans opposés du cube peuvent également être traités ensemble.

Figure III.41: Test d'implémentation de la collision avec une boite au pixel dans le moteur UDK,

68 Bartosz (Behc) Czuba, « Box Projected Cubemap Environment Mapping », 2010, http://www.gamedev.net/topic/568829-box-projected-cubemap-environment-mapping/.

69 Kevin Bjorke, *GPU Gems -Chaptitre 19 : Image-Based Lighting* (Addison-Wesley Professional, 2004).

Voici une fonction HLSL qui effectue l'intersection d'un rayon avec une boite et renvoie un nouveau rayon entre la position de la cube-map et le point d'intersection :

```
float3 IntersectBox(float3 RayDirection)
{
   // Le rayon est en espace-monde au dépard
   float3 RayWS = normalize(RayDirection);

   // Nous transférons le rayon dans l'espace-local de la boite
   // Le rayon sera déformé si la boite a une échelle non-uniforme
   float3 RayLS = mul((float3x3)BoxWorldToLocal, RayWS);

   // Le vecteur entre le centre de la boite et le pixel
   // est également transféré dans le repère de la boite
   float3 BoxLocWS = PixelPosWS – BoxCenterPosWS ;
   float3 BoxLocLS = mul((float3x3)BoxWorldToLocal, BoxLocWS);

   // Une fois dans le répère de la boite, nous effectuons
   // une collision avec une boite de 1x1x1
   float3 Unitary = float3(1.0f, 1.0f, 1.0f);

   // Nous calculons les collision avec les deux faces possibles
   // Pour chacun des trois axes de la boite
   float3 FirstIntersect  = (Unitary - BoxLocLS) / RayLS;
   float3 SecondIntersect = (-Unitary - BoxLocLS) / RayLS;

   // Nous sélectionnons la collision la plus lointaine, car
   // nous cherchons les collisions avec les faces arrières
   float3 MinPlane = max(FirstIntersect, SecondIntersect);

   // Parmi les trois plans, nous sélectionnons le plus près
   float Distance = min(MinPlane.x, min(MinPlane.y, MinPlane.z));

   // Le facteur issu de la collision est utilisé pour
   // multiplier le rayon original
   float3 CollisionWS = RayWS * Distance + PixelPosWS;

   // Nous utilisons la direction entre la position de capture de la
   // cube-map et le point d'intersection pour indexer la cube-map
   return texCUBE(envMap, CollisionWS - CubeMapCenterPosWS );
}
```

La technique que nous présentons ici ressemble à celle utilisée par Epic que nous avons présentée précédemment, à la différence qu'ils utilisent une liste de quadrilatères, associée à une liste de textures. Nous allons utiliser un seul cube et sa texture. Leur algorithme permet d'aller plus loin en approximant n'importe quel décor à base de plans carrés. Cependant, les calculs sont plus lourds, car ils prennent en compte un grand nombre de plans (~100 chez Epic), mais aussi parce qu'ils doivent trier les plans entre eux. Le cube est au contraire un

volume convexe, donc il n'est pas nécessaire de trier les plans, mais simplement de n'afficher que ceux dont la normale pointe vers la caméra.

Figure III.42: Comparaison sans (gauche) et avec (droite) la correction de la parallaxe. L'image avec correction présente une réflexion crédible sur la majorité de la scène.

Ainsi, l'image de l'environnement va être projetée sur un cube de taille, orientation et position réglables. Nous avons besoin de pouvoir calculer l'image de ce cube depuis n'importe où. Les calculs d'intersection entre un cube et un rayon sont suffisamment simples pour être effectués sur la carte graphique. Nous lui procurons la matrice de transformation du cube, et elle calculera l'intersection entre la direction de réflexion du pixel et chacun des plans du cube. Chaque objet peut utiliser la cube-map qu'il souhaite, avec son propre cube de projection.

Figure III.43: Raytracing au pixel dans « Remember Me »

L'avantage immense de cette technique est de ne pas se limiter aux sols, mais de fonctionner sur toutes les surfaces. Ainsi, tous les murs deviennent capables de refléter l'environnement d'une manière convaincante. Bien évidemment, il faut que l'environnement suive globalement la forme d'un cube (ou d'un prisme). Dans les cas où la forme n'est pas un cube, il est en général possible de faire correspondre au moins un ou deux murs avec les bords du cube. Les autres bords vont être placés très loin, afin d'éviter d'avoir une parallaxe trop visible sur ces

côtés. Les bords du cube peuvent être placés grossièrement dans la plupart des scènes, typiquement aux endroits les plus importants, ou un peu plus loin à défaut.

Nous devons éviter que la caméra traverse les bords du cube de projection, car l'effet n'est pas esthétique. Une fois la caméra à l'extérieur du cube, une grande partie de la vue ne pourra pas trouver de collision avec le volume, et sera donc noire. Le cube de projection de la cube-map est donc en général placé autour des positions que le joueur peut atteindre et les bords problématiques sont masqués par une transition vers une autre cube-map.

6.3.b Cube-map avec information de profondeur

La cube-map peut stocker la distance de chacun des points en plus des valeurs de luminosité de l'environnement. La surface de projection de la texture est alors décrite implicitement par ces distances. Comme pour la projection sur le cube, nous pouvons chercher le point d'intersection entre le rayon de réflexion et la texture de profondeur de la cube-map. Il s'agit de résoudre un problème d'intersection entre un rayon et une texture d'élévation. Des techniques ont été traitées de nombreuses fois dans le cadre du « parallax mapping » qui, justement, cherchent à redonner une parallaxe à une texture appliquée sur un plan en fonction d'une texture de profondeur.

Figure III.44: Test de parallaxe avec texture de profondeur sous le moteur UDK

Les mêmes techniques sont donc applicables à notre problème. Cependant, il est toujours nécessaire de faire appel plusieurs fois à la texture de profondeur afin de tester l'élévation en différents points jusqu'à trouver une intersection. Une fois la position d'intersection trouvée, il suffit d'utiliser l'échantillon présent à cet endroit dans la cube-map pour obtenir une parallaxe

qui prend en compte la forme de l'environnement. Le résultat est intéressant, mais ressemble un peu à un drap qui serait projeté depuis le centre de la cube-map. Le décor dans la réflexion semble alors être mou et informe et les bords extérieurs des objets sont étirés, copiant la valeur de couleur de l'environnement sur tout le long du bord. Il est possible de rendre transparent ces étirements pour n'afficher que les éléments connus de l'environnement et cacher les artefacts de la méthode de projection.

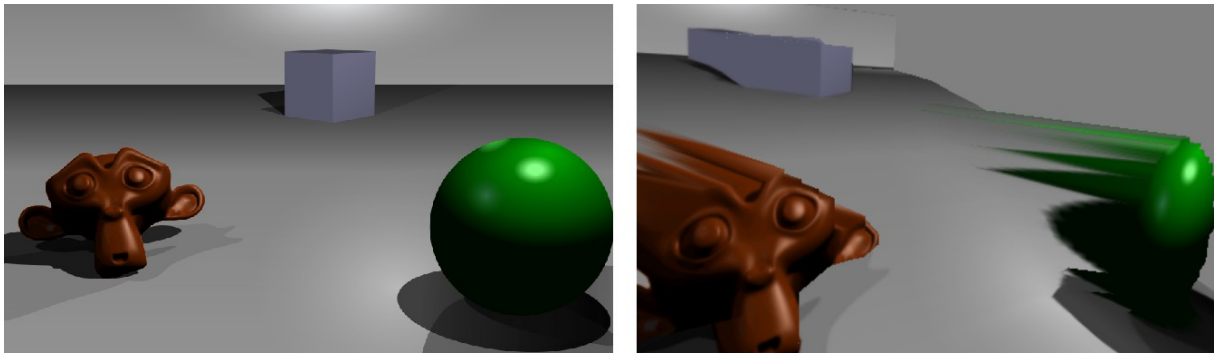


Figure III.45: Exemple d'étirement (à droite) d'une image (à gauche) en fonction d'une texture de profondeur. La couleur est étirée le long des discontinuités de la profondeur.

Le coût de plusieurs appels à la texture rend l'algorithme lourd. Ce n'est plus une solution économique pour approximer la réflexion. Le stockage de la texture de profondeur augmente également le coût mémoire de la cube-map. Nous avons découvert récemment que cette technique existait auparavant,[70] car elle vient d'être adaptée au cas du mélange de cube-maps préfiltrées[71], qui est précisément le problème qui nous occupe.

6.3.c Pseudoparallaxe

Afin de minimiser à l'extrême l'impact sur les performances du calcul de la parallaxe, il existe une technique qui propose d'approximer la parallaxe empiriquement en fonction de la taille des objets. En effet, le phénomène de parallaxe dépend essentiellement de l'emplacement de l'objet par rapport au centre de la cube-map de réflexion, ainsi que de la taille moyenne de l'environnement local. Le principe est donc de décaler la direction de réflexion en fonction de la position 3D du pixel à calculer par rapport au centre de la cube-map. Ainsi, la réflexion se décale pour utiliser majoritairement la partie de la sphère d'ambiance qui est dans la même direction que le pixel par rapport à la cube-map. Le résultat n'est pas parfait et ne suffit pas à

70 L. Szirmay-Kalos et al., « Approximate Ray-Tracing on the GPU with Distance Impostors », in *Computer Graphics Forum*, vol. 24, 2005, 695–704.

71 Daniel Scherzer et al., « Pre-convolved Radiance Caching », in *Computer Graphics Forum*, vol. 31, 2012.

régler le problème des plans comme le sol, mais ajoute des variations selon la position de l'objet pour un coût minimum.

$$D = normalize(k * (Pp - C) + R)$$

avec :
- *D = vecteur d'échantillonnage de la cube-map*
- *k = paramètre réglable empiriquement*
- *Pp = position du pixel calculé*
- *C = position du centre de la cube-map*
- *R = vecteur de réflexion*

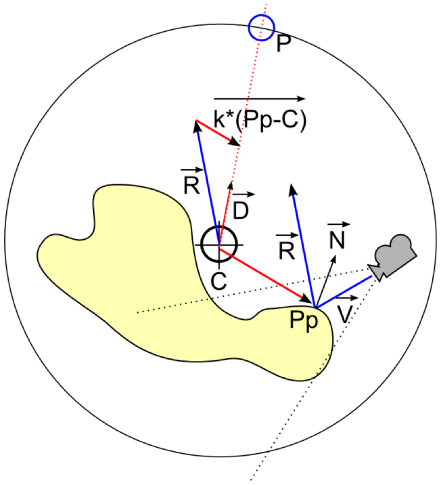


Figure III.46: Pseudoparallaxe

Le coût est comme on le voit très réduit. Le « normalize » présent dans la formule et qui sert à normaliser le vecteur final est en fait inutile et pourra être évité, car l'échantillonnage de la cube-map effectue déjà sa propre normalisation. Le coût final est donc seulement d'une multiplication et de deux additions, ce qui est très raisonnable. La carte graphique sait très bien optimiser ce type de calcul. Différentes implémentations existent, notamment celles de nVidia[72] et de TriACE[73].

6.3.d Parallaxe intégrée à la cube-map unique

Le calcul de la parallaxe de l'environnement projeté sur un cube, nous l'avons vu, est tout-à-fait calculable par pixel. Cela nous permet de prendre en compte la réflexion du cube d'environnement sur n'importe quelle surface. Cependant, cette technique rajoute un coût de calcul par pixel. Ce coût est assez faible en soi, mais devient prohibitif lorsqu'il est combiné avec tous les autres calculs à accomplir (calcul des lumières directes, combinaisons de textures…)

Parallèlement, nous avons vu dans le cadre du mélange de cube-maps que nous pouvions éviter le coût d'un mélange par pixel en calculant une seule cube-map pour tous les objets. Lors de sa création, la position de la caméra est utilisée pour calculer le pourcentage d'utilisation de chaque cube-map. Il est donc très intéressant de chercher à appliquer le calcul de parallaxe au moment du calcul de cette cube-map intermédiaire unique. Le coût au pixel

72 NVIDIA, « The Naked Truth Behind NVIDIA's Demos » (2005).

73 Yoshiharu Gotanda et Tatsuya Shoji, « Real-time Physically Based Rendering - Implementation », *TriACE* (CEDEC 2011).

est alors simplement celui de l'échantillonnage d'une cube-map. L'algorithme de parallaxe est effectué une seule fois par image, au moment du calcul de la cube-map unique. Typiquement, une cube-map fait 128×128 pixels multipliés par 6 faces, plus chacun des niveaux de mipmap ($64 \times 64 \times 6 + 32 \times 32 \times 6$ …), soit au total environ 131.000 pixels, et donc 131.000 calculs individuels de parallaxe. L'écran comporte en général $720 \times 1280 = 921.600$ pixels, soit environ sept fois plus. En réalité, nous n'effectuons ces calculs sur l'écran que pour la surface de réflexion visible, typiquement le sol, qui représente seulement un certain pourcentage du sol (par exemple 1/3 de l'écran). Nous obtenons tout de même une réduction globale du coût dans pratiquement tous les cas, si nous calculons la parallaxe dans la cube-map plutôt que sur l'écran.

Nous pouvons de plus utiliser le « rasterizer » de la carte graphique pour accélérer énormément le calcul de projection sur le cube. Dans l'algorithme de réflexion, le vecteur de collision avec le cube peut potentiellement changer à chaque pixel, en fonction de la normale de la surface, et il est donc exclu d'utiliser les techniques classiques d'affichage de polygones. Ces techniques ne fonctionnent qu'avec une direction de collision qui change régulièrement, comme dans le cas des rayons en provenance d'une caméra. Nous sommes donc obligés d'utiliser un lancé de rayon analytique pour connaître la collision du rayon avec le cube. Au contraire, lors de la création d'une cube-map, le vecteur de collision coïncide avec le vecteur de la caméra de capture. Ainsi, l'affichage de la cube-map avec une parallaxe revient simplement à afficher les plans d'un cube en utilisant directement les techniques habituelles de la 3D temps réel : l'affichage de polygones par « rastérisation ». Les textures de chaque cube-map à mélanger seront plaquées sur la forme du cube en utilisant le vecteur entre la position de la cube-map et la position du pixel du cube. Avec cette technique, le calcul est beaucoup plus rapide, et la complexité de la scène n'influence pas le coût du calcul de la parallaxe.

La forme de projection est également libérée. Nous ne sommes plus obligés d'utiliser un cube, mais nous pouvons prendre n'importe quelle forme, tant que celle-ci reste convexe. L'augmentation du nombre de faces du volume n'est pas vraiment un problème, car la carte graphique est très efficace pour l'affichage de polygones. La forme doit être le plus possible convexe afin de ne pas être obligé de trier les faces ou d'utiliser un Z buffer pour éviter les superpositions des polygones. Le mélange de plusieurs cube-maps est simple également, il suffit d'ajouter chaque cube-map avec son volume, en utilisant un degré de transparence qui correspond au poids de la cube-map. Chaque volume peut avoir sa propre parallaxe.

Cependant, la méthode par raytracing du pixel et celle par parallaxe de la cube-map unique ne sont pas équivalentes. Il est exclu de pouvoir calculer exactement les valeurs finales dans la cube-map puisque nous ne disposons pas du couple {position, normale} du pixel final. La cube-map intermédiaire cherche uniquement à compenser la différence de position de la caméra par rapport aux positions des cube-maps. Elle ne cherche pas à compenser la position du pixel final. La position de la caméra est la même pour tous les pixels et nous la connaissons au moment du calcul de la cube-map intermédiaire. La nouvelle cube-map va donc représenter la sphère d'ambiance lumineuse qui arrive à la position de la caméra, en fonction des différentes cube-maps voisines. Cette technique donne un sentiment de parallaxe, mais ne fonctionne pas avec toutes les surfaces. Par exemple, un sol présentera une parallaxe qui permet à la réflexion d'être en adéquation horizontale avec le reflet, mais pas verticale. Ainsi, les mouvements sur le plan du sol vont montrer une parallaxe convaincante, mais un saut du joueur révélera que la réflexion bouge avec la caméra au lieu de bouger à l'inverse par rapport à l'axe de réflexion. Concrètement, la parallaxe est correcte sur le plan orthogonal à la normale de la surface, mais est inversée le long de cette normale.

Cette parallaxe selon la normale peut être simulée pour un plan de réflexion bien précis, tel que le sol. Nous utilisons la position inversée par rapport au plan de réflexion de la caméra à la place d'utiliser directement la position de la caméra dans le calcul de la parallaxe de la cube-map. L'effet fonctionne parfaitement à la hauteur du plan, et les autres objets ont toujours une réflexion plausible, bien que fausse.

Voici un schéma présentant les différences entre les techniques de mélange des cube-maps :

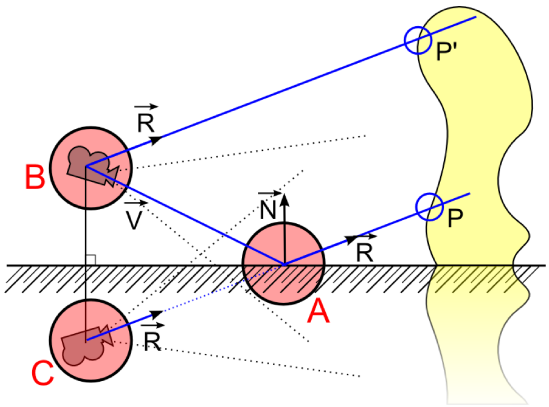


Figure III.47 Parallaxe intégrée à une cube-map unique.

La cube-map A est idéale, elle est centrée sur le pixel de la surface réflexive. L'échantillonnage de la cube-map A par le vecteur R nous permet de connaître la couleur du point P.

La cube-map B est calculée à la position de la caméra à partir de la parallaxe des cube-maps de la scène. L'utilisation du vecteur R avec la cube-map B nous donne cette fois la couleur d'un autre point P' qui est aligné avec P, mais avec un décalage le long de la normale N.

La cube-map C est placée sur la position miroir de la caméra par rapport à la normale N du plan de réflexion. L'échantillonnage de C avec le vecteur R permet de trouver le même point P qu'avec la cube-map A.

La méthode par « rastérisation » utilisera le code suivant dans le pixel shader lors de l'affichage de la boite :

```
// La cube-map est plaquée sur la forme à partir de son centre
float3 RefVector = PixelPos.xyz - CubeMapCenter.xyz ;
float3 RefColor = texCUBE(CubeTexture, RefVector);
```

6.3.e Utilisation de proxies avec une réflexion plane

Au cours de la recherche, nous sommes passés par plusieurs algorithmes de simplification jusqu'à parvenir au calcul d'une cube-map dédié à la réflexion d'un plan particulier (par exemple le sol) tout en étant acceptable pour les autres objets. Nous pouvons alors remarquer que nous avons pratiquement fait un tour complet des techniques de réflexion pour finalement revenir à la réflexion selon un plan unique, qui était notre point de départ. Or il est tout à fait possible de calculer cette réflexion sans utiliser de cube-maps. Nous pouvons utiliser la technique que nous avons déjà présentée, celle du rendu de la scène dans une texture, avec une caméra en miroir.

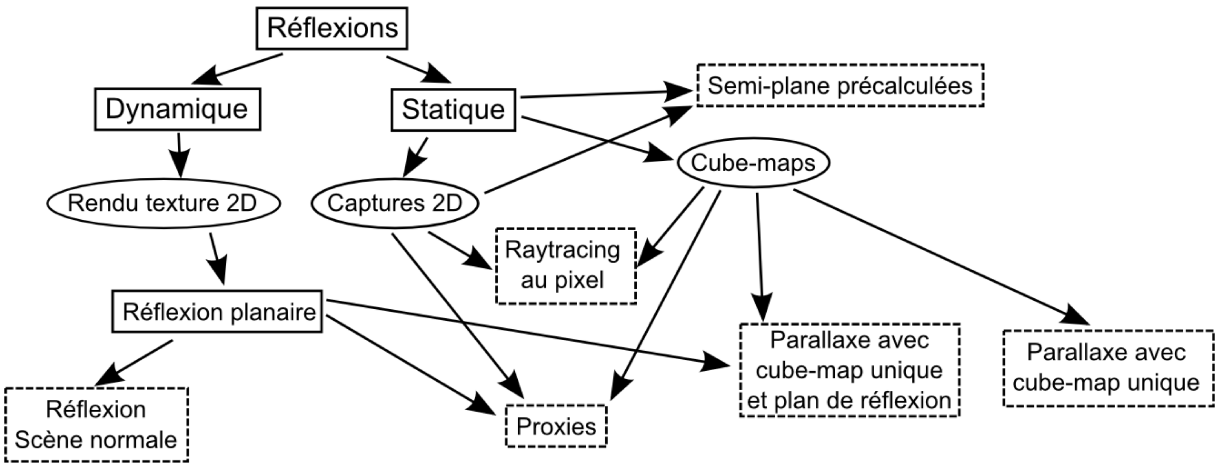


Figure III.48: Schéma résumant les différentes familles de techniques de réflexion que nous avons présentées.
Les techniques sont entourées en pointillés et les formats de stockages de ronds.

Cependant, nous avons tout de même mis le doigt sur quelque chose d'intéressant lors de nos recherches, l'utilisation des proxies. En effet, la cube-map projetée sur un cube est en fait un proxy très simple qui représente grossièrement la forme de l'environnement et utilise une texture afin d'ajouter du détail. La création de ce proxy est simplifiée puisque nous avons

simplement besoin de positionner la cube-map puis de l'orienter et de donner la taille du cube de projection. La capture de la texture du proxy se fait très simplement, à travers le calcul de la cube-map.

Le calcul de la réflexion sera donc au final séparé selon le type d'objet. Les objets du sol vont utiliser notre réflexion plane à base de proxies, et les autres objets utiliseront la cube-map mélangée unique, sans parallaxe.

Nous utilisons une réflexion planaire à base de proxies, nous ne sommes donc pas limités aux seuls cubes d'environnement. En effet, nous utilisons également des plans en tant que proxy. Ces plans effectuent une capture préalable de la scène sous forme d'image 2D et affichent cette texture dans la réflexion.

Les plans qui serviront de proxy de réflexion peuvent être de trois types. Nous avons les simples plans qui sont placés dans l'espace. Ces plans sont idéaux pour modéliser des murs par exemple. Nous avons également des plans qui feront toujours face à la caméra, ce que nous appelons des « billboards ». Ils sont très utiles pour représenter les formes rondes comme les sphères, ou pour modéliser des lumières ponctuelles par un carré avec une texture de halo rond qui s'oriente vers la caméra. Le dernier type de plan que nous avons utilisé se tourne également vers la caméra, mais en conservant l'alignement avec un vecteur donné. Concrètement, le plan est capable de tourner autour d'un axe unique dans le but de s'orienter vers la caméra. Ce type de plan est très pratique pour modéliser les cylindres et donc par extension les membres du corps humain. Ainsi, un personnage peut être représenté simplement par un petit nombre de plans de ce type, chacun placé sur l'axe d'un des os principaux et se tournant vers la caméra. Nous utilisons cette technique pour afficher un reflet sombre approximatif des personnages du jeu.

Figure III.49: Utilisation de proxies simples alignées sur les axes des os d'un personnage pour représenter sa réflexion.

6.3.f Utilisation du mip-mapping pour éviter le crénelage

L'utilisation exclusive de plans et de volumes convexes pour représenter l'environnement présente un avantage majeur concernant la résolution qui est nécessaire à l'affichage d'une réflexion de qualité.

En effet, nous souhaitons avoir des réflexions avec une faible résolution (256x256) pour accélérer le rendu, mais nous voulons éviter le phénomène de crénelage. Avoir une réflexion à faible fréquence avec un joli flou est préférable à une réflexion nette, mais avec des pixels à la forme carrée trop visible et une stabilité temporelle discutable. La réflexion sera, nous le verrons plus tard, floutée par des diminutions de taille successives. La précision de la réflexion initiale n'est donc pas très importante, mais elle doit avoir une évolution spatiale et

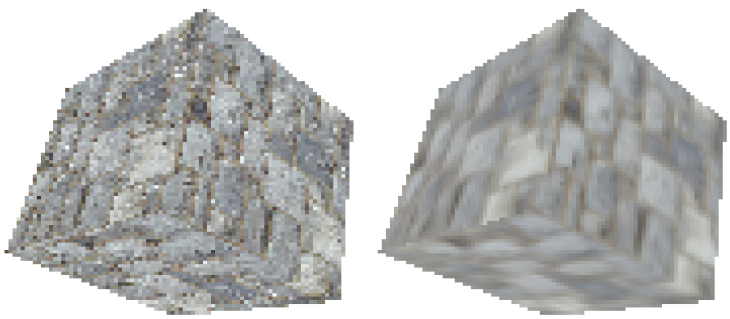


Figure III.50: Cube texturé sans mip-mapping (gauche) et avec (droite).
La texture est floue mais les bords restent crénelés.

temporelle douce. Cette stabilité est habituellement atteignable uniquement avec une grande résolution ou des techniques d'anticrénelage spécifiques.

L'affichage d'une image calculée par la carte graphique passe en général par un « rasterizer » qui ajoute les polygones un à un dans l'image. L'ajout est habituellement binaire, c'est-à-dire qu'un pixel est considéré comme complètement à l'intérieur ou complètement à l'extérieur de la forme. Les bordures du polygone suivent donc une ligne très franche. Les bords internes d'une forme donnée, qui partagent les mêmes informations des deux côtés (positions, coordonnées de textures…), ne présentent pas de problème de crénelage, car le « rasterizer » assure la continuité entre les deux polygones d'un bord en effectuant les mêmes calculs sur les pixels des deux côtés. Les seuls bords qui présentent des problèmes de crénelage sont donc les pixels des bordures externes des polygones.

Les textures sont également des sources de crénelage si elles sont stockées avec une plus grande résolution que la taille du polygone qui sert à les afficher à l'écran. C'est le problème de la « minification » ou le nombre d'échantillons pris dans la texture est faible par rapport à la fréquence du signal échantillonné. Il en résulte un crénelage très visible d'une image à la suivante, car au moindre mouvement de caméra, les échantillons utilisés ne seront plus les mêmes et vont pointer vers des informations différentes. La solution utilisée classiquement est celle du mip-mapping, dont nous avons déjà parlé. Il s'agit de stocker des versions réduites de la texture originale avec des tailles chaque fois divisées par deux. De cette manière, au moment de chercher un échantillon de la texture, la carte graphique calcule le gradient des coordonnées de textures, ce qui nous permet de savoir à quelle vitesse celles-ci varient et donc le rapport entre la texture et la taille du polygone affiché à l'écran. Avec ce ratio, nous pouvons choisir la taille de texture la plus adaptée. Un ratio de 1 utilisera la texture originale alors qu'un ratio de 4 utilisera la version dont la taille est divisée par 4. Le crénelage disparaît effectivement en contrepartie d'un manque de netteté de la texture.

Nous utilisons plusieurs techniques pour éviter d'afficher le moindre bord ou pour les masquer le plus possible. Tout d'abord, la cube-map projetée sur un cube ou un volume convexe ne présente aucun bord extérieur tant que la caméra reste à l'intérieur du volume. Si celle-ci sort, l'extérieur du volume apparaît et donc forme un bord entre l'intérieur et l'extérieur. Il est éventuellement possible d'agrandir le cube dynamiquement lorsque la caméra est proche d'un côté afin d'éviter les trous. En général, ce n'est pas nécessaire, car nous nous arrangeons pour

placer une autre cube-map aux emplacements ou la caméra peut sortir du volume afin d'assurer la transition de manière douce.

La cube-map a donc cet avantage de donner une base, un fond d'ambiance qui recouvre l'ensemble de l'image de réflexion, tant que la caméra reste dans le volume qui sert de proxy pour la parallaxe. L'affichage de la cube-map ne pose donc pas de problème de crénelage même avec une résolution faible.

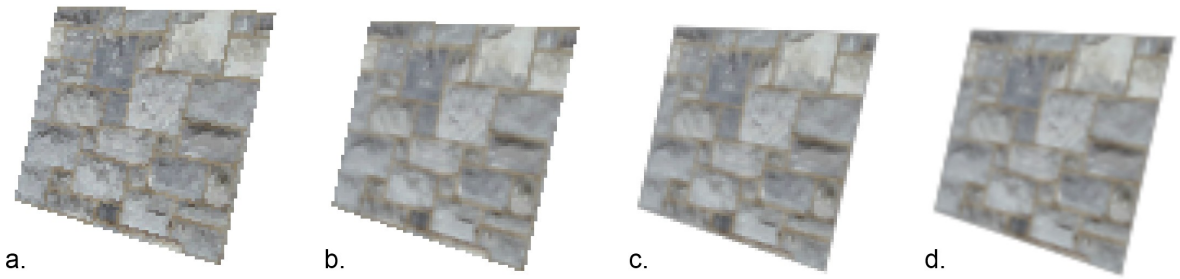


Figure III.51: Utilisation du mip-mapping pour éviter le crénelage sur un plan. Plan texturé affiché sans mip-mapping (a.), avec mip-mapping (b.), avec un rebord transparent de un pixel (c.) puis avec une interpolation bilinéaire finale des pixels (d.).

La dernière source de crénelage restante est constituée des bords des plans que nous ajoutons pour représenter au mieux la scène. Nous pouvons profiter des avantages de la forme plane et de la technique du mip-mapping. Nous allons insérer une bande de 1 pixel transparent sur les quatre bords de la texture de chaque plan, et ce pour chaque niveau de mip-mapping. La technique du mip-mapping va sélectionner un niveau de mip-map qui garantit un ratio de un pixel de la texture pour un pixel du plan. Une fois affiché sur l'écran, le plan aura donc également un rebord de un pixel de coté, quelle que soit sa taille et la résolution de l'écran. Cette légère transparence sur les bords va créer un flou grâce à l'interpolation bilinéaire. Cette interpolation entre le pixel transparent et le pixel opaque voisin va permettre d'obtenir une transition douce et avec peu de crénelages spatial et temporel. La forme étant un carré, il est simple de trouver les bords extérieurs dans la texture, ce sont les quatre bords de l'image. Si nous souhaitons présenter une forme qui ne soit pas carrée, le canal alpha de la texture sera utilisé pour découper la forme dans le carré. Ici aussi, c'est le mip-mapping qui assurera une balance entre le niveau de flou et la faible résolution afin d'obtenir une qualité correcte.

Les rebords légèrement transparents nous forcent cependant à utiliser un mode d'affichage en transparence pour tout l'objet, ce qui signifie que les plans doivent être triés en fonction de leurs profondeurs au lieu d'utiliser un Z-buffer, comme pour les objets opaques. Comme le

nombre de plans est assez réduit en général, cela ne pose pas de problèmes de performance majeurs. Le tri sera effectué sur le processeur principal.

Grâce à ces différentes astuces, nous sommes désormais capables de calculer une image à faible résolution, comme du 256x256, qui ne présente pas de sources importantes de crénelage. La précision d'une telle résolution est suffisante pour simuler la réflexion d'un matériau assez net, même si dans l'idéal il faudrait la même résolution que celle de l'image finale (par exemple du 1280x720, soit environ 14 fois plus de pixels qu'une 256x256).

6.3.g Techniques d'accélération du rendu des plans

Diverses techniques permettent d'accélérer énormément le rendu de l'image de réflexion. Nous souhaitons que le rendu soit le moins cher possible afin de ne pas imposer de contraintes sur les endroits du jeu où nous pouvons utiliser la réflexion. Cela nous permet d'éviter de diminuer la qualité des graphismes du jeu lorsque nous avons besoin de calculer cette image.

Tout d'abord, le coût de l'affichage d'un certain nombre de polygones dépend beaucoup du nombre d'appels à la carte graphique qui seront faits pour les afficher. C'est ce que nous appelons en anglais les « drawcalls ». Ainsi, demander l'affichage de mille triangles sera beaucoup plus rapide que de demander mille fois l'affichage d'un triangle, pour un résultat visuel similaire. Nous souhaitons donc découper l'affichage de notre image avec le moins d'appels possible.

Pour cela, la première étape est de regrouper les textures de tous les plans de réflexion en une seule texture. Par exemple, au lieu d'avoir quatre textures de 128x128, chacune associée à son plan de réflexion, nous aurons une seule texture de 256x256, chaque quart de l'image stockant l'une des quatre textures originales. Cette technique est désignée sous le nom de « atlas de textures ». Grâce à cet atlas, nous ne sommes plus obligés de changer la texture entre l'affichage des quatre plans. Nous pouvons donc directement demander l'affichage d'un objet constitué de quatre plans, chacun ayant ses coordonnées de textures modifiées pour pointer sur la bonne zone de l'atlas de texture. La création de cet atlas est effectuée automatiquement en regroupant tous les plans associés à une cube-map donnée. Les textures sont d'ailleurs en général créées automatiquement en capturant une image de la scène lors d'une phase de précalcul, à l'aide d'une caméra orthographique qui enregistre les couleurs de la scène dans une zone autour du quadrilatère de réflexion. Cette capture automatique permet de rendre la

création des plans de parallaxe très simple. Il suffit de les placer dans le décor près des objets à représenter et le reste de la capture est automatique.

Une différence existe cependant entre les plans qui vont bouger et ceux qui restent statiques. En effet, le maillage qui regroupe les plans statiques peut être stocké sur la carte graphique puis ne plus être modifié. Seul l'envoi d'une commande sera nécessaire à l'affichage, qui spécifiera notamment la position actuelle de la caméra. Les objets dynamiques, qui peuvent bouger d'une image à la suivante, doivent être recalculés à chaque image, et donc transférés vers la carte graphique à chaque image. Ils sont donc un peu plus lourds à rendre (~ 3 fois). Malheureusement, si nous souhaitons trier les plans en fonction de leurs distances à la caméra, le maillage doit être dynamique puisqu'il changera d'une image sur l'autre. Nous avons donc préféré laisser le choix à l'artiste de donner un ordre d'affichage fixe aux plans qui ne bougeront pas, ou bien de passer les objets en dynamique ce qui permettra le tri, mais sera plus lourd. Nous sommes par contre obligés de choisir un ordre d'affichage entre les deux groupes d'objets (statiques et dynamiques). Nous avons choisi d'afficher les dynamiques par-dessus les statiques. Les objets fixes sont donc toujours affichés en fond, derrière les objets qui sont dynamiques. Dans le pire des cas, si certains décors du jeu posent problème, nous avons toujours la solution d'utiliser uniquement des dynamiques qui seront alors triés entre eux. Une seconde subdivision sera opérée en fonction du mode de mélange des plans. Ainsi, nous supportons deux modes de mélange : alpha-blend et additif. Ces deux modes créeront deux groupes d'objets séparés, les additifs se plaçant derrière les alpha-blends afin de profiter pleinement de l'occlusion que permet l'alpha-blending. Ce système a suffi à nos besoins, mais si nous souhaitions autoriser n'importe quelle combinaison avec un ordre d'affichage en fonction de la distance à la caméra, nous pourrions créer un seul maillage à chaque image, triée par le processeur principal pour tous les plans, puis afficher d'un seul coup autant de plans qu'il est possible tant qu'il n'y a pas de changement en terme de mode de mélange (additif ou avec alpha). Cette méthode serait plus coûteuse en performance, mais moins difficile à contrôler puisque tous les plans seraient triés et traités de la même manière.

6.3.h Tri et mélange des proxies de réflexion :

Les techniques existantes de triage des plans entre eux sont loin d'être parfaites. C'est en effet un problème qui n'a pas de solutions exactes. C'est le problème dit de « l'algorithme du peintre », dans lequel nous cherchons à ajouter d'abord les parties les plus lointaines de l'image, afin que les plus proches soient dessinées en dernier, devant tout le reste. Un cas

particulier est indécidable, celui dans lequel les plans se traversent les uns les autres. La solution classique est d'utiliser un Z buffer, mais cette technique n'est pas adaptée aux plans transparents. La méthode de tri la plus utilisée est celle de l'arbre BSP (Binary Space Partition) qui permet de précalculer les découpes des plans nécessaires puis de les trier rapidement en temps réel selon la position de la caméra. La création d'un BSP à chaque image est envisageable grâce au faible nombre de plans, mais reste coûteuse. Pour notre application, nous sommes restés à un simple tri en fonction de la distance du centre du plan à la caméra. Certains cas ne sont pas correctement affichés, mais dans l'ensemble l'algorithme suffit. Sa rapidité est un atout qui compte.

Chaque cube-map possède une liste de proxies à utiliser. Elle définit également le plan de réflexion qui autorisera la correction de la parallaxe pour le plan du sol. Afin de pouvoir passer d'une ambiance locale à la suivante dans le décor, nous allons devoir effectuer une transition entre les cube-maps, mais également entre les plans proxies. La technique idéale serait de calculer l'image de chaque cube-map en même temps que ses proxies, puis de mélanger ces images ensuite en fonction du poids de la cube-map. Nous avons préféré pour des raisons de performances afficher toutes les cube-maps à mélanger dans la même image, chaque cube-map et chaque proxy étant rendu plus ou moins en transparence en fonction du poids associé.

Figure III.52: Utilisation de réflexions flous avec prise en compte de la parallaxe dans « Remember me »

6.4 Les réflexions floues

Au cours de la partie précédente, nous avons vu comment générer rapidement une texture en faible résolution avec peu de crénelage en utilisant des proxies. Nous souhaitons maintenant pouvoir utiliser cette texture dans des modèles d'éclairage de matériaux complexes. Le

principal objectif est de pouvoir afficher une réflexion plus ou moins floue en fonction de la rugosité du matériau.

Pour cela, nous allons effectuer le même type de processus que pour les cube-maps, c'est-à-dire la création d'une pyramide de mip-maps, chaque niveau représentant la texture à un degré de flou de plus en plus élevé.

Création et filtrage de la pyramide des mip-maps

Il est possible de rendre directement chacun des niveaux de mip-map séparément à partir des proxies. Ainsi, nous rendrions les cube-maps et les proxies dans une première texture de 256x256, puis dans une nouvelle de 128x128 et ainsi de suite. Cette solution est assez lente puisque le nombre de proxies à rendre peut être assez élevé, et la qualité n'est pas très bonne dans les bas niveaux de mip-maps, car le nombre de pixels à rendre n'est plus assez grand pour éviter un crénelage inacceptable. Avec ces faibles résolutions, la plupart des proxies finissent par mesurer moins d'un pixel et sont éliminés sans participer au rendu.

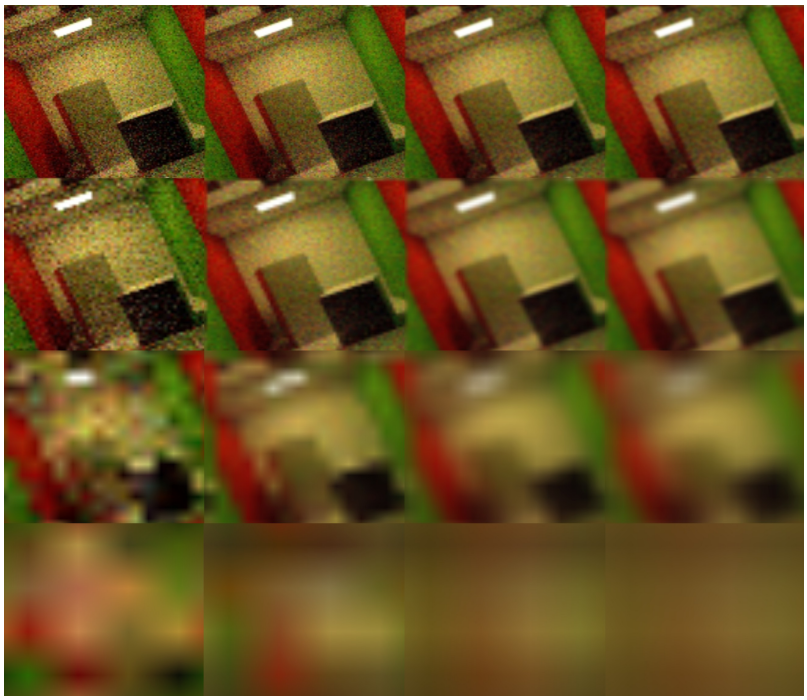


Figure III.53: Comparaison des techniques de création de la pyramide des mip-maps.
Les colonnes de gauche à droite sont les techniques: décimation, bilinéaire, 3x3 et 5x5. Les lignes sont les niveaux de mip-maps de haut en bas : 128, 64, 16 et 4. La texture de départ à une résolution de 256x256.

Une bien meilleure solution est d'utiliser le premier mip-map calculé pour générer le niveau suivant. Chaque niveau va utiliser l'image du niveau supérieur en s'occupant uniquement de la diminution de résolution et du calcul du flou, qui est ainsi réparti sur chaque niveau.

Nous avons testé différentes méthodes de calcul du flou pour passer d'un niveau de la texture au suivant, dont la taille est divisée par deux sur les deux axes. Il s'agit en effet d'obtenir un flou qui élimine au maximum le crénelage spatial et temporel tout en étant le plus efficace au niveau performance. La diminution de résolution entre les niveaux de mip-map implique l'utilisation d'un flou important afin de supprimer les hautes fréquences.

La pire des méthodes est la « décimation » qui utilise un seul des 4 pixels du niveau supérieur. Globalement, les trois quarts des pixels sont complètement perdus. Cette méthode est très destructrice et produit beaucoup de crénelage. La seconde méthode est d'utiliser la moyenne de quatre pixels pour calculer un pixel du niveau suivant. Ainsi, toute l'information de base participe au calcul des niveaux suivant et remonte jusqu'au dernier niveau (taille 1x1), qui est la moyenne de tous les pixels de l'image. Cette technique s'apparente au filtrage bilinéaire. Le crénelage reste un problème, principalement lors des mouvements de caméra.Nous avons testé les différentes méthodes en échantillonnant un plan en rotation. Cela nous permet de juger de la qualité du flou et de la présence ou non de crénelage temporel perturbant. Au-delà de l'utilisation des quatre pixels, il est possible de prendre en compte plus de points et de les affecter de poids suivant une courbe gaussienne par exemple. Nous avons testé en utilisant une distribution sur une grille de 3x3 pixels et avec une grille de 5x5. Ces grilles peuvent être calculées efficacement en utilisant la fonctionnalité de filtrage bilinéaire qui permet d'obtenir une moyenne de quatre pixels en un seul appel. Ainsi, la grille de 3x3 pixels pourra être calculée en utilisant 4 échantillons filtrés et la grille de 5x5 en utilisant 9 échantillons filtrés.

Voici le schéma des grilles et l'emplacement des échantillons, ainsi que le poids attribué à chaque pixel dans la moyenne pondérée finale :

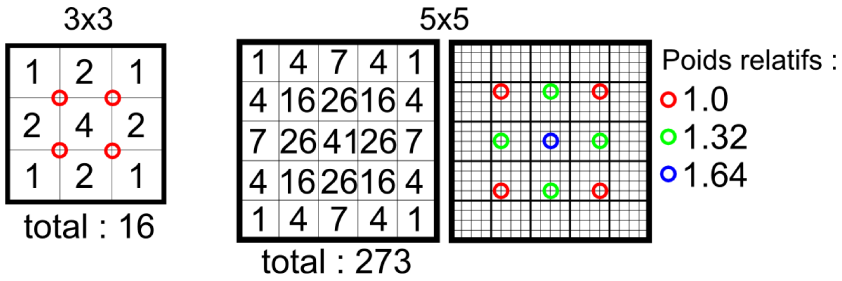


Figure III.54: Filtres de flou 3x3 et 5x5.
Placement des échantillons et poids des pixels individuels dans la moyenne finale pour les grilles de 3x3 et de 5x5

Il est bien évidemment possible de profiter de la séparabilité du flou gaussien pour diviser l'application du filtre en deux parties successives, chacune appliquant le filtre sur un axe seulement, réduisant ainsi le coût si le nombre d'échantillons nécessaires est important. Dans notre cas, nous souhaitons un flou assez faible par niveau et l'utilisation de la pyramide des mip-maps permet déjà d'alléger le calcul par la réutilisation des flous calculés aux niveaux précédents.

Figure III.55: Détails de réflexions par proxies

Conservation de la qualité dans les mip-maps

Obtenir une grande qualité de couleurs et de dégradés dans cette image de réflexion peut être difficile. Il s'agit également de gérer une échelle de luminosité étendue. L'importance du rendu HDR (High Dynamic Range) a déjà été présentée dans le cadre de la profondeur de champ. Les mêmes remarques s'appliquent ici. Les hautes lumières sont celles qui seront les plus visibles lorsque le matériau produit une réflexion floue. Une lumière très petite en taille, mais

très forte en luminosité va ainsi s'étaler sur toute une zone au fur et à mesure de l'augmentation de la puissance spéculaire. Avec une échelle de valeur trop limitée, la luminosité de départ sera rapidement saturée et ne produira plus de changements dans le rendu final. Les couleurs peuvent également souffrir d'une échelle limitée et devenir blanches.

Les textures de bases utilisées pour les proxies de réflexions doivent être compressées en DXT1 (ou DXT5) afin d'obtenir de bonnes performances et un faible coût en mémoire. L'utilisation de la courbe gamma sRGB, qui est gérée automatiquement par la plupart des cartes graphiques, permet d'obtenir une meilleure précision dans les valeurs sombres en suivant mieux la courbe de perception logarithmique de l'œil humain. Nous avons décidé d'une solution simple permettant de gérer au maximum des valeurs allant jusqu'à deux fois la luminosité normale. Il s'agit simplement de diviser par deux les valeurs stockées dans la texture puis de les multiplier par deux au moment de l'utilisation finale. Nous diminuons la précision des couleurs par deux, mais nous obtenons une échelle de valeurs deux fois plus étendue. Il est également possible de chercher dans l'image la valeur maximum de luminosité puis de l'utiliser comme limite de l'échelle utilisée. Tous les pixels seront divisés par ce maximum afin d'utiliser la meilleure échelle de valeurs possible. La valeur maximum de luminosité est stockée à part de la texture et sera multipliée aux couleurs de la texture dans le shader, juste avant l'affichage, afin de retrouver la luminosité de l'image originale.

Nous avons ainsi des textures de base d'une qualité satisfaisante. Il est maintenant nécessaire de conserver cette qualité au cours de la création de la texture de réflexion et au cours du filtrage lié à la génération des niveaux de mip-maps. Le format de texture habituellement utilisé est le RGBA8, c'est-à-dire que chaque canal (rouge, vert, bleu et alpha) est encodé sur 8 bits. Malheureusement, ce format 8 bits ne peut pas être utilisé avec une courbe gamma, c'est à dire avec un format sRBG, ce qui nous assurerait une meilleure répartition des valeurs. Cette restriction provient du fait que le matériel n'est pas capable d'effectuer correctement le mélange de couleur dans ce format lorsque nous utilisons la transparence. Notre technique repose sur l'utilisation de la transparence lors de la création de l'image de réflexion, pour les raisons précisées précédemment. Un mélange de couleurs dans l'espace gamma produit un résultat faux et n'est donc pas adapté à notre application.

Avec seulement 8 bits en espace linéaire, la qualité et l'échelle de luminosité sont assez faibles. Nous avons donc fini par utiliser un format flottant en 16 bits par composant. La qualité est alors plus élevée. Nous pouvons gérer des valeurs très fortes proprement, sans

détériorer les couleurs. Les dégradés liés aux flous des niveaux de mip-maps sont beaucoup plus doux et réguliers.

6.5 Variations en fonction des textures de rugosité et de normale

6.5.a Utilisation de la texture de rugosité

Afin de donner du caractère à la matière que nous cherchons à modéliser, nous utilisons des textures pour faire varier l'aspect de la surface. Au-delà de la simple représentation d'une surface sous-jacente, la variété dans l'aspect de la surface sert également à masquer les artefacts des différentes techniques employées et ainsi créer une image complexe, mais qui semble cohérente.

L'utilisation directe de la texture de réflexion sur tout un plan sans aucunes variations produit un résultat visuel assez pauvre et met en évidence les artefacts indésirables tels que le crénelage, la pauvre qualité du flou, la faible résolution et l'utilisation de proxies au lieu de la géométrie complète. Le degré de réflexion spéculaire est souvent difficile à déterminer précisément par observation. Par contre, la variation de ce degré est bien plus perceptible. C'est donc en jouant sur ces variations qu'une matière acquiert son identité et une plastique propre.

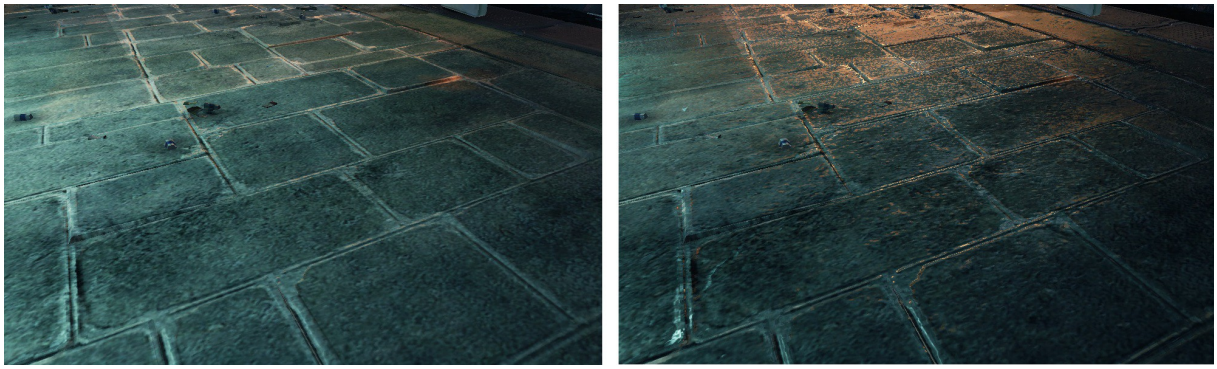


Figure III.56: La variation du degré de rugosité change la perception de la matière. Ici, nous passons d'un matériau sec avec un résidu humide dans les creux à une surface mouillée uniformément. Les textures utilisées sont les mêmes mais sont interprétées différemment lorsque la pluie est présente.

La variation du degré de rugosité dans notre modèle d'éclairage créé ainsi des surfaces lisses et d'autres rugueuses, mais également des réflexions spéculaires fortes, des « points chauds », et des réflexions diffuses mattes. La variation de la rugosité est un moyen très subtil, mais efficace pour manipuler l'aspect visuel de la matière.

La texture de normale fait varier plus radicalement l'aspect de la surface, surtout en présence d'une forte spécularité. Elle doit ainsi être utilisée pour représenter les gros détails de la surface. Utiliser la texture de normale pour modéliser des rides sur un visage, par exemple, produit en général des marques trop profondes et d'assez mauvaise qualité si la texture de normale n'a pas une énorme résolution. Utiliser la texture de rugosité pour souligner la différence de spécularité dans les creux des rides est par contre plus subtil et plus naturel.

6.5.b Distorsion de la réflexion en fonction de la normale

Les variations situées dans la texture de normale modifient directement le vecteur de réflexion servant aux calculs du modèle lumineux. Dans le cas de l'utilisation directe d'une cube-map d'environnement, nous pouvons également nous servir de ce vecteur de réflexion modifié pour faire varier l'emplacement de l'échantillon dans la texture cubique. La réflexion est ainsi naturellement influencée par la texture de normale.

Par contre, dans le cas de l'utilisation d'une texture de réflexion 2D liée à la réflexion plane, ce vecteur ne peut pas être directement utilisé. La texture 2D ne contient pas les informations lumineuses sur toute une sphère, mais uniquement sur l'intérieur du cadre de l'écran.

Figure III.57: Distorsion de la réflexion et prise en compte de la texture de rugosité.

Nous allons donc chercher à reproduire visuellement les variations de l'aspect de l'image en fonction des variations du vecteur de réflexion. Il s'agit d'une distorsion des coordonnées de texture utilisées pour échantillonner la texture de réflexion. Nous utilisons directement les valeurs des axes X et Y de la texture de normale comme vecteur de distorsion. C'est en effet sur les canaux X et Y que l'essentiel des variations sont présentent dans la texture de normale. Ces informations sont disponibles dans l'espace tangent, et non dans l'espace-écran. Or, la texture de réflexion est échantillonnée avec des coordonnées en espace-écran. Il est possible de transférer la normale dans l'espace-écran. Visuellement, le sens de la distorsion sera alors

cohérent avec la direction de la surface dans le repère du monde. Concrètement, les creux d'une surface ressembleront bien à des creux en distordant la réflexion vers la concavité. Sans cette rotation dans l'espace-écran, la distorsion suivra un sens fixe lié aux coordonnées de texture. Cependant, la différence est assez peu perceptible dès lors que la surface est suffisamment détaillée, et la version simple suffit amplement à produire des variations plaisantes et crédibles.

Une question se pose cependant au-delà de la direction de la distorsion : celle de son ampleur. Nous avons opté pour un facteur multiplicateur fixe que l'artiste qui créé le matériau peut régler afin d'ajuster l'ampleur de la distorsion en fonction de la matière et de la texture de normale associée. En effet, si une amplitude de distorsion précise fonctionne dans la plupart des cas, il est utile de pouvoir la modifier si certaines textures de normales sont trop bruitées, par exemple, et que la distorsion produite est ainsi peu compréhensible et peu crédible. Au contraire, des surfaces douces peuvent bénéficier d'un accroissement du facteur de distorsion afin d'accentuer les petites variations de la texture de normale.

6.5.c Modèle d'éclairage de la réflexion ambiante

Voici des extraits du code HLSL nous permettant de simuler la réflexion ambiante à partir d'une cube-map ou d'une texture de réflexion plane mip-mappée. L'influence des textures de normale, de rugosité et de couleur spéculaire est également précisée.

Échantillonnage dans le cas d'une cube-map :

*EnvColor = texCUBElod(EnvMap, float4(R , rugosité * (MaxMip - 1)))*

EnvMap: texture de la cube-map
MaxMip : nombre maximum de niveaux de mip-map
R = vecteur de réflexion en espace monde

Échantillonnage dans le cas d'une texture de réflexion plane :

*EnvColor = tex2Dlod(RTex, float4(SP + TN.xy * Dist, 0.0f, r * (MaxMip – 1)))*

Rtex : texture de réflexion
SP : coordonnées du pixel en espace-écran (entre 0 et 1 sur deux axes)
TN.xy : valeur de la texture de normale sur les axes X et Y
Dist : facteur de distorsion (réglé empiriquement)
r : rugosité (entre 0 et 1)
MaxMip : nombre maximum de niveaux de mip-map

Calcul du facteur de Fresnel par l'approximation de Schlick :

*fresnel = CSpec + (max(rugosité , CSpec) - CSpec) * (1.0f - saturate(dot(N, C)) ^ 5*
CSpec : couleur spéculaire de la surface
N : normale de la surface (espace tangent)
C : direction de la surface vers la caméra (espace tangent)

Et finalement le calcul de la couleur ambiante finale :

*color = fresnel * EnvColor*

7 Conclusion

Cette partie nous a permis d'étudier les paramètres qui influencent l'aspect des matières virtuelles. La simulation physiquement réaliste des matières a un rôle important à la fois pour la crédibilité de l'image, pour sa lisibilité, mais aussi pour procurer à l'artiste des contrôles intuitifs sur la création de ses matières. L'influence de la lumière, et notamment de la réflexion ambiante spéculaire est prépondérante. Pourtant, celle-ci est encore un des effets les plus complexe à produire en temps réel.

Les équations de réflexion des lumières ponctuelles ont été présentées, notamment par le biais de différentes fonctions de BRDF (Bidirectional Reflectance Distribution Function). La représentation de la rugosité de la surface, et de la dualité entre la texture de normale et la texture de rugosité, nous a également mis sur la voie des algorithmes qui combinent les deux types de textures.

Nous avons étudié différentes solutions au problème du stockage et de l'utilisation d'échantillons lumineux spéculaires. Les solutions récentes du domaine sont lourdes en temps de calcul, mais apportent de nombreux éléments de réponse à notre questionnement.

Nous avons introduit une première contribution, sous la forme d'un format particulier de stockage de la réflexion ambiante spéculaire statique.

Notre seconde et principale contribution concerne le problème de la parallaxe lors de l'utilisation de cube-maps pour le calcul de la réflexion spéculaire ambiante. Notre technique à base de volumes de projection de cube-maps associés à des plans (des « proxies ») qui représentent le décor nous permet d'utiliser les effets de réflexions sur tous les sols d'un jeu vidéo. Le coût est très réduit, laissant l'essentiel de la puissance de la machine disponible pour le reste de l'affichage graphique et de la simulation du jeu.

La prise en compte de la rugosité de la matière par un flou sur la réflexion est un des apports visuels majeurs de ce type de technique. Notre solution est capable de produire cet effet très efficacement. Le coût moyen de l'algorithme de réflexion pour le sol se situe aux alentours des 2ms.

Les perspectives de recherches futures sont nombreuses, notamment sur deux axes : la gestion de la parallaxe sur plusieurs plans de réflexion simultanés et l'amélioration de la qualité de la réflexion au niveau du flou. L'ajout d'une gestion de l'anisotropie de la réflexion, c'est-à-dire un étirement de la réflexion dans un sens particulier, pourrait également être intégré rapidement à notre technique.

Chapitre IV - Les fluides

1 Introduction aux effets de fluides

1.1 Les fluides comme élément de mouvement

Jusqu'à présent, nous avons envisagé le mouvement de l'image numérique essentiellement comme résultant du mouvement de la caméra. Le flou de profondeur et les effets de matières ne présentent pas de mouvement en soi, mais évoluent en fonction de la position de l'observateur. Les fluides nous proposent une autre idée du mouvement, qui devient une part indispensable de la crédibilité de l'effet. La tache de mise en image de cette partie est en effet délicate puisque l'attrait principal d'un fluide est son mouvement, qui ne peut être retransmis dans l'image fixe. Un second point important dans l'évaluation de la qualité visuelle d'un effet de fluide concerne la répétition des détails dans l'image. En effet, il est important de préserver le maximum de chaos dans les détails, tout en conservant l'impression de mouvement général, afin d'éviter l'aspect artificiel qui surgit lorsque nous remarquons trop les répétitions.

Même en image fixe, la représentation des fluides évoque le mouvement et a fasciné de nombreux peintres, dans différents styles. Plusieurs artistes du numérique se sont emparés de ces œuvres et ont tenté d'en reproduire les effets en les adaptant à l'animation numérique. Karl Sims s'est intéressé, pour un film de Mark Whitney, à mettre en mouvement certains des dessins de DeVinci sur le déluge[74]. Le film s'intitule : « Excerpts from Leonardo's Deluge ». De même, l'artiste Petros Vrellis[75] a mis en animation le ciel torturé du tableau « La Nuit étoilée » de Van Gogh. Le mouvement des nuages, si perceptible dans la peinture originale, devient ici un mouvement effectif, que le spectateur peut même modifier via une interface tactile. La démarche technique de ces deux artistes rejoint la notre dans cette partie, puisqu'il

74 Karl Sims, « Choreographed Image Flow », *The Journal of Visualization and Computer Animation* 3 (1992): 31-43.

75 Petros Vrellis, « Petros Vrellis on Vimeo », 2012, http://vimeo.com/user10348450.

s'agit de donner une illusion de mouvement tout en respectant les caractéristiques d'une image, pour eux celle d'une œuvre, pour nous celle d'une texture de fluide.

Figure IV.1: Deux tableaux mis en animation par des artistes du numérique.
À gauche, DeVinci, à droite « La Nuit étoilée » de Van Gogh

1.2 Les fluides dans les jeux vidéo

Les fluides sont également des éléments interactifs. Selon les jeux, la simulation interactive de l'eau est simplement un ajout visuel qui met en valeur les actions de la scène, ou bien un véritable élément indissociable du principe de jeu. Les simulations de fluides en deux dimensions sont devenues suffisamment stables et prédictibles pour permettre leur utilisation dans des jeux comme élément majeur du « gameplay ».

Figure IV.2: Trois exemples d'utilisations de la dynamique des fluides dans le cœur du mécanisme d'un jeu en 2D.
De gauche à droite, les jeux « PixelJunk Shooter » et « Puddle » ainsi que l'application ludique « PowderToy », qui permet de jouer avec tous types de fluides.

Dans les jeux 3D, la simulation dynamique est souvent remplacée par des outils de contrôles paramétriques afin de limiter l'intervention du hasard. C'est le cas du jeu de course aquatique « Wave Race Blue Storm » par exemple, où les vagues ont un mouvement périodique fixe. Le jeu « Bioshock » utilise l'eau comme un élément central de l'univers, mais la retranscrit essentiellement par des effets réglés manuellement, sans simulation complexe.

Figure IV.3: À gauche, « Wave Race – Blue Storm » et ses vagues paramétriques. À droite, les effets d'eau impressionnants de « Bioshock ».

Le jeu « From dust » se distingue en mettant la simulation de fluides au centre du jeu. Le joueur peut creuser dans des dunes de sable pour modeler les étendues d'eau et gérer les contraintes de l'environnement telles que des tsunamis et de la lave. Le jeu « Hydrophobia » nous propose une véritable simulation de fluide. Le personnage principal évolue dans des décors intérieurs envahis par l'eau et se fait régulièrement surprendre lors des ouvertures de portes en se retrouvant inondé par un déferlement de liquide.

Figure IV.4: Les jeux « From Dust » (à gauche) et « Hydrophobia » intègrent l'eau sous la forme de simulations. L'expérience ludique en est directement affectée.

L'intérêt purement visuel est mis en avant dans « Crysis » ou dans « Empire Total War », par exemple, ou l'eau est un atout esthétique fort, mais dont le mouvement n'influence pas la jouabilité.

Figure IV.5: Les jeux « Crysis » à gauche et « Empire Total War » à droite utilisent l'eau comme élément esthétique, mais n'utilisent pas de simulations de fluides.

1.3 Les fluides du point de vue physique

En physique, un fluide désigne une substance capable de se déformer en fonction des pressions appliquées. Les fluides prennent la forme du récipient qui les contient, car ils peuvent se déformer librement. Le terme regroupe trois états de la matière : les liquides, les gaz et les plasmas. Certains solides « coulent » également à une vitesse très faible, comme le verre. Les matières en grains, telles que le sable[76], ont également un comportement visuel proche des fluides bien que réagissant très différemment dans des situations très spécifiques[77].

Dans le domaine du développement de jeux vidéo, les fluides désignent plus spécifiquement les liquides et les fumées. Ces deux éléments sont très différents visuellement, mais ils possèdent un point commun important dans leur mouvement. Ils obéissent tous deux aux équations de la physique des fluides qui vont déterminer l'évolution des composants de la matière. Par contre, du point de vue du développeur, ces effets sont plus souvent désignés et regroupés par les algorithmes qui servent à les simuler. Ainsi, de nombreux effets qui correspondent dans la réalité à des fluides, comme des fumées, des surfaces liquides, des nuages et des effets de particules, n'utilisent pas vraiment les équations des fluides dans l'algorithme, mais plutôt des techniques déterministes qui cherchent à reproduire l'effet visuel sans modéliser la réalité qui est sous-jacente.

La séparation entre l'état liquide et l'état gazeux semble très franche, mais la différence de comportement entre les deux états est plus ambiguë. Les liquides forment une surface perceptible sur la limite entre le liquide et l'air alors que les gaz présentent un volume sans délimitation franche. Cependant, les calculs généraux sont les mêmes, bien que les algorithmes utilisés puissent être plus adaptés à un certain type de liquide ou de gaz. La différence visuelle entre une fumée se propageant dans l'air et une encre se propageant dans l'eau est faible, le mouvement est similaire et l'aspect également.

Au-delà de cette séparation entre liquides et gaz, l'aspect des différents effets voulus va mener à des distinctions visuelles plus précises. Ainsi, du côté des liquides, la différence d'échelle produit des aspects différents. À des échelles moyennes, les liquides se présentent sous la forme de gouttes qui s'agrègent ensemble pour former une surface en trois dimensions. À un niveau plus élevé, le liquide prend la forme d'une rivière ou d'un fleuve, et c'est alors la

76 M. Pla-Castells, I. García-Fernandez, et R. J. Martinez-Dura, « Physically-based interactive sand simulation », *Eurographics 2008-Short Papers* (2008): 21–24.

77 Y. Zhu et R. Bridson, « Animating sand as a fluid », in *ACM Transactions on Graphics (TOG)*, vol. 24, 2005, 965–972.

surface bidimensionnelle qui devient importante, le liquide restant essentiellement confiné dans le lit de la rivière. Les gaz peuvent être vus comme des volumes brumeux ou comme des fumées épaisses. Les liquides et les gaz sont parfois de simples véhicules invisibles qui mettent en mouvement des objets qui eux sont visibles. Ainsi, des feuilles emportées par le vent seront influencées par les équations des fluides. Le feu fait également partie des effets qui obéissent à ces équations, car c'est le mouvement de l'air, avec ses différentes températures, qui modèle les zones de combustions et qui disperse la fumée. Les explosions peuvent également faire partie de ce modèle, par une gestion de la température de l'air en conjonction avec sa pression.

Le visuel d'un fluide est influencé par la substance qui le constitue bien sûr et par les objets qu'il transporte, mais surtout par les collisions avec le décor alentour. Un fluide sans aucune collision serait assez pauvre visuellement, car ce sont les perturbations introduites par les obstacles qui vont rendre le fluide complexe et contrasté. Les changements de vitesse artificiels dans le fluide peuvent également jouer ce rôle, par exemple si deux arrivées de flux d'air introduisent des vitesses de fluide contraires qui amènent le chaos.

Au cours de cette étude, nous allons nous intéresser aux mouvements des liquides, principalement sous la forme de grandes surfaces d'eau telles que des fleuves ou des rivières.

Si nous envisageons le fluide comme un volume limité dans lequel des entrées et des sorties d'eau peuvent être pratiquées, il est intéressant de noter que les quantités d'eau qui entrent et celles qui sortent sont généralement les mêmes. Ainsi, la quantité d'eau ne change pas et le contenant ne finira pas par déborder, ou par imploser. Si le fluide est simulé pendant suffisamment longtemps, les perturbations initiales dues au démarrage de la simulation vont en général s’apaiser et l'écoulement va devenir stable, les vitesses à l'intérieur du fluide n'évolueront plus. En présence de collisions en mouvements, ou bien de flux d'apport d'eau changeant, le fluide restera majoritairement turbulent, chaotique et imprévisible.

2 Techniques déterministes non physiques

Préalablement à l'utilisation d'une simulation de fluide cherchant à reproduire les mouvements réels des fluides, nous allons présenter quelques techniques qui permettent d'imiter des mouvements de fluides d'une manière déterministe, c'est-à-dire sans mécanisme d'itération et sans calculs approximés d'une simulation « physique ».

2.1 Les simulations déterministes

Traditionnellement, les techniques utilisées dans les applications temps réel pour afficher un effet de fluide n'utilisent pas de simulation physique. La plupart utilisent des vidéos de fluides ou d'explosions qui vont être affichées sur des « sprites » (ou bill-board) c'est-à-dire des plans qui font toujours face à la caméra. Si l'effet nécessite beaucoup de « sprites », les développeurs utilisent généralement un système de particules pour les animer. Le mouvement général sera donc donné par le système de particules et les petits détails qui nécessiteraient beaucoup de calculs (volutes, écoulements...) seront issus de vidéos. Certains jeux utilisent une technique mixte, qui consiste à calculer un effet de fluide avec des équations physiques dans une petite résolution puis d'utiliser l'image ainsi calculée sur des dizaines de particules à la fois. La répétition n'est pas forcement perceptible, notamment si les particules varient en taille et en orientation. Les particules pourraient être considérées comme des éléments non déterministes étant donné qu'elles sont souvent mises en mouvement par des algorithmes itératifs. Cependant, tant que ces particules suivent des équations simples et sont indépendantes les unes des autres, le mouvement sera stable et résistant aux petites perturbations. Par contre, si les particules s'influencent et se perturbent entre elles, nous obtenons des effets chaotiques semblables aux fluides turbulents.

L'effet de vagues qui est présent à la surface des grandes étendues d'eau peut être simulé par des fonctions mathématiques simples, avec par exemple une simple accumulation de différentes sinusoïdes en décalage, comme avec les vagues de Tessendorf[78].

78 Jerry Tessendorf, « Simulating Ocean Water » (Proceedings of SIGGRAPH 2001, 2001).

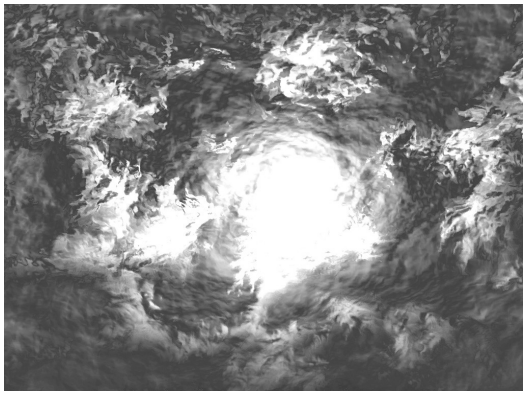


Figure IV.6: Utilisation de fonctions mathématiques et d'une texture de bruit pour un effet de nuages.

La différence fondamentale entre les techniques liées à une simulation physique et celles-ci tient dans le déterminisme des techniques non physiques. En effet, le processus d'advection lié à la simulation physique est itératif, c'est-à-dire que chaque résultat dépend du précédent, et donc de tout l'historique de la simulation. Au contraire, la plupart des techniques utilisées habituellement dans les jeux vidéos sont déterministes, et nous sommes donc capables de calculer directement le résultat de l'effet à l'image N, car il sera toujours le même. Bien sûr, les développeurs utilisent également des fonctions aléatoires, mais ce n'est pas essentiel pour l'effet et cela sert simplement à masquer le côté monotone de la technique utilisée. Ce déterministe est aussi la raison qui impose ces techniques comme choix habituel, car les développeurs et les artistes peuvent contrôler très précisément le résultat. En jouant sur quelques paramètres ou en modifiant les vidéos sources, nous pouvons obtenir des effets totalement différents, ce qui rend la technique utilisable de manière homogène pour la plupart des effets d'un jeu (explosions, fumées, chutes d'eau....). Au contraire, avec une simulation physique, nous pouvons essayer de respecter au maximum les valeurs réelles afin d'obtenir un effet réaliste, mais sur lequel nous avons peu de contrôle et où le moindre changement de paramètres peut modifier totalement le résultat de manière imprévue.

Le domaine des techniques déterministes peut également s'étendre aux algorithmes de précalcul de grilles de fluides. Ainsi, nous avons déjà mentionné la possibilité de laisser une simulation de fluide tourner sans perturbations jusqu'à ce que les vitesses deviennent stables. Une fois la vitesse stable, l'état du fluide peut être sauvegardé puis utilisé pour influencer la vélocité de particules, ou bien pour l'advection d'une texture, dont nous parlerons plus loin. Par contre, le fluide est en général peu perturbé et donc également peu intéressant

visuellement. Le Curl Noise[79] est ainsi une technique de calcul d'un champ de vélocité qui assure la non-divergence du fluide. Concrètement, le fluide est incompressible, les particules qui suivent un tel champ ne peuvent pas rester bloqué à un endroit et n'aurons pas tendance à former des groupes. L'algorithme utilise différentes primitives en entrées : des vecteurs de vitesses préalables qui permettent de donner une direction globale au fluide, une atténuation de la vitesse aux abords des obstacles, et surtout du bruit de perturbation. À partir de la combinaison de ces primitives, l'algorithme calcule un fluide non divergent, et nous pouvons ensuite l'utiliser pour obtenir un mouvement déterministe, mais avec un aspect très turbulent.

2.2 Les mouvements de texture simples

En plus de la simulation du mouvement de la surface ou des éléments particulaires du fluide, il est souvent nécessaire d'ajouter du détail par l'intermédiaire de textures. Nous allons ici étudier quelques techniques permettant de donner du mouvement à ces textures. Nous parlons ici essentiellement des textures de surfaces liquides, mais ces remarques sont également valables lorsqu'il s'agit de représenter des textures de fumées.

2.2.a Mouvements de texture sans direction précise

Il s'agit de donner un mouvement de clapotis général par le biais de plusieurs couches de textures en mouvements, tout en évitant le plus possible que nous puissions percevoir une direction principale au mouvement. Nous cherchons à donner un aspect aquatique aux mouvement des détails de la surface sans pour autant nécessiter une simulation complexe. Cette technique peut servir dans les endroits stagnants, protégés du vent et du courant. L'intensité du mouvement augmente généralement avec la profondeur moyenne de l'eau. Néanmoins, il est assez difficile d'obtenir un mouvement de clapotis sans impression de mouvement global, surtout par des techniques à base de déplacements et d'accumulations d'un certain nombre de textures. Une solution est d'utiliser une animation précalculée qui boucle temporellement et spatialement. Cette animation peut être précalculée par une simulation de fluide physiquement réaliste. La texture se présente alors comme un bruit tridimensionnel, dont une des dimensions est le temps.

79 R. Bridson, J. Houriham, et M. Nordenstam, « Curl-noise for procedural fluid flow », in *ACM Transactions on Graphics (TOG)*, vol. 26, 2007, 46.

2.2.b Mouvements de texture avec une direction globale

Ici, le mouvement présente une direction générale de déplacement qui est la même en tout point de la surface du fluide. C'est la technique la plus facile à mettre en place. L'utilisation de cette technique se justifie par l'influence du vent sur la surface de l'eau. En effet, dans une eau relativement profonde et avec une quantité d'obstacles faible, tel qu'un canal large ou un lac, c'est l'influence du vent qui va déterminer principalement la direction des vagues. Dans le cas d'un canal large, l'intensité du courant et sa direction ont tout de même une influence, car les vagues sont plus fortes lorsque le vent et le courant sont en sens contraire et se font face, ce qui soulève les vagues et agite la surface. Nous pouvons néanmoins considérer le vent comme ayant une direction uniforme. Les différences dans l'intensité du vent peuvent être gérées en précalculant par exemple une texture d'intensité qui dépendrait de la direction du vent et des obstacles au vent. Cette texture d'intensité ne sert pas à modifier la vitesse de déplacement, mais à aplatir plus ou moins les vagues selon que l'intensité est grande ou non.

2.2.c Mouvements de texture avec une direction locale unique

Cette technique est la même que la précédente, mais s'applique à un cas particulier. Il s'agit en fait de calculer la direction du vent et du courant à l'endroit ou se trouve la caméra ou le personnage principal. Toute la surface adopte alors ce sens de mouvement. L'influence du courant peut être réduite en fonction de la distance entre la surface et la caméra afin de progressivement faire disparaître l'impression de mouvement global. C'est une technique utile pour une texture de déchets ou de saleté emportés par l'eau, car en général ces déchets deviennent indistinguables au-delà d'une certaine distance. Cette technique peut être mixée avec la précédente pour marquer la différence de comportement entre les vagues et les saletés. En effet, sur un fleuve large par exemple, les vagues vont dans le sens du vent, donc un sens fixe global, alors que les saletés sur l'eau à proximité de la vue flottent et sont emportées par le courant local. L'important dans cette technique est d'éviter les changements bruts de direction de courant qui seraient perceptibles sur toute la surface de l'eau.

Au niveau de l'implémentation, il faut noter que le calcul du décalage de la texture ne se fait pas d'une manière déterministe, c'est-à-dire en utilisant uniquement la direction de décalage et le temps passé depuis le début de la simulation, mais d'une façon dynamique et itérative, avec la conservation d'un état, le décalage, d'une image à la suivante.

Technique déterministe : $P(i+1)=P(0)+V(i+1)\times t\times(i+1)$
Technique dynamique : $P(i+1)=P(i)+V(i+1)\times t$

P(i) : position/décalage à l'image i
V(i) : vélocité à l'image i
t : durée d'une image (durée fixe)

3 La simulation physique des fluides

3.1 Les équations de Navier-Stokes

La base théorique de la mécanique des fluides que nous utilisons dans les applications temps réels se trouve dans les équations de Navier-Stokes. Ces équations décrivent l'ensemble de la mécanique des fluides et se déclinent en de nombreuses formulations selon le domaine et l'application observée. Ainsi, ces équations permettent tout autant de calculer des débits et des pressions dans des tuyaux que de modéliser le mouvement de l'air autour des ailes d'un avion. Les équations générales sont bien trop complexes pour être adaptable directement, et nous allons nous concentrer sur une petite partie de ces équations, qui suffisent à produire des résultats visuels convainquant en temps réel. Notamment, nous nous limitons aux fluides incompressibles. Dans ce cas la, la substance concernée (air, eau, encre…) conserve toujours le même volume, ce qui n'est pas vrai dans le cas général et notamment pour les gaz qui peuvent se mélanger à l'air et ainsi changer de densité et donc de volume.

L'équation principale simplifié concernée est la suivante[80] :

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \vec{g} + v \nabla \cdot \nabla \vec{u}$$

$$\nabla \cdot \vec{u} = 0$$

Au lieu de chercher une réelle compréhension de cette équation, nous pensons qu'il est plus intéressant de commencer par lister les différentes valeurs qui y jouent un rôle :

- le vecteur u désigne la vélocité du fluide, c'est à dire sa vitesse.
- La lettre grecque ρ est la densité du fluide qui est différente selon le liquide ou le gaz concerné. Elle s'exprime en kg/m^3
- le vecteur g regroupe l'ensemble des forces globales s'appliquant sur tout le fluide, notamment la gravité
- la lettre grecque ν désigne la viscosité du fluide, et mesure la résistance du fluide aux déformations.

80 R. Bridson et M. Müller-Fischer, « Fluid simulation », in *ACM SIGGRAPH 2007 courses*, 2007, 1–81.

- ∇ désigne le gradient, ici soit de la vitesse $\nabla \vec{u}$ ou de la pression ∇p, c'est-à-dire la dérivée spatiale en termes mathématiques. En trois dimensions, nous pouvons le voir comme un vecteur indiquant la direction dans laquelle le paramètre augmente.
- $\nabla \cdot \nabla$ désigne l'opérateur laplacien, qui indique la différence d'une valeur entre un point et la moyenne des valeurs de la zone. Ainsi, le vecteur laplacien de la vitesse $\nabla \cdot \nabla \vec{u}$ sera plus grand dans les zones perturbées que dans les zones calmes.

L'équation dans son ensemble correspond en fait à la fameuse équation du mouvement de Newton : $\sum \vec{F} = m\vec{a}$ qui s'applique normalement sur une particule ponctuelle avec une certaine masse m. Il s'agit ici de son application aux fluides. Cette équation se lit : la somme des forces est égale à la masse multipliée par l'accélération. Dans le domaine de la simulation, nous cherchons en général à calculer les positions et les vitesses des éléments simulées, ainsi que leurs évolutions au cours du temps. Ainsi, l'équation se lit plus aisément comme ceci : le changement de vitesse est égal à la somme des forces divisée par la masse.

Dans le domaine des fluides, la masse est remplacée par la densité (densité = masse / volume).

Du côté du changement de vitesse, nous trouvons $\frac{\partial \vec{u}}{\partial t}$ qui est effectivement la variation de vitesse mais aussi $\vec{u} \cdot \nabla \vec{u}$ qui représente la partie de l'accélération qui est dispersée dans le sens du gradient de vitesse et ne contribue donc pas au changement de la vélocité en un point précis.

De l'autre côté, les différentes forces qui s'appliquent sont :

- la gravité et les forces externes : $\vec{g}$
- la pression : $-\frac{1}{\rho}\nabla p$ le gradient de pression crée une force depuis les zones de fortes pressions vers les zones de basses pressions qui dépend de la densité ρ du fluide.
- la viscosité cinétique : $v \nabla \cdot \nabla \vec{u}$ la différence entre la vitesse en un point et les vitesses moyennes de la zone crée un frottement qui entraîne les particules voisines et oblige les lignes de flux voisines à partager une partie de leur vitesse. C'est la résistance du fluide à la déformation.

Les dernières forces qui entrent en compte et dont nous n'avons pas parlé sont les collisions, avec le décor ou avec des éléments eux-mêmes en mouvement. Ces forces sont souvent celles qui perturbent le plus le fluide, qui par ailleurs tend à l'inertie et donc à la surface plate dans le

cas d'une surface d'eau à l'échelle d'une rivière. Les forces de collisions vont influencer la pression aux abords des obstacles, principalement lorsqu'ils sont en mouvement (personnages, véhicules...).

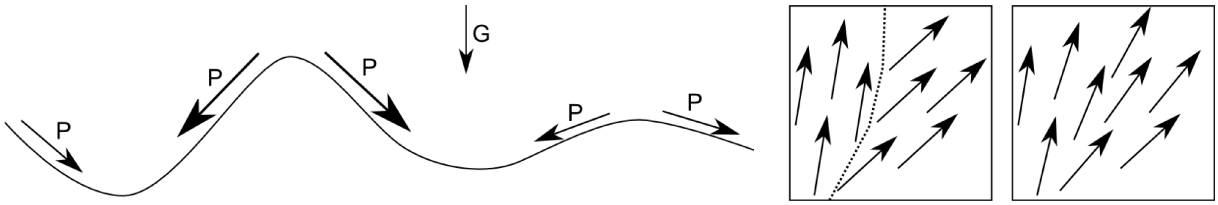


Figure IV.7: Les forces qui s'appliquent sur un liquide.
À gauche, la pression P qui s'applique des zones de fortes pressions vers les basses pressions, ainsi que la force globale de la gravité. À gauche, un champ vectoriel discontinu puis le même champ une fois la viscosité cinétique appliquée. Celle-ci s'oppose aux grandes différences de vitesses au même endroit.

La seconde ligne de notre équation de départ décrit la condition d'incompressibilité $\nabla \cdot \vec{u} = 0$ et se lit : la divergence de la vitesse est nulle. Concrètement, une divergence nulle signifie que le fluide est stable et qu'en tout point, la somme des vitesses qui partent est égale à la somme des vitesses qui arrivent. Ainsi, la substance constituant le fluide n'est jamais véritablement étirée, mais simplement transportée au gré des courants. Dans un fluide à divergence nulle, toutes les lignes de flux mènent quelque part. Si des particules sont en suspension sur le liquide, elles ne resteront jamais bloquées à un endroit du fluide. La non-divergence est un critère essentiel pour pouvoir appliquer correctement cette simplification des équations de Navier-Stokes, mais il suffit souvent en pratique de minimiser la divergence pour éviter les explosions de la simulation qui arrivent sans prise en compte de la condition d'incompressibilité. Si la divergence nulle n'est pas assurée, la simulation va avoir une tendance aux « feedbacks » et aux comportements chaotiques qui entrent en résonance. La simulation est alors complètement inutilisable. Les fluides réels sont plus ou moins compressibles, mais la plupart des effets liés à la compression (compression sonores, ondes de choc) sont trop rapides pour être modélisés et modifient très peu l'aspect visuel du fluide. Nous éliminons donc tous les phénomènes de compression afin de simplifier le modèle et de le rendre plus stable.

Une série d'articles très complète est disponible sur le site d'Intel[81] pour approfondir les différentes méthodes utilisées dans les jeux pour la simulation physique des fluides, et notamment des fluides en 3D.

81 Michael J. Gourlay, « Fluid Simulation for Video Games », 2010, http://software.intel.com/en-us/articles/fluid-simulation-for-video-games-part-1/.

3.2 Interprétations eulérienne et lagrangienne

L'interprétation des équations de Navier-Stokes peut se faire de deux manières principales : la méthode eulérienne et la méthode lagrangienne.

La méthode eulérienne considère le fluide comme un volume de valeurs. Chaque point du volume contient les valeurs du fluide (vélocité, pression, température, couleur…) et évolue au cours du temps, mais les points eux mêmes ne se déplacent pas. Nous pouvons toujours afficher des particules avec la méthode eulérienne, mais celles-ci utiliseront simplement les valeurs de la grille pour se déplacer, sans que leurs mouvements influencent la simulation.

La méthode lagrangienne considère le fluide comme un champ de particules individuelles qui bougent selon les lois du mouvement en s'influençant les unes les autres. Chaque particule porte ses propres valeurs de vitesses, de pression, de température, de couleur… La vélocité en un point de l'espace donné est ainsi la moyenne de la vitesse des particules proches.

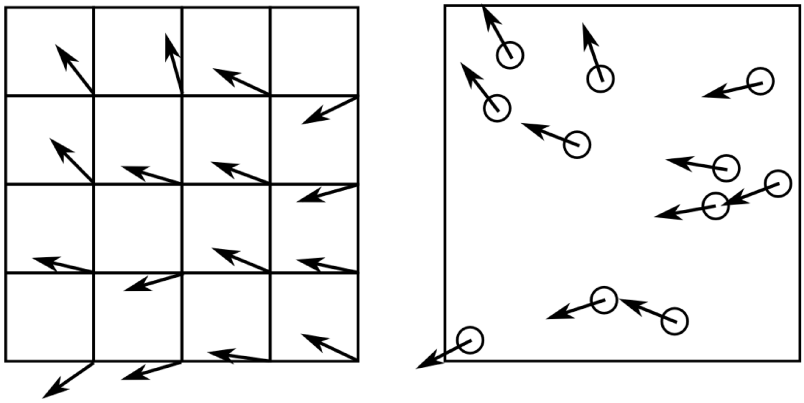


Figure IV.8: L'approche eulérienne (à gauche) et lagrangienne (à droite). L'approche eulérienne s'intéresse aux valeurs du fluide à des emplacements fixes généralement le long d'une grille, alors que l'approche lagrangienne suit les valeurs de particules en mouvement.

Concrètement, l'approche lagrangienne modélise le mouvement d'un fluide en déplaçant ses constituants (les particules) alors que l'approche eulérienne le modélise comme des transmissions de valeurs entre les cellules d'une grille fixe.

L'approche lagrangienne est plus intuitive, il s'agit de faire bouger directement les particules constituant le fluide. La méthode lagrangienne implique cependant plusieurs désavantages au niveau des calculs, notamment pour le gradient et l'opérateur laplacien. Même le simple calcul de la vélocité en un point de l'espace donnée devient complexe et nécessite de retrouver efficacement toutes les particules proches du point. Ainsi, les approches eulériennes sont plus aisées à mettre en place, surtout en temps réel. Le calcul de la dérivée partielle d'une valeur

est facile à approximer avec l'approche eulérienne, par une simple différence entre cases adjacentes.

L'approche eulérienne est en général limitée à un volume fixe de fluide, celui de la grille utilisée. Ainsi, le liquide ne peut pas s'échapper du volume prévu, ce qui limite souvent la simulation à une zone réduite et bien définie : un contenant. Au contraire, l'approche lagrangienne ne définit pas spécifiquement le volume de simulation. Les particules sont libres de se balader partout en fonction de leurs mouvements. En contrepartie, il est nécessaire de maintenir une certaine densité des particules pour obtenir une simulation correcte. Si les particules se dispersent trop, la simulation deviendra pauvre, sans détails. De nombreux algorithmes tentent ainsi de maintenir la densité des particules relativement constante en ajoutant de nouvelles particules et en retirant celles qui sont trop proches les unes des autres.

3.3 Contraintes d'un algorithme classique de fluide

Dans la plupart des cas, la simulation de fluide est un algorithme itératif qui décrit l'évolution du fluide à partir d'un état de départ et des forces qui s'appliquent à chaque subdivision discrète du temps. Ainsi, l'algorithme utilise une durée en tant qu'intervalle de simulation, appelé Δt , qui est souvent fixe afin d'améliorer la stabilité de la simulation.

Les valeurs du fluide sont stockées soit sous la forme d'une grille (point de vue eulérien) soit sous la forme de particules individuelles (point de vue lagrangien), et l'algorithme se charge de faire évoluer ces valeurs à partir de la situation à l'image actuelle pour simuler l'évolution au cours de l'intervalle de temps Δt et ainsi obtenir les valeurs à l'image suivante.

Nous allons nous intéresser ici à la méthode eulérienne, sous la forme d'une grille en deux dimensions. Cette grille est une représentation discrète d'un champ de valeurs considéré comme continu. Nous pouvons connaître l'intensité d'un champ en tout point par l'interpolation linéaire des quatre valeurs de la grille les plus proches. Comme nous utilisons une grille, l'espace couvert va être limité et le fluide ne pourra pas se balader n'importe où, mais restera confiné à l'espace couvert par la grille.

Une simulation de fluide est souvent formée d'un fluide de transport et d'un fluide transporté. Pour de la fumée, le fluide de transport sera l'air et le transporté sera une valeur représentant la densité de la fumée. Pour de l'encre plongée dans l'eau, le fluide transport sera l'eau et le transporté sera l'encre.

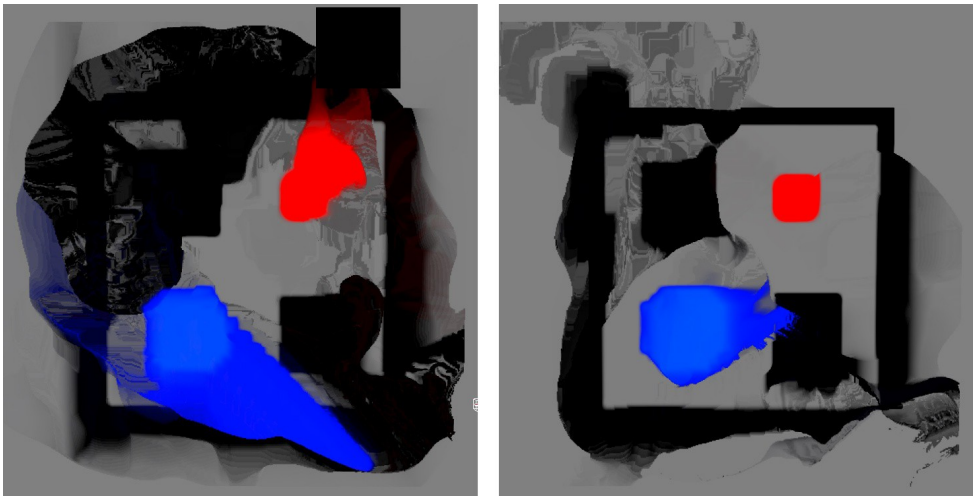


Figure IV.9: Simulation du transport d'une encre de couleur par de l'eau.

Une première grille de valeurs va contenir la vélocité du fluide transport, une autre va contenir son intensité (sa pression) et une dernière celle du fluide transporté[82]. Le procédé fondateur de la simulation est nommé : « l'advection ». C'est le fait de transporter une valeur selon la vélocité du fluide de transport. Ainsi, les valeurs du fluide de transport vont se modifier, se déplacer, en fonction de la vélocité, de même pour celles du fluide transporté. La vélocité du fluide de transport va, elle aussi, se déplacer en fonction de son propre courant, c'est le processus d'autoadvection.

Un second procédé indispensable va nous permettre de modifier la vélocité en fonction de la pression du fluide de transport. Pour connaître la pression à un endroit, il suffit d'utiliser la grille d'intensité du fluide de transport, qui représente la densité du fluide. S'il y a une différence entre deux cases adjacentes, cela signifie qu'il existe une force entre ces deux cases, dont l'intensité dépend de la différence entre les deux valeurs, le sens allant toujours du plus haut vers le plus bas (de la surpression vers la dépression). Cette force est donc ajoutée à la vélocité.

Il existe plusieurs autres étapes plus ou moins dispensables et plus ou moins justifiées physiquement que nous pouvons ajouter ici afin d'obtenir différents types de fluides. Ainsi il existe un calcul nommé « vorticity confinement » qui va chercher à calculer la différence de vélocité entre deux cases adjacentes afin d'ajouter une sorte de friction, qui va créer de petites volutes agréables à l'œil.

82 M. Harris, « Fast fluid dynamics simulation on the GPU », in *GPU gems*, vol. 1 (Addison Wesley, 2004), 637–665.

Bien sûr, il faut également calculer une friction globale, une déperdition, qui existe toujours quelque part et qui va mener le fluide vers un état plus stable et plus calme si rien ne vient le perturber.

Le fluide est également capable de transporter d'autres valeurs que des informations de couleur ou de quantité d'encre. Ainsi, la création de certains effets de combustion va nécessiter de représenter la température. Celle-ci, transportée au grée des courants d'air, va influencer l'écoulement du fluide transport (ici l'air), par des appels d'air des endroits les moins chauds. Nous pouvons également imaginer des explosions si la température est trop élevée en un point.

Si nous souhaitons simuler la formation de nuages, il est possible de prendre en compte une valeur d'humidité, qui va influencer l'évolution de la forme du nuage.

3.4 Le calcul de l'advection

Pour calculer l'advection du point de vue eulérien, il existe deux méthodes : l'advection avant (forward advection), et l'advection arrière (backward advection).

L'advection avant calcule pour chaque point de la grille l'endroit où il doit se déplacer (c'est-à-dire l'endroit où doit aboutir sa valeur), et ajoute à cet endroit la quantité appropriée. C'est une approche semi-lagrangienne, car les cellules de la grille sont considérées temporairement comme des particules dont nous cherchons à suivre le mouvement.

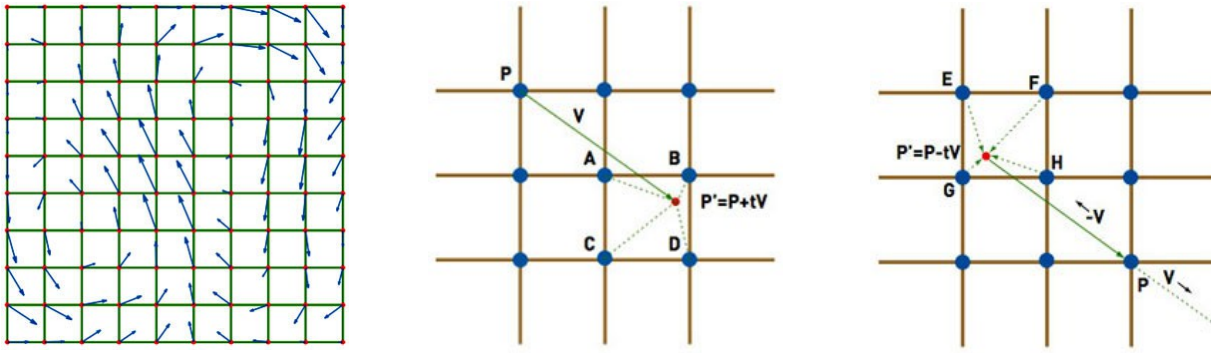


Figure IV.10: La grille de simulation et ses vecteurs de vélocité (à gauche), présentation de l'advection avant (au centre) et de l'advection arrière (à droite).
L'advection avant déplace la couleur de l'origine (P) vers la case d'arrivée (P') alors que l'advection arrière part de la case d'arrivée (P) et cherche la case d'origine (P').
Image gamasutra.com

L'advection arrière va faire le contraire, c'est-à-dire que pour chaque point de la grille, nous allons calculer l'endroit d'où provient la quantité qui devrait s'y ajouter. Cette approche est purement eulérienne.

La seconde manière de calculer l'advection est idéale pour une carte graphique, alors que la première lui pose plus de problèmes. En effet, avec l'advection arrière, il suffit, dans le pixel shader, de regarder la valeur de la texture de vélocité à la position d'un pixel, de l'utiliser pour remonter (donc dans le sens inverse de la vélocité) au point source estimé, et de faire un second appel à cet endroit. Cela nous donne la nouvelle valeur du pixel de départ. De plus, l'interpolation bilinéaire entre les quatre pixels alentour dans la texture de vélocité est effectué automatiquement lors de l'échantillonnage de la carte graphique. Cela nous permet de traiter la texture de vélocité comme un champ continu.

Au contraire, l'advection avant demande à inscrire le résultat dans un autre pixel que celui utilisé au départ, en fonction de la vélocité. Il faudrait même dans l'idéal inscrire le résultat réparti dans les quatre pixels les plus proches de la nouvelle position, afin de respecter le caractère continu du champ de vélocités sous-jacent.

Nous pourrions penser qu'il nous suffit d'utiliser l'advection inverse seule pour simuler l'advection du fluide. mais il se pose également le souci de l'incompressibilité du fluide. En utilisant l'advection inverse, plusieurs pixels peuvent se retrouver à utiliser le même point de source pour connaître leur nouvelle valeur, ce qui logiquement double la quantité de fluide présent. Intuitivement, nous comprenons bien que chacun des deux points de destination ne devrait utiliser que la moitié de la valeur de départ, mais les calculs sont effectués indépendamment et il est difficile de connaître au préalable le nombre de points de destinations. Le souci inverse existe également, car lorsqu’un pixel se retrouve avec une vélocité absolument nulle, il ne sera plus jamais modifié. L'advection arrière ne produira aucune modification, le pixel ira chercher sa nouvelle vélocité à sa propre position. Il est à noter que l'advection avant possède son propre lot d'artefacts, notamment par le fait que les pixels n'ont aucune garantie d'obtenir une nouvelle valeur à l'image suivante. En effet, des trous peuvent se former si les vitesses sont fortes. La nouvelle valeur d'un point dépend des autres pixels alentour qui peuvent potentiellement venir y transférer leurs valeurs. Tous ces problèmes sont liés à l'incompressibilité du fluide.

Lorsque nous utilisons l'advection arrière, il nous faut assurer l'incompressibilité du fluide par un autre moyen, qui se traduit par une sorte de dilution du fluide. Dans l'algorithme disponible

dans une démonstration de nVidia[83], cette incompressibilité était obtenue par des calculs très complexes et gourmands nécessitant notamment une intégration. Afin d'utiliser le parallélisme de la carte graphique au maximum, il existe un algorithme qui décompose cette intégration en plusieurs passes successives (intégration de Jacobi). Plus le nombre de passes est important, plus le résultat sera précis et réaliste. Pour avoir un bon résultat, le calcul nécessitait à peu près 80 passes, ce qui est très coûteux en ressources.

3.5 Un algorithme d'advection alternatif

En cherchant du côté des algorithmes de simulation, nous avons trouvé un algorithme utilisé sur le processeur principal et qui ne nécessite pas d'intégration[84]. Par contre, il demande l'utilisation conjointe de l'advection inverse et de l'advection avant. Nous avons donc tenté d'implémenter une version GPU (sur la carte graphique) de l'advection avant.

Depuis l'arrivée des shaders 3.0, il est possible de faire appel à une texture depuis le vertex shader. En utilisant cette texture, nous pouvons modifier l'endroit où nous allons positionner le sommet des triangles à dessiner. Si nous utilisons un rendu de type « point », nous pouvons dessiner un pixel pour chaque sommet et nous pouvons choisir, dans le vertex shader, l'endroit où nous allons écrire ces pixels. Cette technique rend donc bien possible l'advection avant sur la carte graphique.

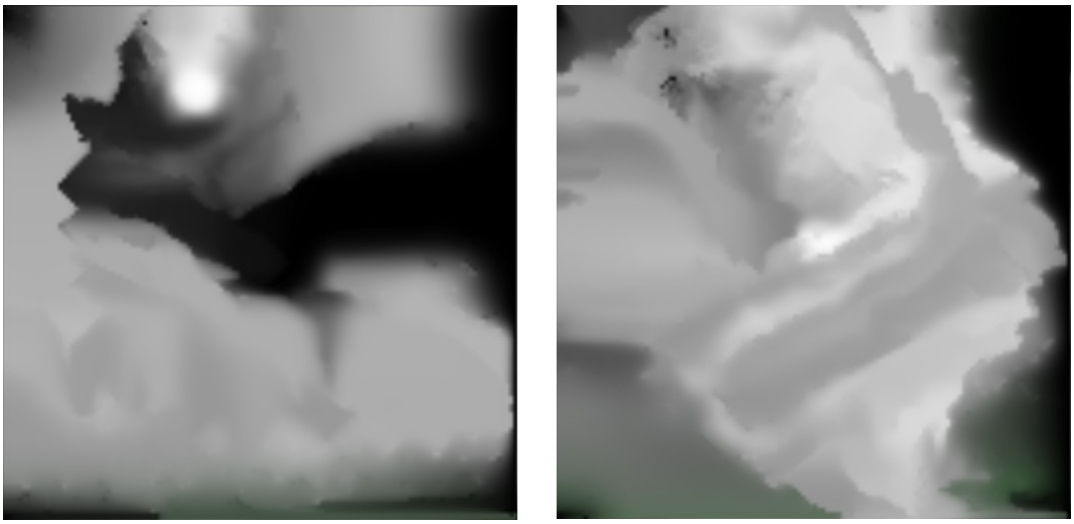


Figure IV.11: Utilisation alternée des advections avant et arrière sur la carte graphique.

83 K. Crane, I. Llamas, et S. Tariq, « Real-time simulation and rendering of 3d fluids », *GPU Gems* 3 (2007): 633–675.

84 Mick West, « Practical Fluid Dynamics », 2008, gamasutra.com.

Afin de disperser le résultat sur 4 pixels adjacents, nous utilisons un rendu de points de taille 2x2 pixels, avec une valeur de transparence pour les poids de l'interpolation, et nous accumulons le résultat dans la texture finale.

Le principe de l'algorithme est d'effectuer une advection avant suivi d'une advection arrière à chaque étape de la simulation. Chaque type d'advection va introduire son type d'erreur, et va compenser les erreurs de l'autre type. Les advections avant et arrière se complètent ainsi l'une l'autre. L'algorithme est fonctionnel, mais deviens très coûteux avec l'augmentation de la résolution (par exemple une résolution de 512x512 pixels est assez lente avec une carte graphique aux shaders 3.0).

En effet, s'il est très rapide d'afficher 512x512 pixels, il est beaucoup plus lent d'afficher 512x512 sommets. Tout d'abord, il faut stocker et calculer 512x512 sommets avec leur position et les transférer vers la carte graphique. L'affichage des 512x512 sommets est très lourd, et comme nous écrivons 4 pixels pour chacun, au final il s'agit d'afficher 512x512x4 pixels. La carte graphique est optimisée pour afficher beaucoup de pixels par triangle. Dans notre cas, chaque primitive n'affiche que quatre pixels. La carte graphique (modèle de shaders 3.0) est complètement inadaptée à ce degré de densité.

Avec DirectX 10, et l'ajout du « geometry shader », il est possible d'obtenir des temps de calcul plus raisonnables, puisque le stockage des sommets n'est plus obligatoire. Ils peuvent être directement produits par le geometry shader. Les « compute shaders » de DirectX 11 pourraient permettre d'obtenir des temps de calculs véritablement corrects avec cette méthode.

Au final, dans le cadre des consoles actuelles (PS3, X360), cette technique est viable pour de petites résolutions, mais explose de manière exponentielle avec de plus grandes résolutions.

Une application concrète des fluides en 2D sera présentée dans la partie sur le rendu expressif.

3.6 Les fluides en 3D

Les techniques présentées ici marchent exactement de la même manière lorsque nous passons à la simulation d'une grille à trois dimensions. Les calculs s'appliquent juste à un plus grand nombre de voisins. De 4 en deux dimensions, nous passons à 6 en trois dimensions. La vélocité est stockée sous la forme d'un vecteur en deux ou trois dimensions, mais les algorithmes sont fondamentalement les mêmes.

Le principal challenge apporté par les fluides en trois dimensions tient dans l'affichage du volume de fluide calculé. S'il est très facile d'afficher une grille en deux dimensions et ses couleurs, une grille en trois dimensions nécessite des techniques plus évolués, aptes à représenter des volumes.

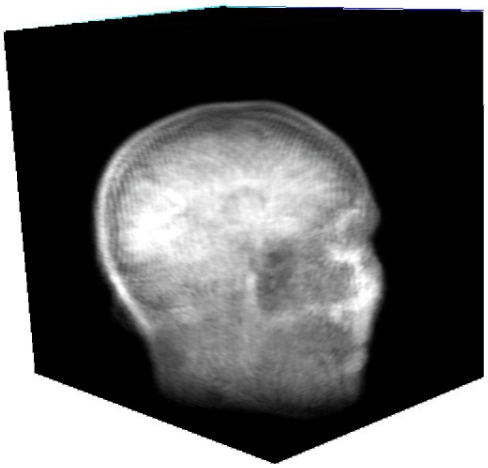


Figure IV.12: Rendu d'une texture volumétrique 3D

La plupart de ces techniques reposent sur le calcul de « slices » c'est a dire de coupes du volume qui vont être affichées les unes devant les autres. Le nombre de coupes calculées va produire une qualité plus ou moins grande. La solution idéale est difficile à définir et de nombreux artefacts graphiques désagréables apparaissent selon la manière d'afficher et de calculer ces slices[85].

Ces plans de coupes peuvent être positionnés de plusieurs manières et en nombres variables. Dans certains algorithmes, ils sont alignés avec la camera, ce qui amène à des soucis lors des rotations de celle-ci et nous voyons souvent des sortes de bandes sur les coupes. Les plans de coupes peuvent être tout simplement fixes, ici pas d'effet de bandes. Par contre, il nous faut alors croiser des plans de coupes dans chacun des trois axes, afin de couvrir tous les points de vue. Si nous regardons depuis un axe intermédiaire, il est difficile de mélanger convenablement les coupes des différents axes.

L'algorithme du « ray marching » est également une technique courante. La géométrie de rendu peut être très simple, un cube entourant le volume par exemple. Nous devons connaître la position d'entrée dans le volume du vecteur de la caméra, ainsi que sa position de sortie. Pour cela, nous pouvons calculer ces informations analytiquement ou bien afficher les faces arrière du cube du volume dans une première texture, en utilisant la position des pixels comme couleur. Le rendu des faces avant est alors en mesure de connaître la position de sortie du rayon en indexant cette première texture. Le calcul de la valeur d'un pixel s'effectue en accumulant la valeur de chaque cellule du volume 3D rencontrée le long du rayon formé par le point d'entrée et le point de sortie.

L'algorithme fonctionne bien, mais le nombre d'appels à la texture peut devenir très important. Nous pouvons utiliser le « dynamic branching » pour accumuler moins d'échantillons sur les bords du volume, où le rayon traverse peu de cellules. Le « dynamic branching » n'est pas une

85 Milan Ikits et al., « Volume Rendering Techniques », in *GPU gems*, vol. 1 (Addison Wesley, 2004).

technique très performante, mais le résultat obtenu est assez propre. La fréquence d'échantillonnage dans l'espace du volume est ainsi la même pour tous les rayons.

Encore une fois, l'utilisation des geometry shader, introduits part DirectX 10 permet de créer les géométries intermédiaires (les plans de coupes, ou slices) très efficacement.

3.7 La simulation Fluidz

3.7.a Les mac-grids

L'utilisation d'une seule grille de valeurs pour effectuer la simulation pose certains problèmes. En effet, le calcul de la pression, par exemple, devrait prendre place entre deux cellules. La vélocité est également mieux représentée comme une force entre deux cellules adjacentes, plutôt qu'une force liée à une seule cellule. Cette idée a mené à l'élaboration de la « MAC Grid » (« Marker-And-Cell » Grid). La MAC grid est un ensemble de deux grilles en décalage d'une demi-cellule. Ainsi, la pression est positionnée au centre de la cellule de la première grille, et les vélocités sont décomposées selon les axes X et Y. Chaque vélocité est positionnée entre deux cases de la première grille, formant ainsi une seconde grille de valeurs.

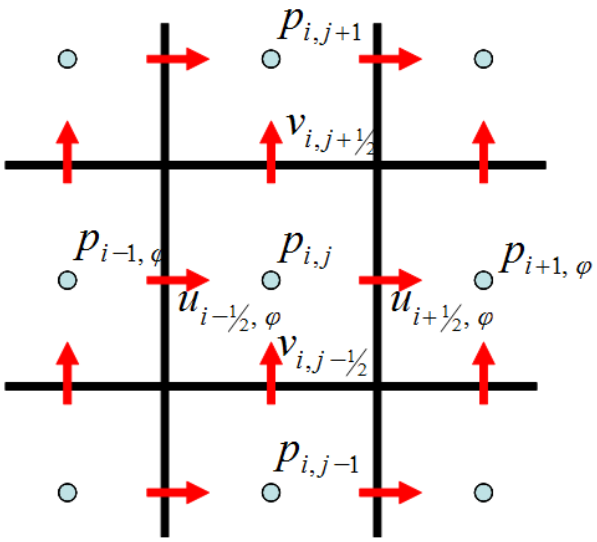


Figure IV.13: Arrangement des valeurs dans une MAC Grid.
La pression est localisée sur les ronds et l'advection sur les flèches rouges.
Image R. Bridson

La MAC-Grid améliore la qualité du fluide. L'advection n'est plus prise en compte comme un déplacement de quantités d'une cellule vers n'importe quelle autre, mais comme un échange de quantités entre cellules voisines. Ainsi, à chaque étape de simulation, les valeurs sont transférées selon les axes X et Y, dans la direction de la vélocité sur ces axes. Les flèches

rouges sur le schéma ci-dessous représentent ces transferts. Par contre, le calcul de la vélocité au centre d'une cellule nécessite de calculer une interpolation bilinéaire entre les quatre vélocités voisines, ce qui peut complexifier certains algorithmes.

3.7.b La décomposition spatiale en sous-simulations

Classiquement, une simulation de fluide eulérienne s'effectue dans un volume rectangulaire subdivisé par une grille. Nous nous intéressons essentiellement à la situation de canaux emplis d'eau et reliés entre eux. La taille de l'espace de simulation ne doit pas avoir des limites rectangulaires. Nous souhaitons de plus avoir un moyen de régler la précision de la simulation afin de simplifier les parties loin de la caméra, et au contraire nous concentrer sur les zones proches. Pour cela, la simulation est découpée en « bacs » rectangulaires d'eau qui se superposent à certains endroits. Les « bacs » sont ainsi reliés et échangent de l'eau entre eux. La résolution de chaque bac peut être modifiée en temps réel pour apporter plus de précision. Les influences entre les liquides des deux bacs sont calculées et appliquées. Les positions, hauteurs et orientations des bacs sont libres, bien que le liquide ait tendance à se stabiliser au bout d'un certain temps de simulation par les échanges entre les bacs et l'influence de la gravité. Le niveau de l'eau est alors le même dans tous les bacs. Afin de former des courants durables, nous pouvons placer des points d'entrées et des points d'extraction d'eau. Au bout de quelques instants, un courant se forme entre ces points en suivant les contours des obstacles.

Chaque « bac » de simulation possède sa propre grille de pression et de vélocité, et divers algorithmes vont assurer la propagation entre les bacs qui sont connectés. Une grille de collisions permet de modéliser grossièrement les contours des obstacles à l'intérieur d'un bac de fluide.

La résolution de la simulation de chaque bac peut être ajustée indépendamment et la continuité temporelle est assurée. Lorsque la caméra se déplace dans le décor, les bacs qui s'éloignent sont simplifiés en adoptant une résolution plus faible alors que les bacs qui sont plus proches deviennent plus précis. Le passage d'une certaine résolution à une autre a tendance à créer des mouvements indésirables dans le fluide, mais l'équipe responsable de la simulation « Fluidz » à DONTNOD a réussi à obtenir une simulation stable et sans artefacts visibles.

La simulation produit à la fois une grille de vélocité qui se modifie dynamiquement, mais également une grille de valeurs de pression, qui correspond concrètement à la quantité de matière en un point. Nous utilisons cette pression comme hauteur de la surface de fluide. Ainsi, le maillage qui va représenter la surface de l'eau est soulevé par les vagues de pression. Ces modifications de hauteur jouent également sur la normale de la surface et donc sur l'éclairage de l'eau. La texture de vélocité va, enfin, nous servir à manipuler une texture de vagues ce qui va être notre préoccupation principale dans la suite de ce chapitre.

4 Techniques classiques d'advection de textures

Une simulation de fluide est dite « physiquement réaliste » lorsqu'elle offre une réponse visuelle crédible aux interactions avec le joueur et avec les obstacles. Pour cela, elle doit posséder des qualités esthétiques et une diversité qui vont rendre le mouvement plaisant et complexe tout en étant crédible.

Ces qualités peuvent être apportées par des techniques d'advection de textures, soit pour modéliser les caractéristiques fines du fluide, soit pour simuler le mouvement de particules en suspension sur le fluide, tel que de l'écume, de la saleté ou des déchets. Ces éléments sont importants pour ajouter les petits mouvements d'eau qui ont lieu à la surface, que ce soit un faible clapotis ou les mouvements complexes à proximité d'obstacles ou encore le déplacement global de l'eau dans le sens du courant.

Avec les contraintes du temps réel, la simulation de fluide possède une limite dans la résolution de simulation qui empêche en général l'obtention de détails à petites échelles. Ces détails ont un rôle principalement visuel et ne sont pas essentiels à l'aspect physique de la simulation et à la simulation du mouvement global. Au-delà de l'apport visuel, ces détails participent tout de même à masquer les artefacts liés à la faible résolution de la simulation. Il s'agit en quelque sorte d'un bruit qui vise à perturber l'aspect du fluide afin de paraître moins artificiel.

Les détails à petites échelles servent également à renforcer l'impression de mouvement. Une simulation de fluide classique tend à idéaliser les mouvements du fluide. La dispersion des forces internes au fluide qui est liée aux approximations des calculs produit un résultat souvent lissé. Percevoir les mouvements et les différences de vitesses à l'intérieur du fluide est alors difficile. Avec l'introduction de détails qui seraient transportés par le fluide, le mouvement est directement perceptible, et les vitesses des différentes parties du fluide peuvent être déterminées.

L'utilisation d'une simulation à grande échelle avec une résolution relativement faible permet également un contrôle plus simple et prévisible sur le mouvement global du fluide. Une simulation qui opère à une échelle très fine est souvent source d'interactions inattendues qui dégénèrent en déstabilisant le mouvement à grande échelle. L'utilisation de deux niveaux d'échelle est donc un bon moyen de contrôler le mouvement global tout en ajoutant des petits détails grâce à une technique différente, l'advection de texture.

L'advection de texture désigne le fait de transporter des valeurs structurées le long de la simulation de fluide. La texture en elle même représente les caractéristiques « idéales » que nous souhaitons voir se répéter sur toute l'étendue du fluide. L'aspect fréquentiel d'un bruit est un exemple de caractéristiques que nous souhaitons pouvoir conserver lors des déplacements de la texture au cours du temps en fonction des mouvements du fluide.

Nous avons donc deux objectifs contradictoires qu'il s'agit d'équilibrer afin d'obtenir le meilleur résultat visuel. D'un côté, nous avons l'aspect de la texture que nous souhaitons conserver le plus possible. De l'autre, nous avons l'impact du mouvement du fluide que nous voulons percevoir, et qui doit donner l'impression d'entraîner la texture.

4.1 Advection avant/arrière d'une texture

L'advection d'une image peut se faire de deux manières différentes, qui correspond aux deux approches de la simulation de fluides que nous avons déjà mentionnée. L'approche semi-lagrangienne et l'approche eulérienne, également appelées respectivement « advection avant » et « advection arrière ». Il est à noter qu'une approche purement lagrangienne (à base de particules) produit des trous dans l'image advectée. Cependant, dans les deux cas, l'image est très rapidement méconnaissable à cause de la dispersion de la simulation. Chaque étape d'advection effectue un léger flou sur l'image lorsque le vecteur d'advection tombe entre plusieurs cellules, et donc répartie la quantité sur plusieurs pixels. Au contraire, nous souhaitons conserver au maximum les petits détails de la texture et notamment le bruit. Ce sont ces détails qui vont permettre de ressentir la vitesse du courant. Les détails à plus grande échelle n'ont pas besoin d'être dans la texture et proviendront de la simulation de fluide en elle-même. Nous devons donc trouver d'autres techniques d'advection de textures plus adaptées à la conservation des détails.

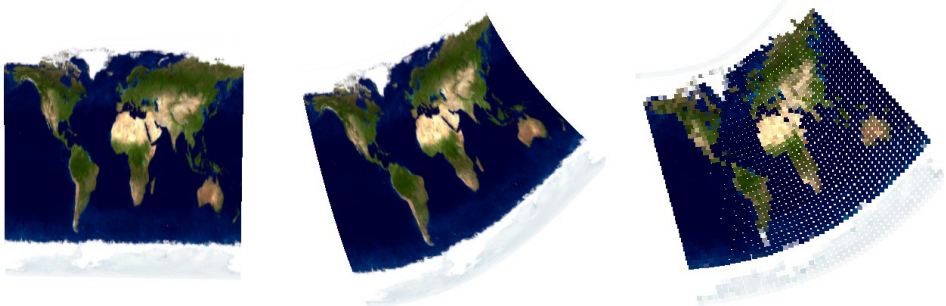


Figure IV.14: Advection avant et arrière d'une image.
À gauche, l'image originale et au centre la déformation de cette image par advection arrière.
À droite, l'image est déformation par advection avant ce qui laisse des trous.

4.2 Advection par distorsion des coordonnées de texture

Pour illustrer notre propos sur le dilemme entre le degré de distorsion de la texture et la qualité de la perception du mouvement, nous allons étudier la manière la plus simple d'envisager l'advection.

Il s'agit d'utiliser le vecteur de vélocité en chaque point pour distordre la texture en manipulant ses coordonnées de texture[86]. Ainsi, nous allons reculer dans le temps pour aller chercher une valeur de plus en plus éloignée en fonction du vecteur de vélocité.

Cette approche est très simple, mais pose plusieurs problèmes. D'abord, au fur et à mesure que le temps passe, la texture s'étire de plus en plus jusqu'à ne plus ressembler du tout à la texture originale. Ensuite, la distorsion se fait en ligne droite selon la vélocité locale au lieu de suivre une courbe en fonction des variations de vélocité passées. De ce fait, l'étirement rend la texture illisible encore plus rapidement, surtout dans les zones à la vélocité très contrastée.

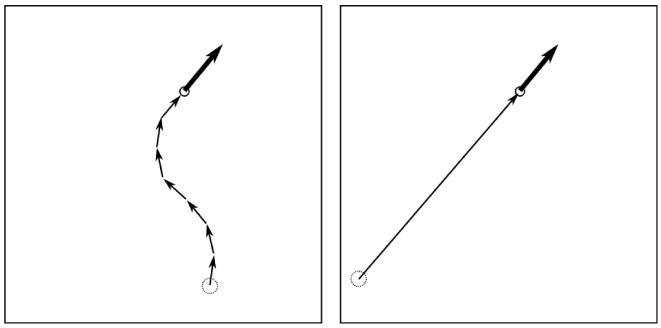


Figure IV.15: La distorsion exacte et l'approximation en ligne droite.
À gauche, la distorsion d'un point est effectué en suivant le champ de vélocité. À droite, seule la vélocité courante est utilisée, donnant lieu à une distorsion simpliste et peu crédible.

La solution la plus simple pour limiter les déformations est d'utiliser plusieurs couches de textures qui se distordent alternativement, puis reviennent à leur état initial. Les différentes couches sont mélangées afin d'utiliser majoritairement celle avec le moins d'étirements.

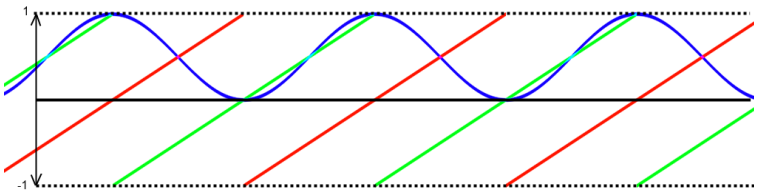


Figure IV.16: Mélange temporel de deux couches d'advection.
Facteur de déformation de la première couche (rouge) et de la deuxième couche (vert).
L'opacité de la couche rouge est en bleu, c'est également le facteur de mélange entre les deux couches.

86 Nelson Max et Barry Becker, « Flow visualization using moving textures », *Proceedings of the ICASW/LaRC Symposium on Visualizing Time-Varying Data* (s. d.): 1995.

Avec un système constitué de deux couches, seule la première couche est visible au départ, sans aucune distorsion. Puis la distorsion commence et la première couche s'étire tout en se mélangeant progressivement à la seconde couche, invisible au départ, qui est déjà étirée dans le sens opposé au mouvement, puis reviens au fur et à mesure vers son état neutre, sans étirement, en devenant de plus en plus visible. Ainsi, l'alternance entre les deux couches permet de masquer le bouclage et de limiter l'étirement.

Cette approche a néanmoins de sérieuses limites. La première limite concerne la différence de qualité visuelle entre une texture non étirée et le mélange de deux textures étirées. Non seulement les textures étirées sont moins belles, mais le mélange entre deux textures différentes crée également une image floue et avec moins de contraste et de détails.

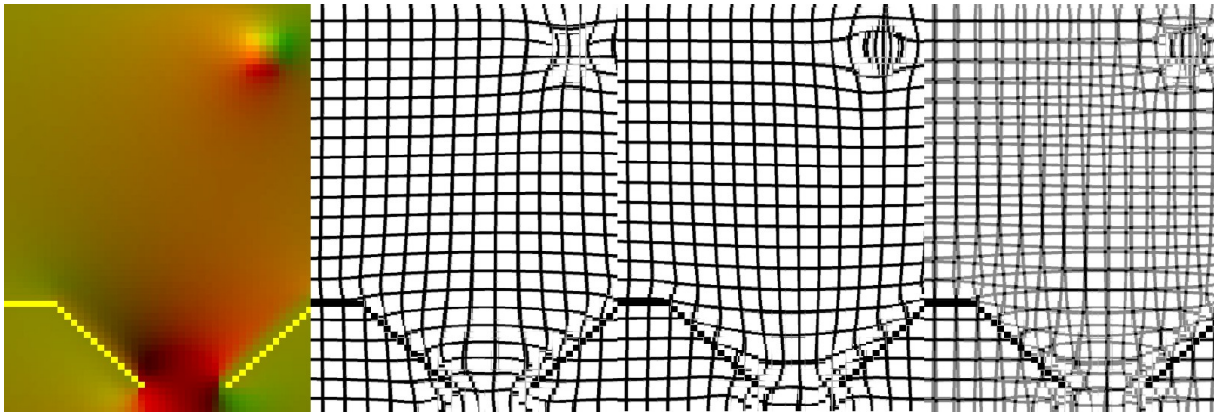


Figure IV.17: L'advection par mélange de deux couches de déformation des textures. De gauche à droite : texture de vélocité, advection inverse d'une image de grille, advection vers l'avant de la même image, mélange à 50 % des deux couches d'advection.

La seconde limite concerne le choix de la quantité d'étirement et de la durée de bouclage. Ces deux choix sont tous les deux liés à la vitesse moyenne du fluide et forment un équilibre. Pour une vitesse donnée du fluide, l'augmentation de la durée de bouclage provoquera une augmentation de l'étirement. Un autre effet désagréable est lié à la durée d'un cycle de transfert d'une texture à l'autre dans les zones avec une faible vitesse. En effet dans ces zones, au lieu de percevoir un mouvement, nous percevons plutôt la disparition d'une texture et l'apparition d'une autre. Étant donné que l'étirement dépend de la vitesse (qui est pratiquement nulle dans ces régions), les deux textures vont très peu s'étirer et nous percevrons un simple mélange progressif au lieu d'un mouvement. Nous devrions modifier la durée de transition afin de laisser la texture s'étirer plus fortement avant d'effectuer une transition. Cependant, l'algorithme nous oblige à utiliser une durée de transition fixe, la même pour toute la texture.

Dans les zones à fort courant, c'est le contraire qui se passe, c'est-à-dire que l'étirement devient trop fort bien avant que la transition se fasse avec l'autre couche.

Ainsi, le choix de la durée de bouclage idéale est directement lié à la vitesse du fluide et à l'étirement que nous sommes en mesure de tolérer. Malheureusement, il n'est pas possible de choisir une durée de bouclage différente par pixel, car le rythme des cycles de bouclage serait alors très vite en décalage d'un pixel au suivant et l'impression de mouvement général serait remplacée par des pixels clignotants chacun à leur rythme.

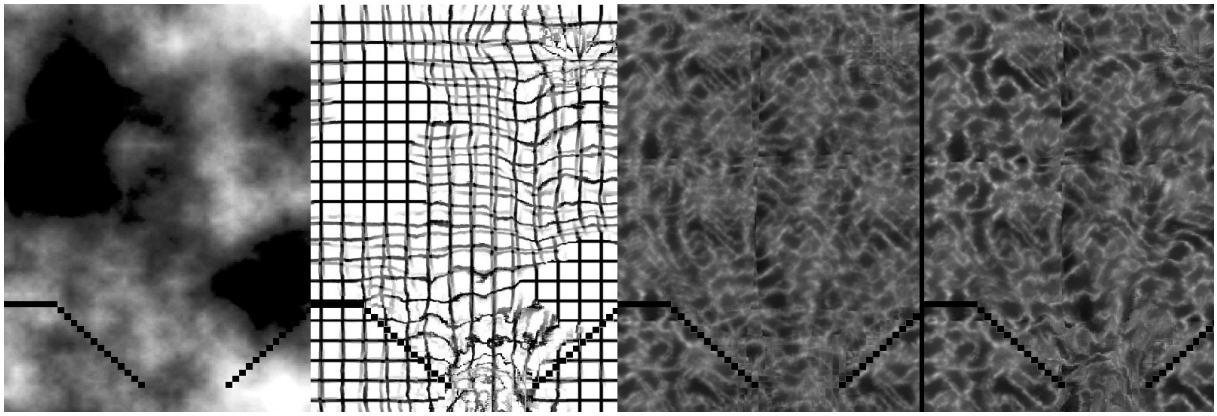


Figure IV.18: Utilisation d'un décalage de phase dans le mélange des deux couches. De gauche à droite :texture de décalage de phase, application du décalage sur l'advection de la grille, application sans décalage sur une image de vagues, application avec décalage. L'utilisation de la texture de décalage répartie effectivement les distorsions et les transitions sur toute la durée d'une boucle d'advection.

Avec une durée de bouclage fixe, une sensation de pulsation est très perceptible entre les deux états possibles (une seule image contrastée ou deux images mélangées). Une solution partielle est d'utiliser une autre texture, par exemple de bruit à faible fréquence, dont la valeur sert à décaler doucement les rythmes de bouclage des pixels. Si la fréquence de la texture de bruit n'est pas trop élevée, les pixels adjacents resteront relativement ensemble. La sensation de pulsation générale disparaît au profit de pulsations localisées beaucoup plus discrètes. Par contre, une distorsion supplémentaire s'ajoute à cause de la différence de rythme de bouclage qui provoque des déplacements de la texture différents entre des pixels proches. Cette distorsion supplémentaire perturbe légèrement le mouvement. La différence dans le rythme de bouclage produit des zones en avance de phase et d'autres en retard. Un mouvement continu sur une grande zone pourra donc subir des effets d'accélérations et de décélérations en fonction des changements de la valeur de décalage.

4.2.a La mise à jour dynamique de la vélocité

Jusqu'ici, nous n'avons pas évoqué un aspect pourtant essentiel : la possibilité de mettre à jour les valeurs de vélocité du fluide. En effet, nous souhaitons utiliser une simulation de fluide pour créer et faire évoluer une texture de vélocité que nous utiliserons ensuite dans le processus d'advection de textures.

Dans notre schéma de distorsion de texture, le changement des vélocités n'est pas possible à tout moment. En effet, une fois la texture étirée, le changement de vélocité va changer la direction de l'étirement de manière brutale d'une image à l'autre, donnant l'impression que le liquide tourne brutalement dans un sens alors qu'il est simplement en train de changer de direction. Si chaque couche de texture possède sa propre texture de distorsion, nous pouvons utiliser le moment ou la couche concernée est invisible pour en modifier la texture de distorsion. Ainsi, chaque couche est périodiquement remise à jour, au moment où elle est invisible. Ce schéma fonctionne et si la vélocité est relativement uniforme et change doucement, le résultat est assez crédible. La technique la plus simple est alors de mettre en place un groupe de trois textures de vélocité. Ainsi, une de ces textures est mise à jour en temps réel par la simulation physique, et les deux autres sont des captures à un moment donné de cette simulation, chacune associée à une couche.

En plus de mettre à jour la texture de vélocité, nous pouvons également modifier la texture qui subit l'advection, au moment précis où une couche est invisible. La manière la plus simple et efficace de modifier la texture est de tirer au hasard une nouvelle valeur de décalage dans les coordonnées de textures. Ainsi, chaque couche peut effectuer l'advection de la même texture de base, mais avec un décalage de coordonnée différent et aléatoire qui change régulièrement.

Cette technique d'advection est très pratique et simple, surtout grâce à la carte graphique qui peut gérer facilement tous les calculs. Il est même possible de se passer de toute simulation en temps réel et d'utiliser des champs de vélocités précalculées sous la forme de textures. Le précalcul est effectué en fonction des décors et des collisions afin d'obtenir des fluides qui coulent autour des obstacles. C'est le cas chez Valve pour le jeu « Portal 2 »[87] ou les textures de vélocité ont été calculées dans le logiciel d'effets spéciaux précalculés « Houdini », en introduisant des sortes de tourbillons perturbateurs placés au hasard et en calculant les rebonds du fluide sur les murs. Il faut que tous les éléments de la simulation restent stables au cours du calcul (les décors ne bougent pas, les tourbillons tournent à vitesse constante) et

87 Alex Vlachos, « Water Flow in Portal 2 », in *Siggraph* (Valve, 2010).

laisser la simulation tourner un certain temps jusqu'à ce que la vélocité se stabilise. Une fois ces textures calculées, la création d'un shader d'advection pour la carte graphique est très simple et peu coûteuse en ressource. Le résultat n'est cependant crédible que pour des vitesses moyennes, ni trop basses ni trop hautes.

5 L'advection de texture par le biais de particules

Le but principal de l'advection de texture est d'aider à la compréhension du mouvement d'un fluide. Ainsi, une manière d'envisager l'advection de texture est d'imaginer que la surface du fluide est recouverte de petites particules en suspension qui se déplacent au grès du courant. La distribution de ces particules est essentiellement aléatoire, et nous allons étudier des techniques qui calculent l'advection d'une texture de bruit par une simulation de groupes de particules. Nous allons parler essentiellement de deux publications, la première est de l'INRIA[88] et la seconde est le mémoire de Master de Tim Burrel[89] et la publication associée[90].

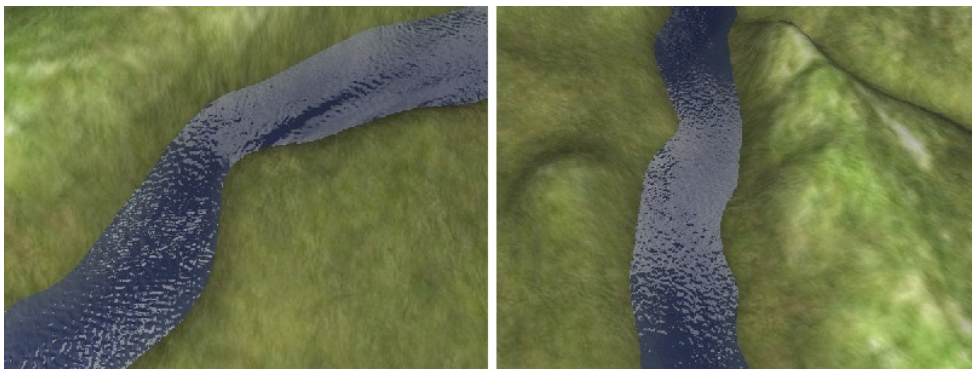


Figure IV.19: L'advection de textures dans un fluide, par Tim Burrel

L'idée principale de ces techniques est de distribuer des particules régulièrement sur la surface du fluide, puis de déplacer ces particules en fonction du mouvement du fluide situé en dessous. Rappelons que les particules sont des objets ponctuels, définis par leur position, leur vélocité et divers paramètres personnalisables. Parmi ces paramètres, les deux publications dont nous parlons ici utilisent un attribut de coordonnée de texture. Cette coordonnée servira à indexer une grande texture de bruit ou de vague, à une position différente pour chaque particule. C'est cette opération qui produira des particules d'aspects différents, bien qu'issus de la même texture de base.

La première étape de l'algorithme est la distribution des points sur la surface. La publication de l'INRIA nous indique qu'ils utilisent une distribution de Poisson afin d'obtenir des particules dispersées, mais pas trop, en assurant que l'écartement entre deux particules

88 Q. Yu et al., « Scalable real-time animation of rivers », in *Computer Graphics Forum*, vol. 28, 2009, 239–248.

89 Tim Burrell, « Advected River Textures », *Dalhousie University (Canada)*, Master (2008).

90 Tim Burrell, Dirk Arnold, et Stephen Brooks, « Advected river textures », *Comput. Animat. Virtual Worlds* 20, nº 2-3 (juin 2009): 163–173.

voisines est compris dans des limites définies. Le point important est d'assurer que chaque point de la surface est couvert par une particule, voir même par plusieurs. Ils utilisent également une distribution placée dans l'espace-écran. Ainsi, les particules sont placées afin de maintenir une distance entre elles constante vue depuis la caméra. Cette méthode permet d'assurer une adaptation du niveau de détail. Les parties du fluide loin de la caméra recevront moins de particules parce que leurs surfaces à l'écran sont plus petites. La taille des particules est également adaptée afin de maintenir une taille constante qu'elle que soit la distance de la caméra.

Une fois les particules placées, elles vont se déplacer à chaque image en allant chercher la vélocité du fluide à leurs positions dans la simulation sous-jacente. La simulation peut utiliser divers types de fluides, notamment des fluides paramétriques ou des fluides sous forme de grilles. La seule contrainte est de pouvoir connaître la vélocité en tout endroit afin de pouvoir mettre en mouvement les particules.

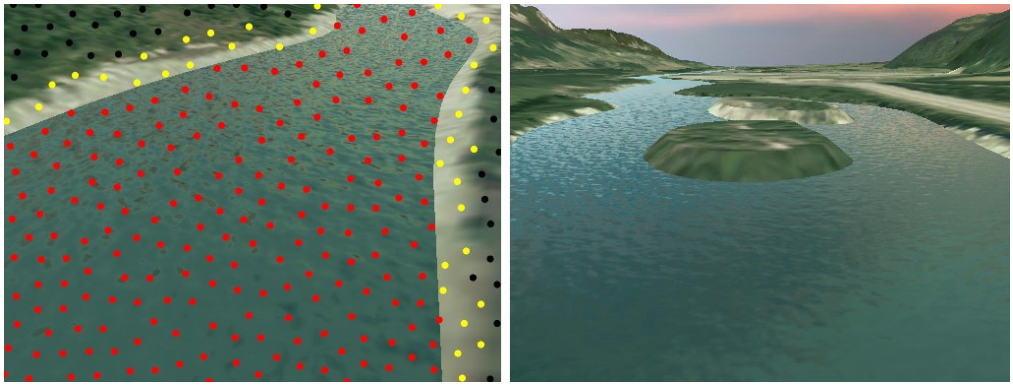


Figure IV.20: L'algorithme de l'INRIA.
Distribution de Poisson des particules en espace-écran (gauche) et résultat visuel en temps réel (droite).

Les particules se déplacent donc en fonction du courant, et commencent à se regrouper par paquets au grès du fluide, laissant des portions de la surface vide, sans particules. Pour contrer ce phénomène, de nouvelles particules sont constamment ajoutées et d'autres sont retirées. Dans le cas de Tim Burrel, les particules ont une durée de vie, qui est assez courte. Elles ne restent pas longtemps à l'écran, puis disparaissent. En contrepartie, de nouvelles particules sont constamment ajoutées sur la surface du fluide. Les durées de vies des particules sont décalées dans le temps afin de distribuer uniformément les disparitions et les apparitions. Dans la technique de l'INRIA, en revanche, les particules n'ont pas de durée de vie fixe, mais les règles de la distribution de Poisson sont vérifiées à chaque image. Si une particule ne satisfait plus à la distribution, car la concentration de particules est trop importante, celle-ci va

être éliminée. Au contraire, les espaces qui menacent de devenir vides suite à un étalement des particules vont se remplir par l'ajout de nouvelles particules dont les positions sont compatibles avec la distribution de Poisson. La technique de l'INRIA est donc meilleure dans le sens ou les particules restent les mêmes plus longtemps et les détails des textures sont ainsi mieux conservés dans le temps, rendant le fluide plus cohérent.

Les apparitions et disparitions de particules sont progressives dans les deux techniques. Il s'agit d'avoir un facteur d'opacité qui augmente doucement au cours de l'apparition et diminue à la disparition. L'obtention de la texture finale nécessite de mélanger toutes les particules entre elles. Une simple somme pondérée par l'inverse de la distance au centre de la particule permet d'effectuer une moyenne qui privilégie les valeurs de la particule la plus proche.

La texture utilisée comme motif pour les particules est une texture animée dans le cas de Tim Burrel, c'est-à-dire une série d'images qui représente un mouvement répétitif de fluide. Le motif peut être simplement un bruit d'assez haute fréquence, mais peut également suivre des formes précises, telles que des vagues. Nous pouvons également utiliser les valeurs contenues dans ces textures comme des paramètres d'un modèle de vague, tel que celui de Tessendorf. Les vagues peuvent ainsi être sculptées plus finement.

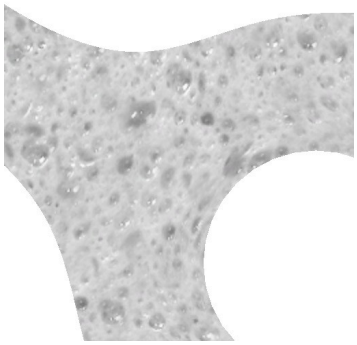


Figure IV.21: Advection d'une texture détaillée (INRIA).

Le rendu final se fait soit par l'affichage direct de chaque particule sous la forme d'un plan avec de la transparence (Tim Burrel), soit par la création d'une grille de redirection, contenant dans chaque cellule la liste des particules qui l'influencent (INRIA). Le pixel shader peut alors accéder à cette liste et effectuer correctement le mélange. Nous pouvons comparer ces deux techniques respectivement au deferred lighting et au foward Tile-based lighting qui existent dans le domaine de l'éclairage.

Ces deux techniques sont particulièrement intéressantes pour leur gestion de l'advection de texture. Le choix des particules permet une prise en compte aisée de la vélocité du fluide. Les problèmes d'étirements de texture ne sont plus du tout présents. Par contre, le maintien d'une liste de particules stables sur l'espace-écran est assez complexe. Le rendu en lui même est également assez cher. Le rassemblement des textures des différentes particules est coûteux pour la carte graphique. L'équipe de l'INRIA a également publié une étude[91] sur la préservation des caractéristiques spectrales de la texture de bruit lors de l'advection liée au fluide.

91 Qizhi Yu et al., « Spectrum-preserving texture advection for animated fluids », *INRIA*, 2009.

6 Techniques personnelles d'advection de textures

6.1 Advection séparée en zones arbitraires

6.1.a Séparation en zones de même vélocité

La séparation en zones est l'idée initiale qui nous a menés sur la voie de l'algorithme final présenté dans cette étude. En effet, nous avons vu qu'il n'était pas possible d'attribuer une vitesse de déplacement différente à chaque pixel, car cela crée des étirements entre les pixels que nous devons masquer. Habituellement, c'est une alternance temporelle entre plusieurs couches en déplacement qui permet de masquer l'étirement avant qu'il ne devienne trop important. La durée d'un cycle d'alternance ne peut pas être différente par pixel, car la cohérence de la texture d'origine disparaît très vite au fur et à mesure des décalages de phases entre pixels voisins. Par contre, des zones homogènes peuvent être utilisées, chacune possédant une vitesse d'étirement différente. Les seuls artefacts visibles se produiront aux intersections entre zones.

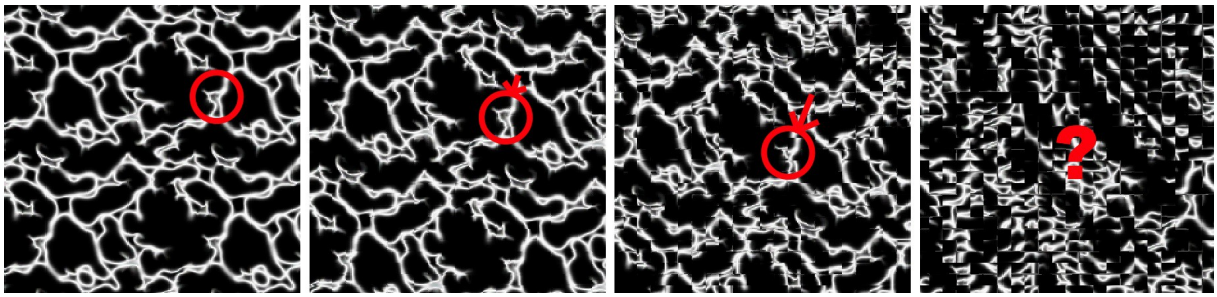


Figure IV.22: Décalages de phases au fur et à mesure de l'advection d'une texture.
La direction d'advection est différente pour chaque pixel. Au fur et à mesure que l'étirement se prolonge, chaque pixel va se décaler par rapport aux autres, rendant la texture incompréhensible. De gauche à droite, la texture est de plus en plus déformée par l'advection.

Ainsi, la première étape est de découper la texture de vélocité en différentes zones présentant des vitesses similaires. Ensuite, une vitesse moyenne est calculée pour chaque zone, puis réassignée aux pixels de la zone concernée. Le même processus de déplacement de textures est alors calculé en se basant sur le temps passé et la vélocité. La texture ne subit pas d'étirements comme précédemment, à part aux frontières entre deux zones.

Ces cassures entre les zones sont disgracieuses, mais peuvent être masquées par une seconde couche d'advection de textures à base de zones séparées différemment. Cette couche présentera également des bords entre ses propres zones. La surface visible de la seconde

couche est constituée uniquement des frontières entre les zones de la première couche. Ainsi, le nombre de frontières de la seconde couche est moins grand. Une troisième couche est alors nécessaire pour masquer les frontières de la seconde couche. Chaque couche va ainsi masquer les bords de la couche précédente, avec un mélange spatial progressif entre les couches.

6.1.b Le théorème des quatre couleurs

Le nombre de zones minimum qu'il est nécessaire de mélanger pour assurer un parfait masquage de toutes les bordures semble être un problème théorique complexe qui se rapproche du théorème des quatre couleurs. Le théorème des quatre couleurs prouve qu'il est possible de colorer tout plan découpé en régions de sorte que deux régions adjacentes reçoivent deux couleurs différentes en utilisant au plus quatre couleurs en tout.

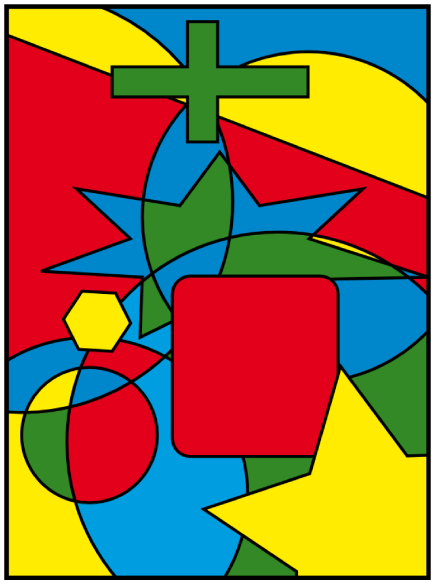


Figure IV.23: Illustration du thèorème des quatre couleurs. Image Wikipedia (Inductiveload)

Ce théorème, conjecturé en 1852, n'a été prouvé qu'en 1976 et a été le premier théorème majeur nécessitant l'utilisation de l'informatique pour sa démonstration, ce qui indique la difficulté de la preuve formelle. La majorité des cartes classiques peuvent cependant être colorées avec seulement trois couleurs et seuls quelques cas particuliers ne fonctionnent pas et nécessitent une quatrième couleur.

L'analogie avec notre problème est assez claire si nous imaginons que chaque couche représente une couleur. Si nous découpons la texture de vélocité en zones, nous pouvons assigner chaque zone à l'une des quatre couches (donc couleur) sans que deux zones d'une même couche ne partagent un bord. Chaque couche étend ensuite ses zones dans les endroits masqués par les autres couches, le mélange se fera ensuite linéairement en fonction de l'éloignement avec le bord original de la zone. Dans le théorème des quatre couleurs, des zones qui se touchent seulement en un point, c'est à dire sans partager un côté, peuvent avoir la même couleur. Dans notre cas, nous souhaitons que le moindre contact soit masqué et donc fasse l'objet d'une nouvelle zone dans l'une des couches, même si le contact est un point et non un côté. Le nombre de couleurs nécessaire si deux zones doivent avoir des couleurs différentes si elles partagent un point n'est pas limité. Il est au moins aussi grand que le

nombre de zones qui peuvent se rencontrer en un point, et ce nombre peut être arbitrairement grand.

Dans la pratique, la carte exacte des régions n'est pas prédéterminée dans notre cas, et nous pouvons donc tricher facilement en générant des zones qui seront adaptées à l'utilisation d'un petit nombre de couches. Chaque couche représente un coût de calcul en plus, nous souhaitons donc nous limiter à trois couches seulement.

6.1.c Découpage des zones

Nous avons fait le choix de proposer un outil de découpage manuel des zones plutôt que de chercher un algorithme entièrement automatisé. Notre outil permet de placer simplement des points aux endroits où nous pensons qu'une zone est nécessaire, et l'algorithme se charge d'étendre ces zones par élargissement progressif à partir du point initial, jusqu'à ce que toute la carte appartienne à l'une ou l'autre des zones. Cette première découpe représente la première couche. La seconde couche cherche à masquer les bords de la première, nous commençons donc par créer un masque qui détecte les bords entre les zones de la première couche. Ces bords sont ensuite épaissis d'une taille réglable pour créer la seconde couche. Au court du processus de création de ces bords, la valeur de mélange entre la couche 2 et la couche 1 est calculée, c'est la distance au bord, c'est-à-dire un dégradé du centre du bord vers l'extérieur. Encore une fois, nous utilisons un processus manuel pour découper la seconde couche, constituée des bords de la première. Nous plaçons des points et la même technique d'expansion est utilisée pour créer les zones de la seconde couche. Pour la troisième couche, le processus est répété une dernière fois ce qui suffit en général, car chaque bord entre la seconde et la troisième peut être masqué avec une seule zone.

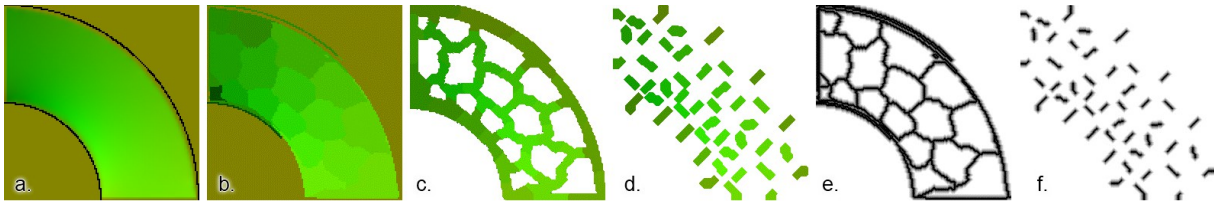


Figure IV.24: Exemple de découpage de la vélocité.
a. vélocité originale, b. première couche, c. seconde couche, d. troisième couche, e. opacité de la seconde couche, f. opacité de la troisième couche

La création d'un processus complètement automatique de découpage des zones est possible. Une technique très simple consiste à générer automatiquement des points suivant une distribution de Poisson, qui assure une répartition homogène des points et une distance

minimum et maximum entre eux. Les points servent ensuite dans l'algorithme déjà exposé, en remplacement des points placés manuellement. Dans l'idéal, l'emplacement des points devrait correspondre à des zones de vitesses homogènes. Il serait également intéressant d'augmenter le nombre de points aux endroits à forte variation de vitesse afin de mieux capturer les petits détails.

6.1.d Équilibre entre taille des zones et vitesses du fluide

L'advection par zone est une technique innovante que nous avons créée dans le but de produire une impression de courant convaincante. Cependant, il existe un équilibre entre la vitesse du fluide et la taille des zones. Lorsque la vitesse du fluide augmente, la taille de la zone doit également augmenter afin de permettre de suivre le déplacement des caractéristiques reconnaissables de la texture advectée pendant une durée conséquente. En effet, si la zone est trop petite, un déplacement rapide ne sera pas perceptible, mais s'apparentera plus à du bruit, à un changement aléatoire de l'image affichée. Si la vitesse est plus faible, le mouvement sera perceptible même dans les zones plus petites, ce qui nous permet de découper plus finement le plan de fluide. Le pire cas est celui de mouvements rapides et fortement perturbés, comme des tourbillons. Les mouvements de rotations sont d'ailleurs problématiques, car nous cherchons à les représenter par des zones en translation linéaire. Cependant, si la rotation s'étend sur plusieurs zones, le mélange de celles-ci offrira une impression de rotation suffisante. La possibilité de placer des points plus ou moins espacés selon la vélocité moyenne est intéressante afin d'adapter au mieux la fréquence d'échantillonnage. Les zones fortement perturbées sont subdivisées plus finement, quitte à perdre la cohérence du mouvement, alors que les zones plus homogènes sont divisées en grandes zones pour obtenir une bonne impression de mouvement.

6.1.e Mise à jour dynamique

Encore une fois, nous avons fait l'impasse sur la mise à jour dynamique de la texture de vélocités. En effet, la technique est adaptée pour représenter une texture de vélocité statique, dont les zones sont précalculées une fois pour toutes. Néanmoins, le but ultime reste d'utiliser la vélocité mise à jour en temps réel par la simulation de fluide.

L'opération de zonage peut être faite en temps réel en utilisant un algorithme simple de placement de points et en calculant les zones et la vitesse moyenne sur la carte graphique pour gagner en performance. Cela pose par contre des problèmes de continuité. En effet, le

placement des points représentant chaque zone doit être continu dans le temps, de même que la création des zones autour de chaque point. Or, la texture de vélocité est complètement libre et il est donc très difficile de créer des zones de même vitesse tout en assurant une continuité temporelle. En effet, l'ajout ou la disparition de zones résulterait forcément en un changement visuel majeur d'une image à la suivante.

Deux possibilités sont alors envisageables. Tout d'abord celle d'utiliser un nombre fixe de points organisés selon une structure également fixe. Ainsi, seule la création des zones à partir de ces points doit être continue. La seconde possibilité est de permettre le mouvement des points au cours du temps en utilisant une technique à base de particules. Nous avons choisi d'utiliser une structure fixe, les techniques à base de particules ayant déjà été très étudiées. La question de la meilleure structure de répartition des points s'est alors posée.

6.2 Advection suivant une grille carrée régulière

6.2.a Découpage en zones carrées

La texture de vélocité calculée par la simulation de fluide se présente sous la forme d'une grille de valeurs, il est donc naturel d'utiliser directement cette grille comme structure des zones. Ainsi, chaque cellule sera associée à une zone d'advection de texture. Tous les points de la grille sont exploités directement par une zone. La résolution de la simulation doit être assez faible, car chaque pixel formera une zone indépendante. Chacune de ces zones doit être assez grande pour que nous puissions y percevoir un mouvement de texture.

Pour exploiter la structure en grille, nous utilisons directement des carrés comme première couche. Nous pourrions assigner une cellule sur deux à chaque couche afin de n'utiliser au total que deux couches. Les zones d'une couche ne seraient en contact que par leurs coins. Malheureusement, les contacts aux coins posent problème puisqu'ils ne laissent pas la place pour effectuer une transition douce. En clair, il nous faut pouvoir disposer d'un espace libre entre deux zones où la couche est invisible et masque les discontinuités.

La solution simple serait d'utiliser quatre couches. Ainsi deux zones adjacentes même par les coins seraient dans des couches différentes. Pour des raisons de performance, nous allons diviser la grille en trois couches seulement.

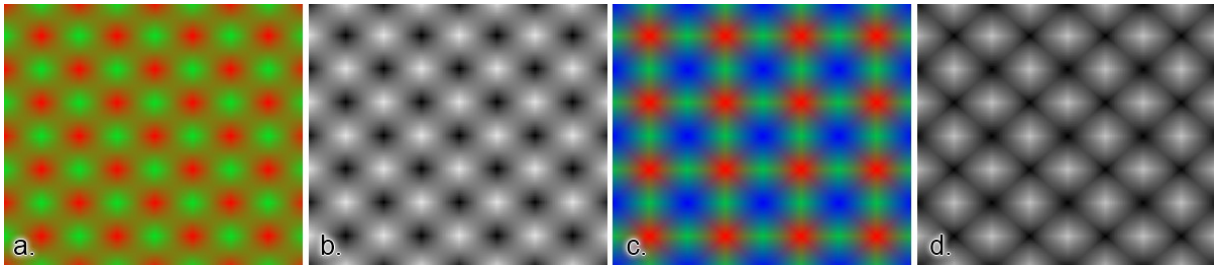


Figure IV.25: Séparation en zones selon une grille carrée.
a. séparation de l'espace selon deux couches (rouge et vert)
b. détail de la couche verte, les zones ne sont pas clairement séparées par un espace vide
c. séparation de l'espace selon trois couches (en rouge, vert et bleu). Les zones en rouge et en bleu sont légèrement plus grandes que les zones en vert.
d. détail de la couche verte, les zones sont plus petites, un espace vide est présent entre chaque zone.

Pour cela, les deux premières couches (rouge et bleu dans le schéma ci-dessus) seront légèrement plus grandes, mais moins nombreuses. Deux zones rouges ne se touchent pas, même par les coins. Les zones vertes, par contre, sont deux fois plus nombreuses et devraient donc se toucher par les coins. Afin d'éviter le contact, les zones vertes sont légèrement plus petites et décroissent très vite. Chaque carré de quatre zones contiendra ainsi une zone rouge, une zone bleue et deux zones vertes.

La structure de découpage de zones en grille s'adapte très bien à la structure de la texture de vélocité. Chaque zone correspond spatialement à un pixel différent de la texture de vélocité et tous les pixels seront utilisés.

Concrètement, cette structure ressemble à l'interpolation bilinéaire classique effectuée au moment de l'échantillonnage d'une texture. Cette interpolation mélange habituellement les résultats de la texture de vélocité. À la place, nous utilisons séparément les échantillons de la texture de vélocité pour déplacer la texture à advecter, et nous effectuons l'interpolation sur les résultats de cet échantillonnage. C'est un principe similaire qui est utilisé dans le PCF (Percentage-Closer Filtering) pour l'échantillonnage des textures d'ombres[92]. L'interpolation n'est pas effectuée sur les échantillons de la texture d'ombre, mais sur le résultat du test binaire d'appartenance à la zone d'ombre.

6.2.b Texture de positions advectées

Nous parlons depuis le début de l'étude d'une « texture de vélocité », mais nous allons voir qu'un autre format convient mieux. Il s'agit d'une texture de positions advectées. En effet, la

92 W. T. Reeves, D. H. Salesin, et R. L. Cook, « Rendering antialiased shadows with depth maps », in *ACM SIGGRAPH Computer Graphics*, vol. 21, 1987, 283–291.

texture de vélocité ne permet pas à elle seule de conserver une continuité temporelle dans l'image si la texture de vélocité se modifie à chaque image. Si la texture de vélocité est fixe, nous pouvons calculer les coordonnées de textures dans le « pixel shader » simplement par un décalage dans la direction de la vélocité proportionnel au temps passé. Par contre, si la vélocité se modifie au bout d'un certain temps, le mouvement sera également modifié comme si cette vélocité avait toujours été ainsi. Le mouvement observé est une rotation de la texture, à partir d'un point qui s'éloigne de plus en plus avec le temps, et non une rotation autour du centre du pixel concernée. Pour pouvoir effectuer la translation de texture correctement, nous allons devoir garder une trace des changements de vélocités.

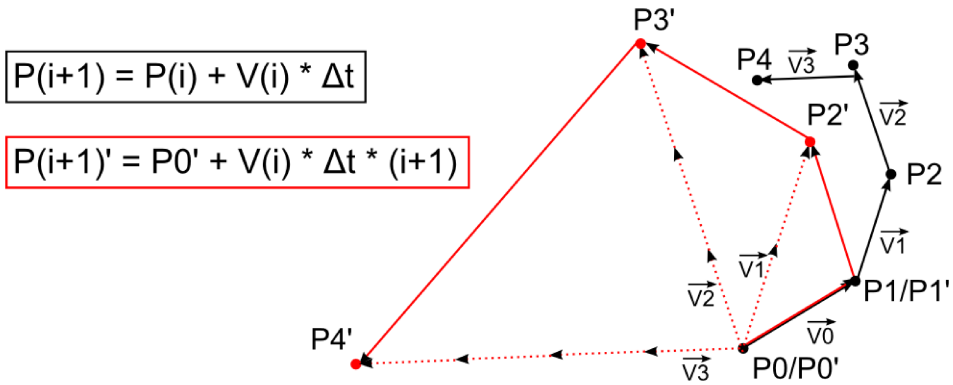


Figure IV.26: Trajectoire d'un point P dont la vélocité change à chaque image.
En noir, la trajectoire souhaitée et en rouge la trajectoire lorsque l'on applique la vélocité sans conserver la position de l'image précédente.

Cette trace va être directement conservée au niveau du CPU dans notre cas, mais pourrait également être conservée sur le GPU. Il s'agit d'une grille de positions advectées.

Au départ, la texture est remplie d'un couple de coordonnées aléatoires, entre 0 et 1 par exemple, puis ces coordonnées sont advectées par la simulation, de la même manière que si elles représentaient des positions de particules. Ainsi, la valeur stockée dans la texture subit à chaque image le calcul de sa nouvelle position (P(i+1)) en fonction de l'ancienne position (P(i)), de la variation de temps (Δt) et de la dernière vélocité calculée (V(i) :

$$P(i+1) = P(i) + V(i) \times \Delta t$$

La texture de vélocité est également mise à jour en parallèle. La texture de position sert de stockage de l'état et permet d'assurer la continuité. La vélocité peut changer d'une image sur l'autre, mais la texture de position conserve la dernière position calculée et présentera un

comportement cohérent. Les cellules de la grille de positions advectées sont similaires à des particules qui conserveraient leur état.

Il est également à noter que pour conserver de la précision dans la texture de positions advectées, nous stockerons plutôt un modulo de cette position entre 0 et 1. En effet, si le processeur principal utilise essentiellement du calcul en flottants 32 bits, la carte graphique préfère les flottants 16 bits, qui prennent moins de place et sont plus rapidement transférées vers la mémoire vidéo. Étant donné que la position advectée servira uniquement comme valeur de translation des coordonnées UV d'une texture répétable, nous pouvons éliminer la partie entière de la valeur. Celle-ci sert uniquement à indiquer la répétition, mais c'est la partie fractionnelle qui servira à l'échantillonnage dans la texture. Par contre, la valeur fractionnelle ne doit pas être multipliée ensuite sur le pixel shader, ou la répétition de la texture ne coïncidera plus avec le modulo appliqué. Grâce à cette astuce, la texture de positions advectées peut être stockée en flottants 16 bits, voir même en entiers 8 bits si la résolution de la texture qui sera advectée n'est pas trop élevée[93].

Dans les prochaines parties, nous utiliserons uniquement une grille de positions advectées à la place de la vélocité.

6.2.c Code du pixel shader

L'obtention d'un pavage carré est très simple, voici un exemple de code :

*UVcarré = floor(UVbase * Nb) / Nb*

floor : fonction ne conservant que la partie entière
Nb : nombre de carrés le long des deux axes de la grille

Ce code va créer des zones de forme carrées qui auront des coordonnées de texture uniformes.

La gestion de plusieurs couches décalées multiplie ce code, avec quelques instructions de décalage supplémentaires.

93 Si la texture qui est advectée a une résolution importante, la précision des coordonnées de texture utilisée doit également être élevée pour assurer un échantillonnage correct. Dans le cas contraire, le manque de précision se traduira par une pixelisation de la texture advectée.

Voici un extrait du « pixel shader » en HLSL permettant de découper selon quatre couches :

```
// Distance relative en x et y jusqu'au pixel courant
float2 Blend = float2(frac(TexCoord * VelocityTexSize));

// Coordonnée de texture du pixel courant
float2 SnapUV = uv - Blend / VelocityTexSize;

// Calcul des coordonnées de textures des quatre pixels voisins
float2 UV1 = SnapUV;
float2 UV2 = SnapUV + float2(1.0f, 0.0f) / VelocityTexSize;
float2 UV3 = SnapUV + float2(0.0f, 1.0f) / VelocityTexSize;
float2 UV4 = SnapUV + float2(1.0f, 1.0f) / VelocityTexSize;

// Récupération des coordonnées advectées dans la texture
// de la simulation physique
float2 GUV1  = tex2D(VelocityTex, UV1);
float2 GUV2  = tex2D(VelocityTex, UV2);
float2 GUV3  = tex2D(VelocityTex, UV3);
float2 GUV4  = tex2D(VelocityTex, UV4);

// La taille de répétition de la texture advectée est ajustable
float2 AdUV = uv * AdvScale;

// échantillonnages de la texture à advecter selon
// les quatre couches
float3 CL1 = tex2D(AdvectTex, AdUV – GUV1);
float3 CL2 = tex2D(AdvectTex, AdUV - GUV2);
float3 CL3 = tex2D(AdvectTex, AdUV - GUV3);
float3 CL4 = tex2D(AdvectTex, AdUV - GUV4);

// Interpolation bilinéaire des quatre couches pour former
// une couleur unique
float3 LerpX1 = lerp(CL1, CL2, Blend.x);
float3 LerpX2 = lerp(CL3, CL4, Blend.x) ;
float4 FinalTexture = float4(lerp(LerpX1, LerpX2, Blend.y), 1.0f);
```

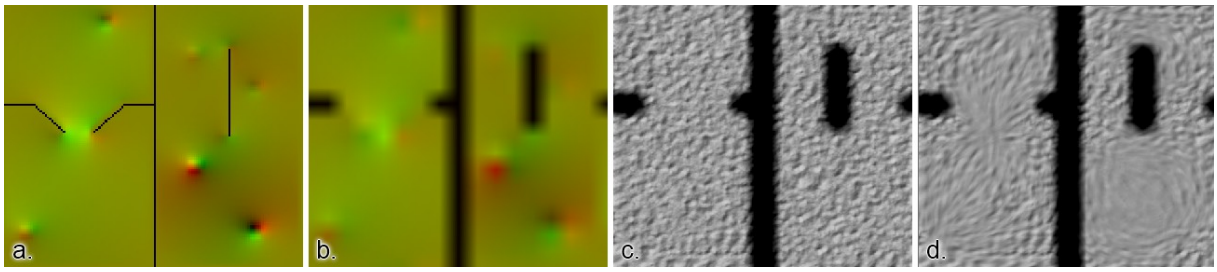


Figure IV.27: Utilisation d'une texture de vélocité avec l'advection en zones carrées. a. Texture de vélocité originale, b. mélange des vélocités des couches, c. advection d'une texture de vague par la vélocité, d. moyenne de dix images successives permettant d'observer les traînées de mouvement.

Nous n'avons pas inclus le code permettant le découpage en seulement trois couches, car il est plus complexe. Il permet en effet de réduire la taille des zones d'une des couches, qui va alors réunir deux des couches de la version ci-dessus en une seule, sans laisser voir de discontinuités.

L'utilisation d'une grille rectangulaire simple est très intéressante et les calculs sont simples et très rapides sur la carte graphique. La simulation physique de fluide peut être directement calculée sur le GPU, mais dans notre cas nous l'effectuons sur les coprocesseurs de calcul vectoriel de la PS3, qui sont encore plus adaptés à ce type de calculs. La XBox360 utilisera directement son processeur principal. Si le calcul n'est pas fait sur la carte graphique, il est nécessaire de transférer la texture de positions advectées depuis la mémoire principale vers la mémoire de la carte graphique. Le transfert est très rapide étant donné que la simulation peut être calculée dans une résolution faible tout en ayant un rendu visuel convaincant grâce à l'advection de textures.

La structure en grille présente un défaut majeur, car la division en cellules est assez perceptible, notamment lorsque le déplacement du fluide ne suit pas l'un des deux axes de la grille. Lorsque le mouvement s'opère en diagonale, les frontières et les zones de mélange entre les couches sont clairement visibles quand le courant devient rapide. L'œil reconnaît un défaut de continuité qui suit une ligne droite et nous percevons la forme de la grille. Les défauts de la technique de découpe en zones arbitraires sont également présents ici. La taille des zones doit être en équilibre avec la vitesse du fluide et l'échelle de la texture advectée afin de permettre au mouvement d'être perçu correctement. De plus, la taille des zones est fixe et ne peut pas être modifiée en fonction de la vitesse du fluide. Par contre, la grille régulière remplace la définition manuelle des zones et la texture de vélocité peut être utilisée directement, ce qui rend l'algorithme plus rapide.

6.3 Advection suivant une grille hexagonale

Nous avons vu que le choix d'une grille régulière de carré n'offre pas une très bonne qualité et souffre d'artefacts visibles lorsque le mouvement n'est pas aligné avec l'un des deux axes de la grille. Nous avons donc cherché un autre type de grille régulière qui ne présente pas la même sensibilité à la direction du mouvement. La forme hexagonale est ici idéale, car elle permet de créer un pavage périodique, comme le carré, mais se rapproche d'un cercle et présente des

coins moins aigus. Les hexagones forment une grille régulière dont une ligne sur deux est placée en quinconce.

6.3.a Découpage en zones hexagonales

La grille hexagonale possède plusieurs avantages, notamment celui de présenter trois directions principales au lieu des deux axes d'une grille carré. L'ajout de cette troisième direction permet de gérer beaucoup mieux les mouvements de translation en diagonale de la texture advectée.

Le pavage hexagonal fait partie des trois seuls pavages réguliers pouvant recouvrir un plan, les deux autres étant le pavage triangulaire et le pavage carré. L'hexagone est la forme qui se rapproche le plus du cercle parmi ces trois formes. Le pavage hexagonal est également la manière la plus dense de remplir un plan avec des sphères.

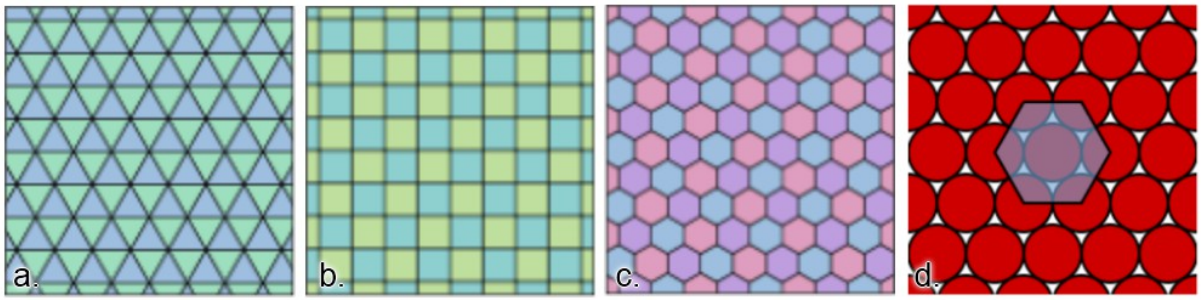


Figure IV.28: Les pavages réguliers du plan.
Pavages réguliers par des triangles (a.), des carrées (b.), des hexagones (c.) et couverture compact du plan par des cercles (d.)
Images Wikipedia (R.A.Nonenmacher et Inductiveload)

La caractéristique qui est la plus appréciable avec le pavage hexagonal est la facilité avec laquelle nous pouvons masquer les bords entre les hexagones en utilisant seulement trois couches. Ces trois couches sont décalées les unes par rapport aux autres, chacune prenant pour centre des hexagones l'un des sommets d'une autre couche. Ainsi, nous pouvons utiliser trois couches de pavages hexagonaux pour créer des zones de translation uniforme sans que les bords soient visibles.

Le pavage hexagonal présente néanmoins un inconvénient, car il ne correspond plus directement à la grille de vélocité qui est organisée en cellules carrées. Deux possibilités sont envisageables : nous pouvons soit chercher à faire correspondre exactement une cellule de la grille carrée à une cellule de la grille hexagonale, soit nous choisissons simplement la cellule carrée la plus proche de chaque cellule hexagonale.

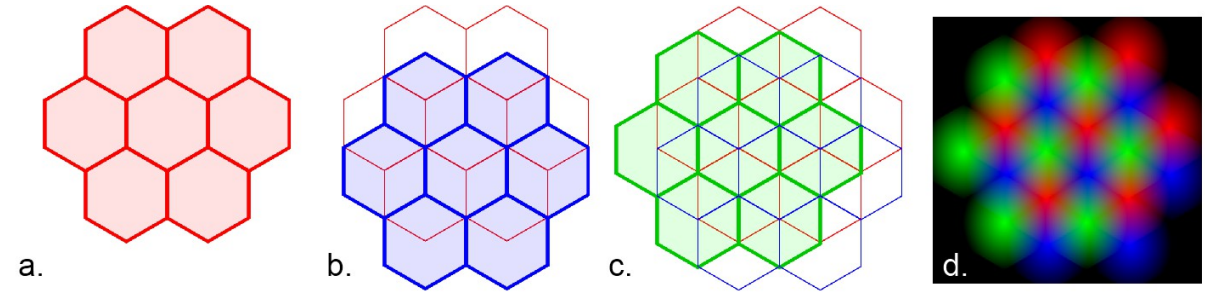


Figure IV.29: Utilisation de trois couches hexagonales pour assurer des transitions douces. a. Première couche de pavage hexagonal, b. seconde couche de pavage placée en décalage, c. troisième couche, d. mélange des trois couches afin de masquer les bordures.

Si nous utilisions une seule couche d'hexagones, la correspondance entre les deux grilles pourrait être faite facilement. Nous pouvons décaler une ligne d'hexagones sur deux pour annuler l'aspect « quinconce » de la grille hexagonale. Nous nous retrouvons alors avec une structure carrée qui correspond à la grille de vélocité. Chaque pixel de vélocité correspond à un hexagone différent et inversement. Par contre, avec trois couches de pavages hexagonaux qui utilisent toutes les trois la même grille de vélocité, l'association n'est pas aussi évidente. La structure à base de trois pavages intriqués ne correspond pas bien à la forme carrée de la grille.

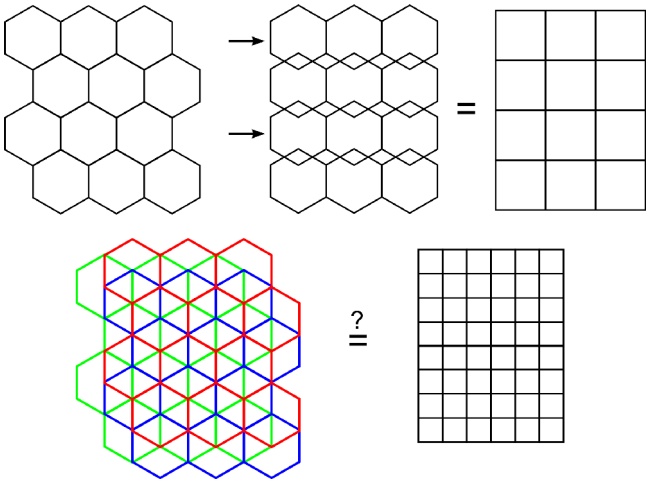


Figure IV.30: Le stockage des informations d'une grille hexagonale.

Sur la première ligne :
Adaptation d'une structure hexagonale à une structure carrée.

Sur la seconde ligne :
L'adaptation de trois couches hexagonales à une structure carrée commune n'est pas triviale.

La couverture spatiale d'un pavage hexagonal n'est pas la même sur les deux axes d'un plan. L'axe selon lequel les hexagones sont placés en quinconce est plus compressé que l'autre axe. Si nous admettons que les hexagones sont placés en quinconce selon l'axe X, le nombre d'hexagones nécessaires pour couvrir une aire carrée est environ 1,15 fois plus grand sur l'axe Y que sur l'axe X.

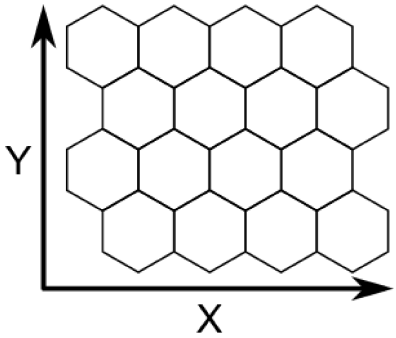


Figure IV.31: Ratio entre les axes X et Y d'une grille de 4x4 hexagones.

En effet, si nous nommons la longueur d'un coté de l'hexagone r, alors la distance entre deux hexagones sur l'axe X est de $\frac{3}{2}r = 1{,}5\,r$ alors que sur l'axe Y elle est de $\sqrt{3}\,r \approx 1{,}732\,r$, le ratio entre les deux axes est donc de $\frac{2\sqrt{3}}{3} \approx 1{,}15$

Nous devons donc utiliser une grille de vélocité rectangulaire au lieu de carrée si nous souhaitons obtenir une correspondance directe entre les deux pavages.

Finalement, nous avons choisi de ne pas imposer une relation précise entre les hexagones et les cellules de la grille de vélocité afin d'ajouter le moins de contraintes possible sur les calculs. Ainsi, chaque hexagone va utiliser simplement la vélocité de la cellule la plus proche.

Cette méthode permet également d'utiliser facilement une orientation et une taille différente entre la grille hexagonale des textures et la grille carrée de la simulation. Le point principal à respecter pour obtenir de bons résultats est d'assurer une densité de la grille de vélocité suffisante pour que chaque hexagone aille utiliser une cellule différente. Ainsi, bien que tous les points de la grille de vélocité ne soient pas représentés par une zone, chaque zone utilise bien une vitesse de déplacement différente. Si la résolution de la grille de vélocité n'est pas suffisante, plusieurs hexagones utiliseront la même vitesse et la même direction de déplacement, ce qui n'est pas toujours visible directement, mais diminue la qualité globale en créant des zones d'advection uniforme plus larges qu'un hexagone. Les transitions douces entre zones ne peuvent pas prendre plus d'espace que la largeur d'un hexagone. Le mouvement du fluide est donc constitué de plaques plus larges avec des frontières plus abruptes.

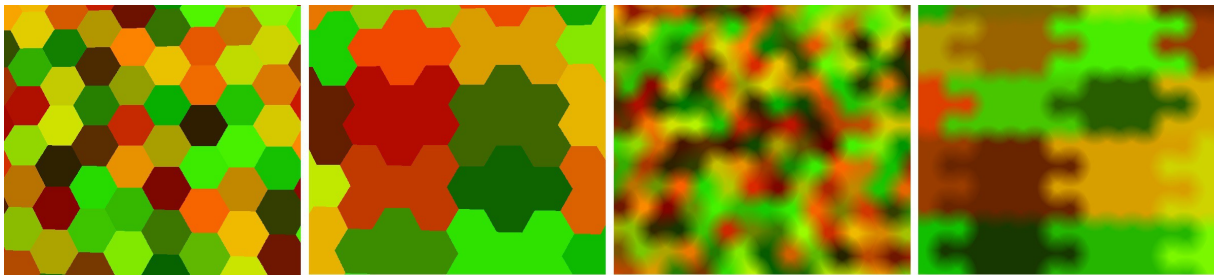


Figure IV.32: Artefacts liés à la faible résolution de la texture de vélocité.
De gauche à droite : le cas idéal où chaque hexagone d'une couche référence une cellule différente, puis le cas où plusieurs hexagones partagent la même cellule. Ensuite, les mêmes cas avec le mélange des résultats des trois couches.

6.3.b Calculs de la grille et des poids entre couches

Pour faire fonctionner notre algorithme, nous devons être capables de connaître, pour chaque pixel, la position d'échantillonnage de chaque couche. Ainsi, le but de ces calculs est d'obtenir

trois coordonnées de texture, une pour chaque couche, ainsi que les poids à appliquer à chacun des trois échantillons effectués.

Les calculs pour l'obtention d'une grille hexagonale sont plus complexes que pour la grille carrée. Voici un exemple de code HLSL pour une seule couche d'hexagones, ainsi que les images de chaque étape de l'algorithme :

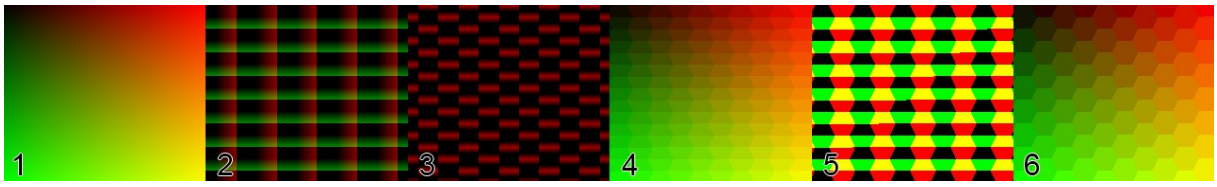


Figure IV.33: Les étapes de création d'une grille hexagonale sur le pixel shader.

```
// Spécification du nombre d'hexagones (1)
float2 Taille = float2(10.0f, 15.0f);
float2 SUV = uv * Taille;

// Création de la grille principale (2)
float2 Grid = frac(SUV * 0.5f + float2(0.25f, 0.0f)) - 0.5f;

// Calcul d'un décalage en zigzag (3)
float Offset = (abs(Grid.y * 2) - 0.5f ) * 1.333 * sign(Grid.x);

// Application du décalage et création des deux
// moitiés des hexagones (4)
float2 HexUV1 = SUV + float2(Offset * 0.25f, 0);
float2 HexUV2 = (floor(HexUV1) + 0.5f) / Taille;

// Sélection des deux moitiés des hexagones (5)
float2 UV2 = floor(frac(HexUV1 * 0.5f) * 2);

// Réunion des deux moitiés des hexagones (6)
float2 HexUV3 = HexUV2;
HexUV3.y += (UV2.y - 0.5f) / Taille.y * (UV2.x * 2 - 1) ;
```

Nous pouvons constater que le calcul est bien plus complexe et nécessite beaucoup plus d'instructions. Le calcul de trois couches nécessite à peu près le triple de calculs. Il est bien sûr possible de factoriser efficacement les instructions des trois couches, afin d'utiliser la puissance du calcul vectoriel de la carte graphique. En effet, la différence de temps de calcul d'une instruction sur un élément unique ou bien sur un élément constitué de quatre composantes est très réduite sur la carte graphique. Cela nous permet par exemple de calculer en une seule instruction le résultat sur les trois couches.

Le calcul des poids entre les différentes couches peut s'effectuer simplement en utilisant la distance entre les coordonnées de texture initiales et celles que nous avons calculées.

L'inverse de cette distance représente effectivement, une fois correctement multiplié, le poids de la couche concernée.

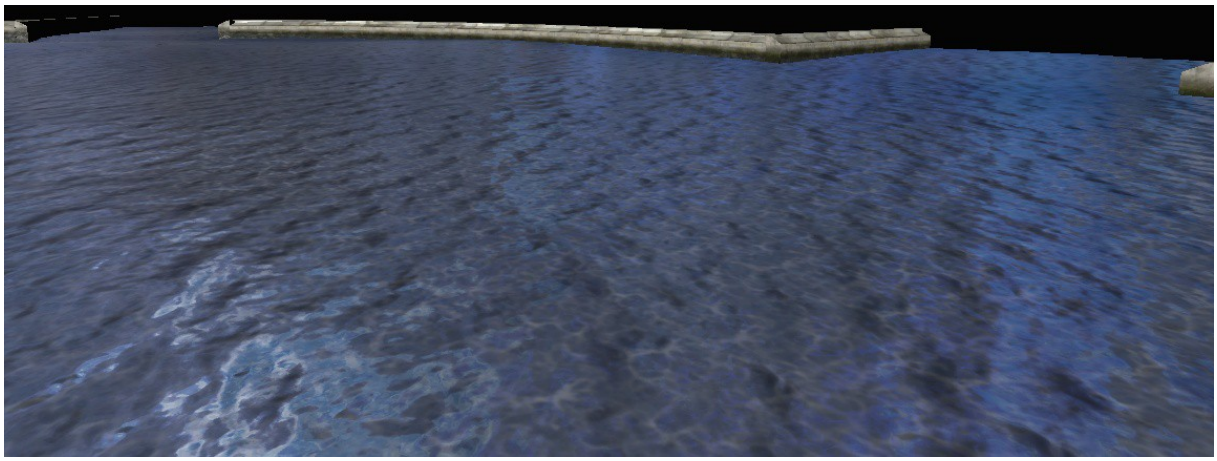


Figure IV.34: Premiers résultats de l'advection d'une texture de normale selon une grille de zones hexagonales.

6.3.c Précalcul des couches dans une texture

Devant la quantité de calculs à effectuer pour séparer les couches, nous avons recherché des moyens de précalculer ces informations et de les stocker dans une texture. Ce type de technique permet d'échanger du temps de calcul contre du temps d'accès mémoire, ce qui est souvent très efficace, car les accès textures peuvent être effectués en parallèle des autres calculs.

Le premier test effectué consiste à enregistrer, dans la texture, la différence de coordonnées UV classiques et celle d'une couche hexagonale. Ainsi, il suffit d'échantillonner la texture puis d'ajouter ce résultat aux coordonnées de texture pour obtenir les nouvelles coordonnées d'une couche. Nous faisons de même pour les deux autres couches. La situation se répète assez rapidement sur la grille hexagonale. La texture pourra donc se concentrer sur une zone restreinte comportant seulement un hexagone sur Y et deux sur X.

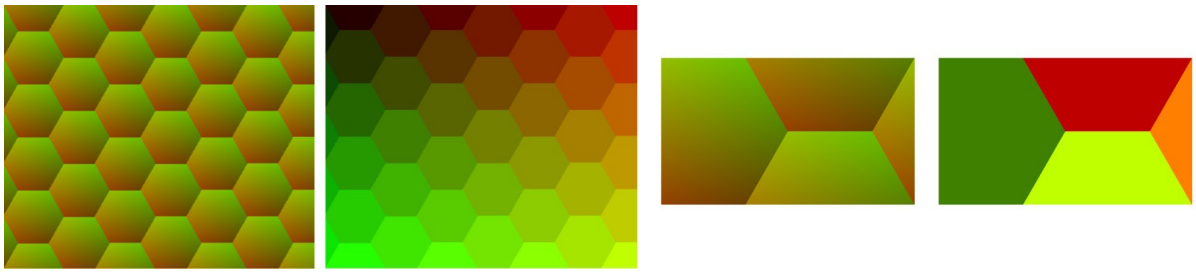


Figure IV.35: Création d'une texture répétable des coordonnées hexagonales.
À gauche, la différence entre les coordonnées normales et hexagonales suivit des coordonnées hexagonales seules. À droite, la texture répétable extraite des deux textures.

Nous constatons que la texture précalculée produite présente essentiellement des gradients doux, puis des discontinuités fortes entre les différentes zones, mais aucune surface unie. De ce fait, cette technique impose des contraintes fortes sur la texture. La moindre différence de calcul entre deux pixels de la même zone va produire des coordonnées de texture légèrement différentes, mais qui suffiront à introduire des discontinuités visibles.

Pour obtenir une qualité maximum, il convient d'éviter de compresser la texture en DXT, car la compression va modifier légèrement les valeurs, notamment aux discontinuités entre zones. De même, les techniques de types mip-maps doivent être évitées. Si l'utilisation du « mip-mapping » est nécessaire, les textures des différents niveaux de mip-maps doivent être produites séparément et ne peuvent pas être de simples versions réduites du niveau supérieur.

Un second test consiste à stocker le décalage de coordonnées depuis le coin en haut à gauche de la texture. Ainsi, contrairement au test précédent, la texture est constituée de grandes zones unies, avec la même valeur de décalage, à la place des gradients. La texture résiste alors mieux à la compression et au mip-mapping. Par contre, pour reconstituer les coordonnées d'une couche, nous avons besoin des coordonnées du coin supérieur gauche de chaque répétition de la texture, ce qui nécessite quelques calculs, d'une complexité équivalente à ceux d'une grille carrée.

Le troisième test cherche à éviter de faire appel à la texture pour chacune des couches. En effet, jusqu'à présent chacune des trois couches devait aller chercher une valeur dans la texture de précalcul, avec un décalage différent. Nous pouvons tenter de concentrer les informations des trois couches dans une seule texture, afin de les récupérer avec un seul échantillonnage au lieu de trois. Les informations que nous souhaitons pouvoir connaître avec cette texture sont : les trois coordonnées UV des couches, c'est-a-dire 2x3 = 6 informations, et les poids des 3 couches. En tout, nous avons donc 9 canaux à stocker. Comme l'addition des poids des trois couches est égale à 1, nous pouvons stocker seulement deux poids et déduire le dernier avec le calcul : P3 = 1 - P1 - P2

Nous avons donc 8 canaux, ce qui peut très bien se stocker dans deux textures de quatre canaux chacun. Pour arriver à une seule texture, nous allons stocker uniquement un indice qui référence les coordonnées de texture stockées statiquement dans un tableau. Cette technique est possible, car nous n'avons besoin de stocker qu'un nombre limité de valeurs différentes. Chaque zone de chaque couche possédera ainsi son propre indice.

Pour compresser les index des trois couches en une seule valeur, nous allons calculer une combinaison entre les indices des zones des trois couches. Le nombre de combinaisons est assez grand. Ainsi, chaque couche présente 5 zones différentes, ce qui nous donne $5^3 = 125$ indices. Ces indices peuvent être stockés dans un canal 8 bits (maximum 256 indices différents). Obtenir les coordonnées de texture peut alors se faire de deux manières. Soit en indexant directement dans un tableau très grand (125 entrées) contenant les trois coordonnées de texture associées à l'indice. Soit en séparant d'abord les indices des trois couches individuelles. Cette seconde méthode est assez coûteuse en performance. Il faut en effet utiliser deux opérateurs modulo pour séparer les indices de chaque couche. Une fois ces indices retrouvés, l'association avec les coordonnées de textures originales se fait également par un tableau, mais de seulement cinq entrées cette fois.

Étant donné que nous n'avons besoin que de deux canaux pour stocker les poids de mélange des trois couches, nous disposons de deux autres canaux libres dans notre texture. En effet, une texture peut contenir quatre canaux en tout : rouge, vert, bleu et alpha. Nous pouvons donc répartir l'information des indices sur deux canaux. Ainsi, le premier canal peut stocker la combinatoire de deux couches, c'est-à-dire $5^2 = 25$ indices différents. L'autre canal stockera directement l'index de la dernière couche.

I1, I2 et I3 sont les indices des trois couches
P1 et P2 sont les opacités des couches 1 et 2

Compactage :
C1 = I1 / 25 + I2 / 5
C2 = I3/5
C3 = P1
C4 = P2

Décompactage :
*I2 = floor(C1 * 5)*
*I1 = C1 * 25 – I2 * 5*
*I3 = C2 * 5*
P1 = C3
P2 = C4
P3 = 1 – P1 – P2

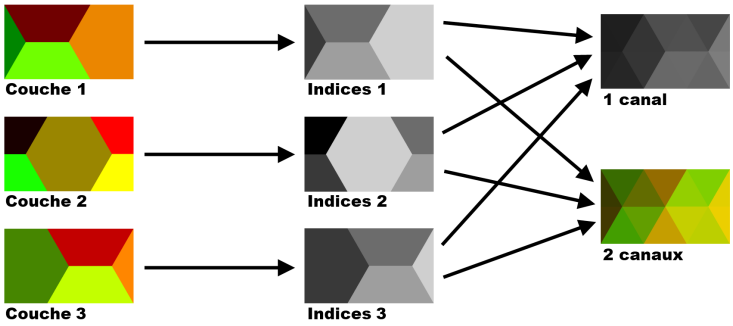


Figure IV.36: Compression des informations des trois textures en un seul canal ou en deux canaux.

Grâce à cette technique, le nombre de calculs est très réduit. En plus des instructions ci-dessus, un simple modulo nous permettra de connaître la position du coin supérieur gauche de la texture et de faire l'appel à la texture en lui même. La texture doit toujours être non compressée et le mip-mapping nécessite des précautions supplémentaires telles que mentionnées auparavant.

L'utilisation d'une grille hexagonale nous a permis d'améliorer la qualité de l'advection par rapport à la grille carrée. Les mouvements sont mieux représentés, car les axes principaux de la grille hexagonale sont moins visibles que sur une grille carrée. En contrepartie, nous avons un nombre de calculs plus important à gérer. L'utilisation d'une texture de précalculs permet de revenir à des temps de calcul raisonnables. Cette technique d'advection de texture est possible facilement en temps réel (>30fps) sur la console PS3, tout en effectuant la simulation de fluide proprement dite et le reste des calculs du jeu.

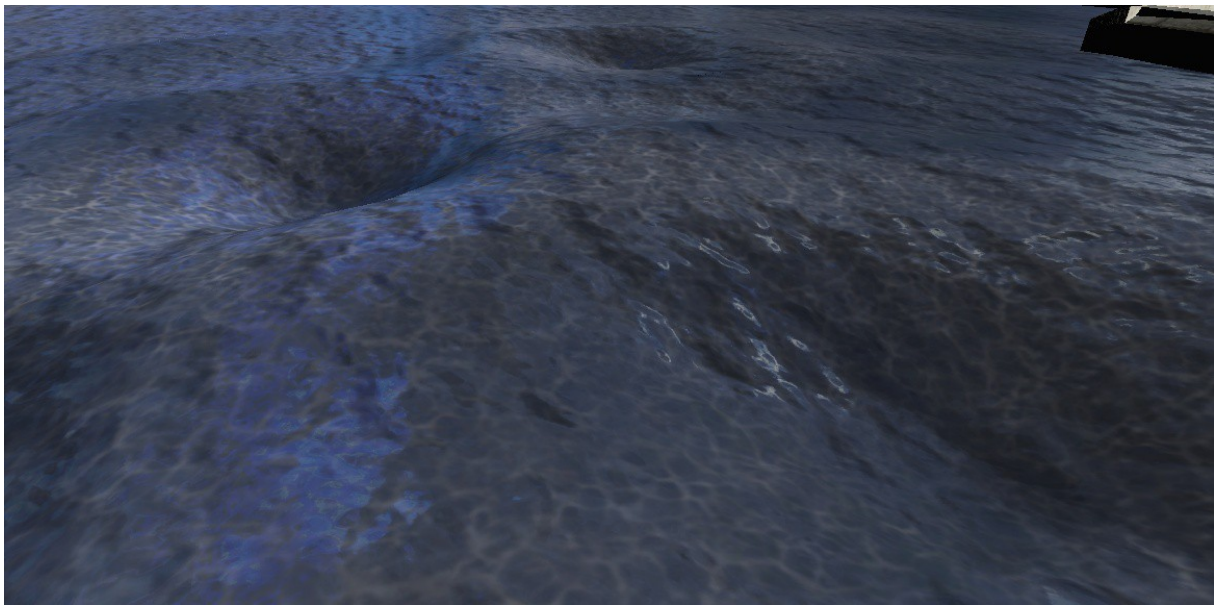


Figure IV.37: Advection d'une texture de normale appliquée sur la surface d'un fluide en mouvement. La hauteur des points de la surface est animée par la simulation de fluide.

6.3.d Gestion de l'advection de texture entre deux simulations

Dans notre application, chaque « bac » de fluide possède sa propre simulation qui met à jour sa propre texture de « positions advectées ». Cette texture est envoyée sur la carte graphique et servira à l'advection de textures. Jusqu'à présent, les simulations étaient considérées comme indépendantes les unes des autres du point de vue de l'advection de texture. Seule la partie simulation physique de la vélocité se chargeait d'adoucir les frontières entre simulations voisines. Notre procédé d'advection de texture ajoute de nouvelles contraintes pour améliorer l'aspect des frontières.

Discontinuités aux frontières

Les zones hexagonales d'advection uniforme ne sont pas localisées aux mêmes endroits des deux côtés d'une frontière, étant donné que les hexagones sont créés dans l'espace local de la simulation. De plus, chaque côté de la frontière utilise une texture de vélocité différente, et produira donc des translations différentes.

Une autre source de discontinuité concerne le changement d'espace de la texture de normale. En effet, la texture en translation est souvent une texture de normale qui permet d'ajouter des détails qui réagissent en fonction de la direction de la lumière. Cette texture nécessite un espace de travail adéquat, qui est généralement l'espace tangent, c'est-à-dire l'espace constitué par les deux axes des coordonnées de textures et par la normale de la surface. Or, cet espace est différent pour chaque bac de fluide. Dès qu'une rotation est appliquée sur le conteneur de fluide, l'espace local et l'espace tangent tournent avec lui. D'un côté et de l'autre d'une frontière, la même texture de normale sera interprétée comme des directions différentes dans l'espace global. La frontière entre deux simulations devient alors très visible. Cet effet est accentué si la texture de normale utilisée comporte des caractéristiques orientées selon une direction particulière, formant une sorte d'anisotropie. Lorsque l'orientation est différente d'une simulation à la suivante, cette anisotropie change de direction, ce qui est immédiatement repérable par l'œil et forme une frontière évidente. Il est difficile de créer une texture sans orientation particulière tout en conservant un aspect naturel.

Toutes ces raisons vont créer des discontinuités importantes le long de la limite entre deux simulations voisines.

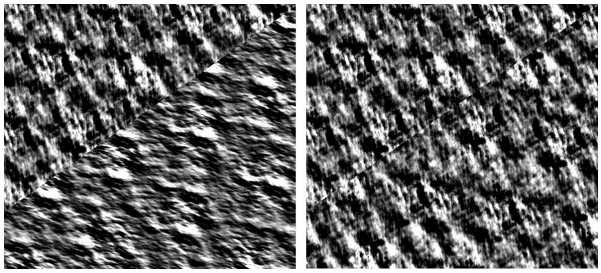


Figure IV.38: Artefacts aux frontières entre simulations en espace local et global. À gauche, la frontière (en diagonale) entre deux simulations en espace local est clairement perceptible à cause du changement d'orientation de la texture. À droite, l'espace global permet de conserver l'orientation : la frontière est toujours présente mais bien moins visible.

Passage en espace global

Les problèmes dont nous avons parlé peuvent être contrés par une solution commune : l'utilisation d'un espace global. Au lieu de définir les hexagones par rapport aux coordonnées de texture, c'est-à-dire par des coordonnées allant de 0 à 1 d'un bout à l'autre de la simulation, nous utilisons les coordonnées dans l'espace monde. Une fois les coordonnées des hexagones créées, il est nécessaire de les convertir dans l'espace local avant d'échantillonner la texture de vélocité, qui est alignée avec l'orientation de la simulation. Cette étape est nécessaire à partir du moment ou nous utilisons un espace global, mais elle est coûteuse en performance. La direction de la vélocité, qui est stockée dans la texture, doit également être convertie dans l'espace monde avant de pouvoir servir à déplacer la texture de normale.

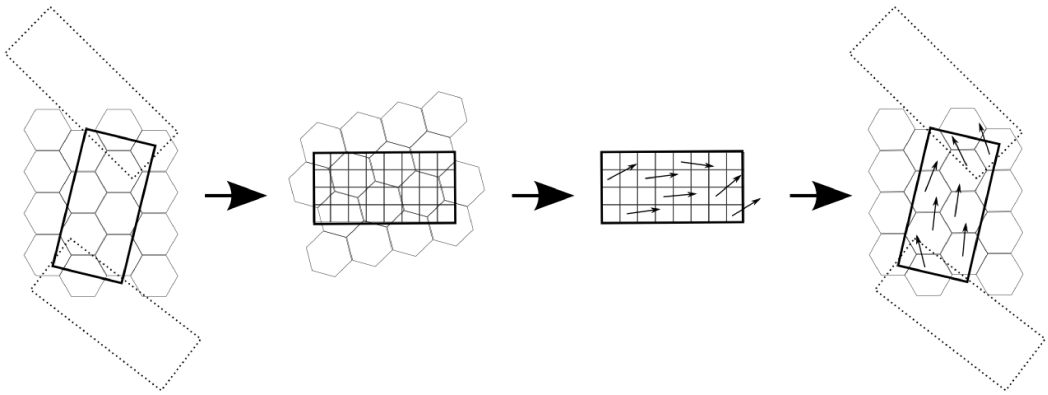


Figure IV.39: Utilisation d'une grille hexagonale dans l'espace du monde. De gauche à droite : les hexagones sont placés dans l'espace monde puis transférés dans l'espace local de chaque « bac » de simulation. Les vélocités sont alors extraites de la grille locale de la simulation puis transférées vers l'espace monde.

Heureusement, nous pouvons nous limiter à la gestion d'espaces bidimensionnels puisque nous nous intéressons uniquement à des simulations planes. Les calculs de transfert de direction d'un espace à l'autre se simplifient en 2D. Dans le cas de la 3D, il serait nécessaire d'utiliser une multiplication de matrice complète pour le changement des coordonnées de

texture de chaque couche d'hexagones dans l'espace local, puis une seconde multiplication de matrice pour transformer la vélocité issue de la texture de vélocité pour chaque couche dans l'espace global. En tout, 6 multiplications de matrices par pixel, ce qui est très lourd. Dans le cas de la 2D, les calculs peuvent être grandement simplifiés et certains paramètres peuvent être précalculés, car l'orientation est fixe pour tout les pixels d'un « bac » de fluide.

Une rotation de coordonnée en 2D selon un angle thêta autour du point {0,0} se calcule très simplement comme ceci :

$$F.x = I.x \times \cos(thêta) - I.y \times \sin(thêta)$$
$$F.y = I.x \times \sin(thêta) + I.y \times \cos(thêta)$$

I : coordonnée d'entrée
F : coordonnée de sortie
thêta : angle de rotation

Mais peut se simplifier en précalculant cos(thêta) et sin(thêta) et en effectuant les calculs sur les deux composantes (X et Y) en même temps :

Ro = float2(cos(thêta), sin(thêta))
Inv = float2(-1, 1)

$$F = I.x \times Ro.xy + I.y \times Inv \times Ro.yx$$

Grâce à ce passage par un espace global, les hexagones sont maintenant exactement alignés d'une simulation à la suivante, et les textures de normale en translation sont également alignées. Les transitions sont alors très peu visibles, mais pas complètement masquées. En effet, même si les vélocités des deux côtés de la frontière sont similaires, il est difficile de garantir la parfaite égalité. La moindre différence de vélocité entre les deux côtés va s'amplifier jusqu'à aboutir à des positions advectées complètement différentes. De plus, les positions de départ des deux côtés sont aléatoires, donc différentes.

Il est à noter qu'il est très difficile de donner exactement la même vitesse aux deux côtés des hexagones sur la frontière sans que toute la frontière n'utilise la même vitesse. Dans ce dernier cas, cela créerait une zone unique pour la frontière qui serait très grande et peu esthétique.

Deux solutions sont en théorie disponibles pour régler ce problème de vitesses différentes :

- copier les valeurs de toute la zone de transition de la grille d'une simulation à l'autre
- retrouver les positions de chaque hexagone sur le processeur principal afin de copier les valeurs uniquement pour ces positions.

La première solution n'est pas envisageable, car il n'y a pas de restrictions sur l'emplacement et l'orientation de la frontière, et donc pas de moyen générique d'associer les deux grilles carrées des textures de vélocité. Chaque carré d'une grille peut potentiellement recouvrir plusieurs carrés de l'autre grille et inversement, donc impossible de faire un transfert parfait de l'une vers l'autre.

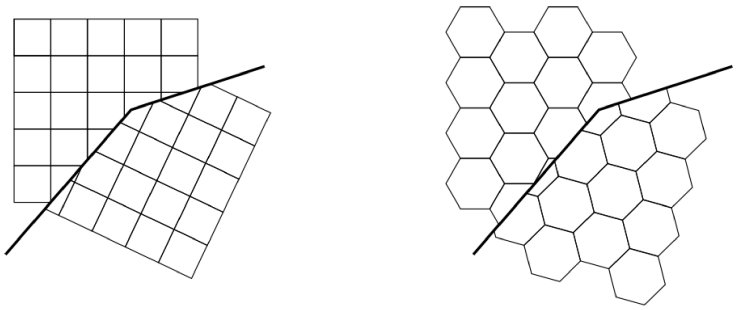


Figure IV.40: Alignement impossible entre les deux grilles carrées ou hexagonales des deux cotés d'une frontière.

La seconde solution nécessiterait beaucoup de calculs afin d'associer les hexagones à leur emplacement de grille de chaque côté de la frontière, puis copier la valeur d'un côté à l'autre. Cette solution ne fonctionne que si chaque hexagone de chaque couche est associé à un emplacement de grille différent des autres.

Nous avons décidé de ne pas utiliser ces techniques car le résultat est déjà très correct.

Au final, l'utilisation des hexagones en espace global réduit les artefacts aux frontières, qui deviennent difficiles à repérer, sans nécessiter un surcoût rédhibitoire. De plus, le passage en espace global permet d'utiliser tout type de texture de normales sans problème d'orientation. Par contre, la taille des hexagones est fixée une fois pour toutes les simulations, et ne peut pas être modifiée individuellement sans retrouver les artefacts aux frontières. Adapter la taille de la grille hexagonale en fonction de la résolution et de la taille de la simulation aurait pu être un outil intéressant d'amélioration de la qualité.

6.3.e Simplifications du fluide en fonction de la distance

Lors de la conception d'un effet particulier, tel que les fluides, deux étapes doivent être travaillées : le prototype et le système en condition réelle.

Nous commençons toujours par un prototype qui permet de valider les techniques et la qualité atteignable. Il est important ensuite d'intégrer les contraintes de production dans le système final. Notre effet de fluide doit bien sûr être optimisé, mais il doit surtout être capable de s'adapter aux diverses situations. Pour cela, nous devons pouvoir faire varier la qualité en fonction des performances disponibles.

Dans le prototype, seules deux ou trois simulations sont présentes, et les conditions externes ne changent pas : les décors sont fixes, il n'y a pas d'ennemis et la caméra ne s'éloigne pas de la scène. En condition réelle par contre, les simulations doivent pouvoir se compter par dizaines dans un niveau et la caméra passe d'une simulation à l'autre, pouvant même revenir sur ses pas. Les décors sont plus ou moins chargés selon les endroits et la gestion d'ennemis et du combat prendra une part très importante des ressources de la machine.

La technique principale pour adapter notre système aux situations réelles est d'utiliser la distance entre la caméra et la simulation pour faire varier le niveau de qualité de la simulation. Ainsi, une simulation qui n'est visible que sur quelques pixels n'aura pas besoin d'une précision importante et l'advection de texture peut même être complètement évitée. Par contre, les simulations proches de la caméra doivent présenter une qualité maximale. Cette adaptation impose certaines contraintes, notamment sur le passage d'une simulation d'un niveau de qualité à un autre, qui doit être le plus doux possible.

Au niveau de la simulation de fluide en elle-même, la résolution de la grille est ajustée en fonction de la distance, jusqu'à devenir un simple mécanisme de transmission d'une quantité de fluide aux simulations voisines connectées. Nous pouvons précalculer la quantité moyenne de fluide qui passe habituellement d'une simulation à la suivante, ainsi le flux principal de fluide n'est pas perturbé par les changements de résolution des simulations.

Nous pourrions également changer la résolution de la texture de positions advectées, qui sert à animer les textures. Malheureusement, notre système à base d'hexagones ne fonctionne pas correctement si la taille des hexagones n'est pas en adéquation avec la résolution de la texture de positions advectées. Nous avons en effet précisé précédemment que chaque hexagone doit dans l'idéal utiliser un pixel différent de la texture de vélocité, ce qui impose une limite

minimum à la résolution de celle-ci. Étant donné que nous souhaitons conserver une taille fixe d'hexagones, la résolution de la grille de positions advectées est également fixe.

Cette résolution peut être indépendante de celle de la simulation de fluide, car nous pouvons effectuer une interpolation bilinéaire de la grille de vélocité au moment de la création de la texture de positions advectées afin de fournir une vélocité interpolée différente pour chaque pixel. La diminution de la résolution de la grille de positions advectées au profit d'une interpolation bilinéaire directement sur la carte graphique n'est pas possible, car la cohérence des valeurs entre les pixels voisins se perd très rapidement au fur et à mesure de la simulation. Il devient donc très vite inutile de chercher à calculer une valeur intermédiaire.

Néanmoins, nous souhaitons pouvoir gagner en performance sur tous les points de l'algorithme afin de ne pas limiter le nombre de simulations. Les simulations loin de la caméra ne doivent pratiquement rien coûter en termes de performance.

6.3.f Advection sans simulation

En ajustant légèrement notre algorithme d'advection de texture, il est tout à fait possible d'arrêter la mise à jour de la grille de positions advectées en conservant un mouvement visuel périodique. Ainsi, les « bacs » lointains continuent à présenter des mouvements de textures, du détail animé, mais ces mouvements ne changeront plus de direction en fonction de la simulation, qui n'a donc plus besoin d'être mise à jour. Les simulations lointaines ne coûtent donc plus rien en performance sur le processeur principal, mais uniquement de la place en mémoire vidéo pour la grille de positions advectées et de la puissance graphique pour l'affichage des pixels associés. Les performances du rendu des pixels dépend bien évidemment du nombre de pixels et donc de la taille de la surface à l'écran qui décroit avec la distance. Les simulations lointaines deviennent donc également moins gourmandes en terme de coût d'affichage.

Afin de continuer l'advection de texture bien que la simulation physique soit à l'arrêt, nous avons besoin de modifier légèrement l'algorithme d'advection ainsi que les données stockées dans la texture d'advection. Le but est d'utiliser des mouvements sans sauvegarde d'état. Ces mouvements ne sont plus itératifs, mais paramétriques, dépendant essentiellement de la quantité de temps passé pour évoluer.

De base, chaque pixel de la texture de positions advectées contient une position 2D qui est modifiée à chaque image par la simulation, ainsi qu'une information de quantité d'écume

(nous verrons plus tard son utilité lors de l'affichage). Afin d'opérer une advection sans sauvegarde d'état, nous avons besoin d'une direction 2D de vélocité ainsi que d'une position de départ, et bien sur de l'information d'écume. En plus, nous avons besoin de la valeur de temps passé depuis le moment de l'arrêt. Cette dernière valeur est commune à tout le bac de fluide et n'a donc pas besoin d'être stockée dans une texture.

Au moment de l'arrêt, les positions advectées de l'ancienne grille sont copiées dans la nouvelle grille de positions de départ, de même que la vélocité courante.

Le calcul sur la carte graphique est alors simple :

$$Pi = P0 + V0 \times Ti$$

Pi : position à l'image i
P0 : position de départ
V0 : vitesse initiale
Ti : temps passé depuis l'arrêt

Au niveau du stockage, nous avons besoin de cinq informations (la position 2D, la vélocité 2D, la valeur d'écume). Or, une texture ne contient que quatre couches d'informations (rouge, vert, bleu et alpha). Pour compresser l'information, nous avons fait le choix d'utiliser un seul canal pour encoder l'information de position. En effet, l'information de position n'est pas fondamentale dans notre calcul et sert simplement à rajouter de la diversité dans l'advection de texture. Compresser deux informations dans un seul canal va donner une précision réduite à chaque information. Si la texture à une précision de 8 bits par canal, c'est à dire 256 niveaux différents, seuls 16 niveaux sont disponibles une fois le canal séparé en deux. Visuellement, la réduction de précision se manifeste par un « snap », c'est-à-dire un alignement de la position de la texture avec la valeur la plus proche.

Pour réduire l'impact visuel du « snap » et même le rendre pratiquement invisible, nous avons mis en place une période de transition avant l'arrêt de la simulation, pendant laquelle la vélocité n'est plus mise à jour, mais la texture d'advection ne change pas de format. Au début de l'arrêt, nous précalculons la position et la vélocité que nous souhaitons stocker dans la texture d'advection une fois la simulation arrêtée. La texture de positions advectées est toujours mise à jour par le processeur principal, mais sans modification de la valeur de vélocité. Pendant ces quelques secondes de transition, nous allons modifier en douceur la position de départ de chaque pixel jusqu'à atteindre la valeur « snappée ». Une fois cette

nouvelle position atteinte, nous pouvons arrêter véritablement toute la simulation sur le processeur principal et passer à la technique totalement GPU, ou la texture de positions advectées n'est plus mise à jour. La période de transition permet ainsi de masquer le passage à une version moins précise de la position de départ. Visuellement, cette transition est invisible tant que les vélocités ne sont pas nulles, car elle est masquée par le mouvement du fluide. Dans le cas de vélocités très faibles ou nulles, un léger mouvement se fait sentir au moment de la transition.

7 Le rendu de l'eau

7.1 La valeur d'écume

Précédemment, nous avons mentionné une valeur d'écume. L'agitation de la surface d'eau créée de l'écume et des éclaboussures qui sont essentielles à la fois pour la crédibilité de l'aspect visuel de l'eau, mais aussi pour améliorer l'impression de mouvement. Cette écume est en effet emportée par les vagues et constitue un important indice visuel du courant et du sens du mouvement. La représentation de l'écume est un sujet très complexe. Elle doit bien sûr être advectée comme la texture de normale des vagues. Elle doit également apparaître lors des chocs entre vagues ou avec les obstacles, puis doucement disparaître. Les crêtes des vagues sont habituellement blanchies par l'écume que le vent soulève. Tous ces effets sont complexes, car ils dépendent fortement du mouvement du fluide, mais sont présents à une échelle de précision bien plus fine que celle du mouvement des vagues.

Figure IV.41: Deux images montrant le mouvement de l'écume sur l'eau.

Nous nous sommes tout d'abord intéressés aux circonstances d'apparition d'écume sur la surface d'eau, à l'échelle d'une rivière par exemple. L'écume traduit essentiellement une agitation de l'eau, un conflit entre les courants, mais perdure ensuite pendant quelque temps en suivant simplement le courant. Le vent est également générateur d'écume, principalement sur la crête des vagues. Nous avons mis en place une détection des situations perturbées dans la simulation, ainsi qu'une détection de la crête des vagues. Malheureusement, la résolution de la simulation est bien trop faible pour permettre l'obtention d'une valeur d'écume à petite échelle. Nous avons obtenu les meilleurs résultats visuels en calculant une valeur de densité d'écume qui augmente en fonction des différences de vélocités entre cellules voisines, puis diminue en continu. Cette valeur est ensuite saturée pour utiliser uniquement la partie la plus

dense, et ainsi limiter artificiellement l'écume à des surfaces inférieures à la taille des cellules de la simulation. Cette valeur d'écume ne sert pas directement dans la modification des couleurs de la surface d'eau, mais comme masque d'une texture représentant une surface d'écume.

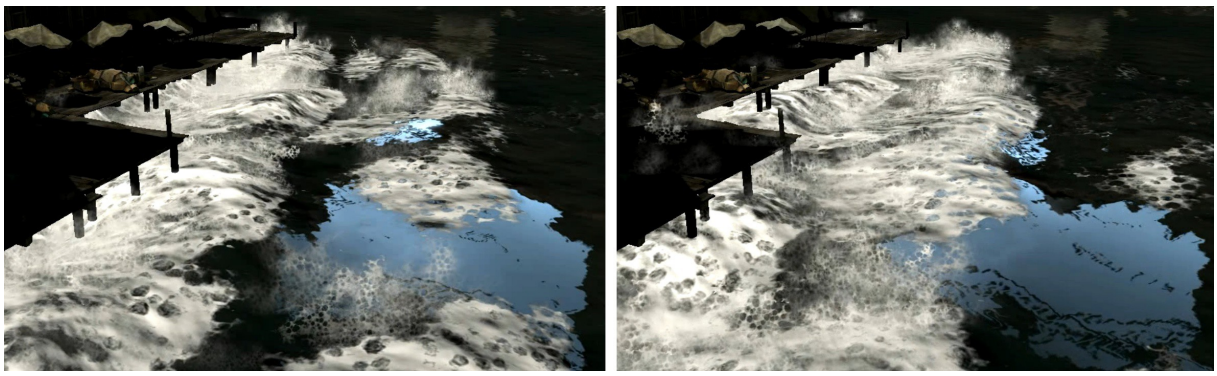


Figure IV.42: L'écume en présence d'obstacles.

La récupération automatique d'une valeur de densité d'écume depuis la simulation de fluide n'est pas très crédible. Nous avons donc mis en place des outils pour ajouter artificiellement de l'écume aux endroits nécessaires. Ces outils sont couplés avec les outils d'agitations de la surface et avec ceux d'apport et d'extraction d'eau. En effet, le mouvement global de l'eau est spécifié par la création de points d'apport ou d'extraction d'eau qui peuvent être placés partout. D'autres outils peuvent être utilisés afin d'agiter artificiellement l'eau selon les besoins des game designers du jeu.

Figure IV.43: Une surface de fluide agitée.

Ces outils, en plus d'influencer la simulation de fluide, ajoutent également de l'écume, qui va être emportée par le courant.

La gestion de l'écume est clairement perfectible dans notre application et pourra constituer un point important de recherche pour les futures améliorations du système.

7.2 Le rendu de la surface

Le rendu de l'eau est bien évidemment très important dans la crédibilité de l'effet, autant que celui du mouvement de la simulation. De nombreuses recherches ont déjà couvert le sujet, voir Jensen et Golias pour un aperçu général[94].

Globalement, la surface de l'eau est partagée entre transparence, réflexion et couleur propre. La transparence induit une réfraction du décor sous marin, ainsi qu'une gestion de l'opacité de l'eau dans sa profondeur. Tous les effets de brouillard de distance sont adéquat pour simuler la dispersion de la lumière dans le volume d'eau. La réfraction du décor s'effectue, dans notre cas, en effectuant l'affichage de la surface d'eau dans une « passe » séparée du reste du décor. L'image de ce décor est alors disponible et peut être indexée au moment de l'affichage de l'eau, avec une distorsion des coordonnées de textures en fonction de la normale de la surface et de la texture de normale advectée.

Figure IV.44: La couleur est constituée par le mélange de la réflexion, de la réfraction, de l'écume et de la couleur de l'eau.

94 L. S. Jensen et R. Golias, « Deep-water animation and rendering », in *Game Developer's Conference (Gamasutra)*, 2001.

La réflexion a déjà pleinement été étudiée dans la partie dédiée. Nous rappellerons simplement que les techniques à base d'une réflexion planaire sont idéales tant que la surface n'est pas trop agitée. Le système à base de cube-maps peut être utilisé aisément soit à travers la création d'une texture bidimensionnelle sans crénelage, soit avec le système de cube-map unique et la compensation de la parallaxe. La réflexion est un aspect fondamental de la surface d'eau et contribue énormément à sa couleur. Pour évoquer une anecdote, notre surface d'eau a paru terne et sans vie pendant longtemps jusqu'à ce que nous ayons la très bonne idée d'ajouter un ciel bleu au-dessus de l'eau. Ainsi, le contraste entre le ciel clair et la réfraction plus sombre a apporté énormément de vie et de crédibilité simplement par le changement de l'environnement extérieur.

Les valeurs d'opacité de la réfraction et de la réflexion sont liées par le facteur de Fresnel. La réflexion s'opère essentiellement aux angles rasants entre la caméra et la surface, alors que la réfraction apparaît au contraire lorsque la caméra est alignée avec la normale de la surface. Cette balance permet également de rajouter de la complexité visuelle en fonction de la normale advectée.

La couleur propre de l'eau n'est pas évidente à définir, car elle joue finalement un rôle assez faible dans la couleur finale, bien qu'elle soit l'un des seuls paramètres visuels sur lesquels nous pouvons avoir une influence simple et directe. Nous avons utilisé des couleurs vertes et brunes pour figurer une eau sale et plutôt opaque. La couleur de l'eau peut être modulée en fonction de sa profondeur, mais également à l'aide de textures de bruit à basse fréquence permettant d'apporter quelques variations à la couleur de l'eau.

L'éclairage direct va principalement influencer la couleur propre de l'eau, la faisant passer d'une couleur brune à une couleur vert clair. La lumière va être essentielle pour l'aspect de l'écume. Elle va donner du relief et de la variété à la surface d'eau lorsque de l'écume est présente.

La prise en compte de la valeur d'écume issue de la simulation va nous servir de masque pour la texture d'écume, qui est advectée en même temps que la texture de normale du fluide. Cette texture d'écume n'est pas pleinement opaque, mais laisse de nombreux trous ou la surface de l'eau sous-jacente sera visible. La texture d'écume, modulée par la densité d'écume, va servir à modifier la couleur propre de l'eau pour la faire passer de son ton verdâtre habituel à un blanc cassé. La réflexion et l'opacité sont également diminuées en présence d'écume.

8 Conclusion

Au cours de cette partie, nous avons présenté les techniques classiques de simulation des fluides. Nous avons délaissé les simulations déterministes, qui ne font pas appel aux équations physiques, mais qui sont très appréciées pour leur stabilité. Nous nous sommes plutôt intéressés aux simulations basées sur la physique et qui sont capables de s'adapter à l'évolution de la scène virtuelle. Plus précisément, nous avons cherché à simuler des étendues d'eau parcourues par des courants, telles que des rivières. Dans ce type de simulation, l'élément important qui est encore délicat à implémenter est celui de l'advection de texture en fonction du courant de la simulation de fluide. Nous avons vu que le transport de cette texture est essentiel dans la crédibilité du fluide.

Notre processus de recherche a suivi une progression en fonction des tests et des prototypes, dont nous avons choisi de décrire les différentes étapes, qui nous semblent essentielles dans la compréhension de l'intérêt de cette technique.

Nous avons ensuite introduit notre technique inédite finale permettant d'ajouter des détails à une surface d'eau. Ces détails suivront le mouvement d'une simulation de fluide à faible résolution. Cette méthode propose plusieurs avantages par rapport aux techniques existant précédemment. Sa simplicité au niveau des calculs à effectuer sur le processeur principal la rend idéale pour l'intégration à une simulation déjà mise en place. Les transferts vers la carte graphique sont légers. Ils peuvent être évités si la simulation de fluide est effectuée intégralement sur la carte graphique. Ses performances sont très fixes et régulières puisqu'elle repose sur la manipulation des valeurs d'une grille, sans que la complexité évolue en fonction des circonstances, comme c'est le cas des techniques à base de particules. La possibilité de stopper les calculs de la simulation tout en conservant un mouvement périodique est également très utile.

Notre nouvelle technique présente ainsi de nombreux intérêts concernant la qualité obtenue et les coûts de calculs. Néanmoins, le résultat devient moins crédible si le fluide est très rapide en comparaison de la précision spatiale du champ de vitesse désiré. La taille des hexagones est alors trop petite pour permettre la perception du mouvement. Envisager des solutions à ce problème est une voie prioritaire pour nos recherches futures. Il pourra s'agir, par exemple, de la superposition de plusieurs échelles de zones d'advection de texture afin de pouvoir s'adapter à la vitesse du fluide.

Nous avons enfin étudié brièvement le rendu de la surface d'eau, délaissant la recherche sur le mouvement de l'eau pour celle sur son aspect dans une image individuelle. L'importance de la réflexion sur la crédibilité de l'image nous a paru évidente, de même que la nécessité d'utiliser une texture d'écume. La gestion de la valeur d'écume est encore trop imprécise pour obtenir un résultat réellement convaincant, spécialement en mouvement. Cet aspect sera essentiel dans nos recherches futures, afin d'améliorer l'aspect visuel de la surface d'eau.

Chapitre V - Le rendu expressif

1 Introduction au rendu expressif

Au cours des précédentes parties, nous avons pu observer l'importance du travail technique en profondeur dans l'évolution de l'esthétique visuelle d'une œuvre interactive. Ce travail devient encore plus fondamental lorsqu'il est question de rendu non-photoréaliste. Ce type de représentation n'a pas pour but premier la reproduction fidèle de la capture du réel, à la manière d'une photographie. Il s'agit bien plus de perpétuer l'héritage de la peinture et du dessin, ou de tout autre style graphique qui possède une expressivité propre au médium.

1.1 L'héritage pictural

L'immense diversité des styles et des objectifs qui forment le domaine des Arts est bien évidemment trop vaste pour être abordée en détail dans cette étude. Nous allons donc simplement citer quelques exemples de peintres dont nous pensons que les styles ont un grand potentiel d'adaptation et d'influence sur le rendu des images numériques. Ces styles ont en commun une vision particulière et subjective de la réalité, ils véhiculent des sentiments et des sensations inhabituelles pour le média vidéoludique. Ils présentent également un grand sens de la représentation du mouvement, de la lumière et de la matière.

Le peintre britannique W. Turner est un parfait exemple de cette recherche picturale, que nous sommes incapables d'approcher dans l'image en temps réel. Ses œuvres mélangent habilement les textures et les couleurs pour atteindre un équilibre délicat entre l'abstraction picturale et l'expressivité de la scène. Le mélange de la vapeur, de l'influence de la lumière et de la matière produit une fusion des éléments, qui ne sont plus discernables individuellement, mais forment un tout cohérent.

D'après Edmond Couchot :

> « Ce que peint Turner, c'est bien la puissance transformatrice du feu, la Machine à Vapeur, dans son principe essentiel, celui de la thermodynamique. Il traduit avec des formes et de la couleur ce que Carnot a énoncé dans le langage de la science »
>
> « C'est la vocation de l'image, sa vocation profonde, de traduire, dans son langage plastique, ce qui «fait sens» aux yeux d'une société ou d'un artiste mais qui relève d'un ordre invisible »[95]

Le peintre arrive ainsi à évoquer des concepts liés à la mécanique et au mouvement dans son œuvre. Il cherche à représenter une réalité personnelle qui ne se limite pas au visuel.

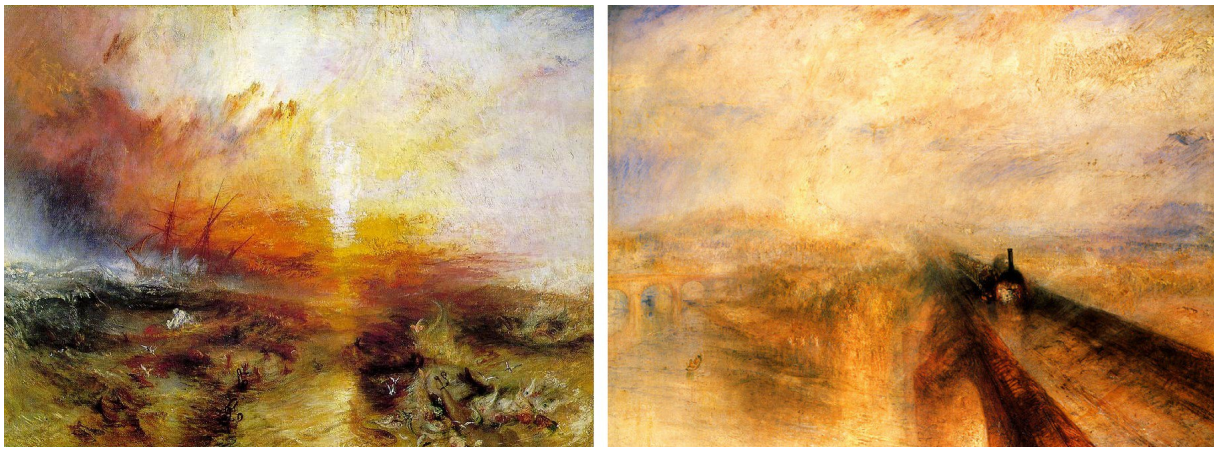


Figure V.1: « The Slave Ship » (1840) et « Rain, Steam and Speed – The Great Western Railway » (1844) de J.M.W. Turner.

Dans un tout autre style et une autre époque, le peintre Georges Rouault délimite fortement des zones de couleurs vives par des traits et des contours sombres très marqués. Sa pratique du vitrail semble transposée en peinture tout en y incorporant la matière et le relief de la toile. Les contours sont à la fois très définis et en même temps suivent un tracé tortueux et imprécis.

95 Edmond Couchot, « Art et technique. L'émergence du numérique », *revue La pensée*, n° 268, collectif Arts de l'ère numérique (1989).

Figure V.2: « Nu incliné » (1937-1938) et « Ne sommes nous pas tous forçats? » (1920-1929) de Georges Rouault.

1.2 La représentation par l'image

Dans l'article « An invitation to discuss computer depiction »[96], Frédo Durand sépare les types de représentations entre les « reflets » et les « images » (« Image » et « picture » en anglais). Les représentations de type « reflet » sont des reproductions ou des imitations d'une scène réelle. Celles de type « image », par contre, sont des schémas, des « descriptions si vivantes et visuelles qu'elles suggèrent une image mentale ou donnent une idée précise de quelque chose »[97].

Frédo Durand nous rappelle également que :

> « la représentation ne consiste pas en une projection d'une scène pour former une image, mais consiste à appliquer des propriétés de la scène à des propriétés de l'image »[98].

La différence est très intéressante, car elle ne limite pas la représentation à un simple reflet qui suivrait les règles optiques, mais concerne toutes les manières d'utiliser les attributs d'une scène pour créer une image.

96 F. Durand, « An invitation to discuss computer depiction », in *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, 2002, 111–124.

97 Websters Merriam, *Webster's Ninth New Collegiate Dictionary*, Revised. (Merriam-webster+ Inc, 1983).

98 « Depiction is not about projecting a scene onto a picture, it is about mapping properties in the scene to properties in the picture. »

Suivant une idée similaire, Edmond Couchot souligne que :

> « Créer une image de synthèse revient à imaginer et à combiner d'abord des modèles, des processus algorithmiques qui dépassent très largement l'univers figuratif et empruntent au champ immense de la connaissance scientifique. »[99]

C'est exactement dans cette idée que les techniques de rendu dites « photoréalistes » et celles dites « non-photoréalistes » se rejoignent et peuvent se croiser. Ces techniques sont essentiellement des moyens d'utiliser des données fournies par les graphistes et de les interpréter pour former une représentation qui permet au joueur de comprendre le monde et d'en apprécier les caractéristiques. Les trois parties précédentes de cette étude ont concerné essentiellement la création de reflets de la réalité, mais nous verrons plus loin que les mêmes sujets peuvent être utilisés dans la création d'une image plus libre et plus expressive.

L'imagerie par ordinateur a longtemps été considérée comme étant l'outil idéal pour représenter et concevoir le monde exactement comme il est dans la réalité. C'est la quête du photoréalisme, qui suppose une connaissance profonde du fonctionnement physique du monde réel, et comporte ainsi de nombreux défis qui dépassent le simple but de la reproduction pour atteindre celui de la connaissance et de la réelle simulation. Au fur et à mesure de l'avancée de l'infographie, les chercheurs se sont rendu compte qu'il était possible, grâce à l'informatique, de travailler l'image bien au-delà des limites du seul réalisme photographique. Les images non-photoréalistes peuvent être beaucoup plus efficaces dans la transmission d'une information, d'un sentiment, d'une idée[100]. Elles peuvent également être plus expressives et même magnifier la réalité en la transformant au travers du prisme d'un style artistique.

Le terme de rendu « non-photoréaliste » n'est pas satisfaisant pour plusieurs raisons. Ce terme fait appel au concept complexe de « réalisme » et à la limite floue et subjective entre le reflet et l'illustration. Pour éviter de définir ce concept uniquement par opposition au photoréalisme, l'utilisation du terme « rendu expressif » commence à devenir populaire parmi les chercheurs. Au contraire du « toon shading » ou « cel shading » bien connu du grand public, ce terme ne se limite pas aux styles de représentations liées au dessin animé traditionnel, mais cherche à

99 Couchot, « Art et technique. L'émergence du numérique ».

100 A. Santella et D. DeCarlo, « Visual interest and NPR: an evaluation and manifesto », in *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, 2004, 71–150.

englober l'ensemble des styles visuels qui possèdent une expressivité qui vient s'ajouter à celle de la scène, telle que les traits d'un pinceau ou d'un dessin. Les médiums traditionnels que sont la peinture et le dessin sont bien évidemment les premières inspirations. Des rendus d'apparence plus neutre tels que le dessin technique font également partie du rendu non-photoréaliste, et par ce biais du rendu expressif.

Le rendu expressif et le rendu dit « photoréaliste » ont plus en commun qu'il n'y paraît au premier abord. En effet, la plupart du temps, lorsque nous parlons d'une image « réaliste », nous parlons en fait d'une image « convaincante », c'est-à-dire qui paraît naturelle et dont la compréhension des formes et de l'espace se fait immédiatement. Pourtant, la création d'une photographie qui parait convaincante, naturelle et compréhensible représente un véritable travail d'interprétation de la réalité. L'ajout d'un objectif de transmission d'informations ou d'émotions complexifie encore le champ des représentations possibles en photographie, faisant entrer la subjectivité du photographe au centre de l'image. La réalisation d'une image convaincante ne peut pas être réduite à un processus mécanique d'enregistrement, mais doit intégrer la suggestivité de l'auteur, jusque dans la création des outils utilisés dans la réalisation. Les images « photoréalistes » et « non-photoréalistes » doivent répondre aux mêmes exigences de contrôle et d'ouverture à la diversité créative de l'artiste. De nombreux indices importants relèvent, dans toute image, du domaine de l’intention de l'auteur et ne sont pas issus de la scène réelle. Parmi eux figure le choix du sujet principal mis en valeur, la construction des masses et des formes dans l'image, et les sentiments que l'artiste souhaite communiquer.

Il est ainsi possible d'utiliser toutes les techniques que nous avons vues précédemment comme des outils génériques de travail de l'image. Ces techniques ne portent pas en elles même des qualités photoréalistes, mais plutôt des qualités plastiques. Elles permettent également de lier diverses informations issues du monde virtuel, de les interpréter, puis de leur donner une forme visuelle. Ainsi, le flou de profondeur de champ utilise l'information de distance des objets pour manipuler l'image. Le travail de la matière et de la réflexion cherche à lier les informations de l'état de surface de la matière (rugosité par exemple) à celles de l'environnement lumineux. Les effets de fluides peuvent évidemment servir à représenter les matières liquides, mais sont également des outils de travail du mouvement qui possèdent une expressivité propre.

1.3 L'influence de la réalité sur le rendu expressif

Le terme de « non-photoréaliste » nous donne à penser que les techniques du rendu « expressif » n'utiliseraient pas les éléments du réel, qu'elles ne chercheraient pas à reproduire des mécanismes du réel. Au contraire, de nombreux styles de rendus dit « expressif » cherchent à reproduire les règles de la réalité, souvent pour les mélanger et les hybrider afin d'utiliser leurs significations et leurs connotations pour acquérir toute leur puissance d'expression. L'exemple le plus parlant est peut être celui des matières, dont nous utilisons les caractéristiques, comme la couleur, l'épaisseur, la rugosité, la transparence, la manière de réagir à la lumière afin de donner une expressivité à l'image. Cette expressivité est utile dans les styles de rendu photo-réalistes, mais aussi dans les styles expressifs.

Les caractéristiques issues du réel peuvent également servir de système conceptuel, qui permet à un artiste de plaquer une signification sur les textures qu'il est en train de peindre ou sur les paramètres qu'il règle. Il est plus simple conceptuellement de subdiviser la création d'une matière en différents composants, tels que la couleur diffuse, la rugosité et la normale, plutôt que de vouloir produire directement la couleur finale de la matière. L'algorithme de rendu expressif peut utiliser ces informations, qui ont un sens et donc une cohérence, et les détourner pour créer un style non photoréaliste, mais qui se base pourtant sur des données pensées d'après les phénomènes du réel.

Il est également souvent question de reproduire les outils d'expression traditionnels (peinture, dessin, sculpture…) en utilisant les règles qui régissent l'interaction entre le pinceau et la toile, entre le geste et le trait, voir même entre le sujet et sa représentation.

Afin de comprendre le lien entre le rendu dit photoréaliste et le rendu expressif, nous devons préciser la différence entre une « simulation » et un « simulacre ». La simulation est une reproduction d'une perception par l'imitation des principes qui sont à l'œuvre. Le simulacre se concentre sur l'obtention d'une représentation par des moyens détournés, sans chercher à reproduire les principes réels. L'acceptation « grand public » du terme « réalisme » est souvent fausse. Le « réalisme » serait normalement une représentation la plus fidèle possible de la réalité perceptible, sans intention graphique particulière et sans idéalisation. Il s'agit de reproduire le réel avec le minimum d’erreurs, ce qui se rapproche de la « simulation ». Le réalisme dans le sens « grand public » est tout autre : c’est la capacité à sembler réel sans forcement l’être. Le réalisme signifie, pour la plupart des concepteurs de jeux, la crédibilité. La différence entre réalisme et crédibilité tient dans le degré de stylisation qui est toléré, mais

aussi dans les bases des algorithmes utilisés. Le réalisme nécessite une simulation, une reproduction des mécanismes réels, alors que la crédibilité s'intéresse plus à l'impact perceptif et sensible pour le joueur.

Traditionnellement, les simulations et les « serious games », comme les simulateurs militaires (pour l'aviation par exemple), ont tendance à utiliser des documents photographiques comme sources et à les modifier le moins possible. Ainsi, la fidélité est importante, car les photos sont des relevés précis du réel. Malheureusement, elles ne couvrent qu'un point de vue de la réalité et ne se complètent pas toujours entre elles. Le résultat est généralement plat, sans cohérence de l'image dans son ensemble et donc pas très crédible. Au contraire, un jeu vidéo va chercher à styliser et rendre compréhensible son univers. La prise en compte de la lumière va notamment apporter une grande crédibilité en plus de permettre d'apporter du contraste à l'image. Les données sources comme les photos seront grandement retouchées afin de pouvoir séparer différentes caractéristiques (diffuse, spéculaire, normale…) et ainsi utiliser des équations qui se rapprochent de la réalité, tout en laissant à l'artiste un contrôle important.

1.4 La frustration du polygone

Une des grandes frustrations qu'apporte le graphisme 3D actuel, surtout en temps réel, est la grande dépendance aux polygones. En effet, tout ou presque passe par le polygone, surface faussement lisse par excellence, forme aux bords acérés et au crénelage facile. Bien sûr, ces dernières années ont apporté des pistes d'espoir. C'est d'abord les textures de normales (normal mapping) et ses outils aux possibilités libératrices (Pixologic ZBrush et Autodesk Mudbox) qui permettent de s'approcher de plus en plus d'une gestion intuitive des formes 3D. Un jour, les moteurs de jeux nous assureront qu'aucun rebord de polygone ne sera visible, à l'instar de l'actuelle image de synthèse précalculée.

En réponse à cette frustration commune à tous les styles, le rendu expressif cherche à remplacer le contour abrupt et binaire du polygone pour y intégrer la signification et l'émotion que peut dégager un objet, un personnage. Plusieurs pistes de travail seront présentées dans cette partie. Le travail de la profondeur de l'image, par des moyens au-delà du flou, sera présenté. Nous étudierons ensuite la piste de l'utilisation de fluides, d'où peut émerger une grande impression de matière et qui apporteront du sens par leur mouvement lié à la physique et de la beauté par l'abstraction de leurs volutes. Enfin, nous étudierons la piste du rendu crayonné, où les contours et les ombres imitent le dessin manuel, avec ses multiples traits qui

parviennent, bien mieux que n'importe quel amas de polygones, à faire ressentir tout à la fois les matières, l'émotion et le mouvement de l'élément qu'ils décrivent.

1.5 La reproduction d'un style

Le style d'un artiste ou d'un courant artistique n'est pas une donnée quantifiable et précise. Chaque œuvre et chaque artiste des arts traditionnels peuvent être considérés comme uniques à la fois au niveau de l'exécution de l'œuvre, mais également dans l'intention qui préside à sa conception. Néanmoins, les outils et médiums utilisés peuvent être modélisés et produisent un comportement cohérent dont les caractéristiques statistiques peuvent être reproduites automatiquement. Le trajet de la main de l'artiste est plus complexe et variable, mais peut également être étudié.

Joshua Seims[101] nous présente une courte analyse des éléments artistiques dans la peinture et des techniques permettant de les reproduire virtuellement. Il insiste sur le rôle majeur de l'artiste, qui doit pouvoir limiter le travail technique et privilégier la création instinctive par le biais d'outils adaptés. Dans la même idée, Barbara Meier[102] nous présente les outils actuels de travail de l'image dans un style expressif, et nous en montre les limites. Des pistes sont également introduites, concernant les possibilités de collaborations efficaces entre l'artiste et l'ordinateur. L'idée d'interpréter et d'enrichir automatiquement les coups de pinceau d'un artiste, de manière interactive, est également très intéressante.

Les techniques de reproduction des traits d'un artiste peuvent être classées en trois catégories :

- création paramétrique
- création manuelle
- création par l'exemple

La création paramétrique s'effectue en utilisant des procédures et des fonctions mathématiques afin de déformer ou de modifier des traits simples. L'utilisation d'un décalage aléatoire ou d'une fonction sinusoïdale par exemple pour déformer l'image des contours extraits de la scène 3D entre pleinement dans cette catégorie.

101 Joshua Seims, « Putting the artist in the loop », *ACM SIGGRAPH Computer Graphics* 33, n° 1 (février 1, 1999): 52.

102 Barbara Meier, « Computers for artists who work alone », *ACM SIGGRAPH Computer Graphics* 33, n° 1 (février 1, 1999): 50.

La création manuelle concerne la production manuelle d'exemples de hachures ou de traits qui vont ensuite être directement dupliqués le long d'un contour que nous souhaitons tracer. Ainsi, tous les traits affichés auront été dessinés à un moment par l'artiste, mais le programme se charge de faire le choix entre les différents exemples de traits pour dessiner chaque image.

La création par l'exemple est une combinaison des deux approches précitées. Il s'agit de fournir des exemples réalisés manuellement, par un artiste par exemple, puis d'exécuter un algorithme qui cherche à les reproduire paramétriquement. Les différentes étapes de cet algorithme consistent à isoler les traits les uns des autres puis d'en extraire des caractéristiques statistiques telles que la longueur, la densité, l'épaisseur, la forme des traits. Ensuite, l'algorithme est capable de produire une infinité de traits différents qui présentent ces mêmes caractéristiques. Ces traits sont ensuite placés automatiquement sur les formes 3D.

Ces trois approches permettent toutes d'obtenir des résultats intéressants, mais qui ne sont pas adaptés aux mêmes usages. L'approche paramétrique nécessite une créativité dans la création de l'algorithme même, puisque la procédure utilisée va directement contrôler le contour final. Tous les styles ne peuvent pas forcément être atteints avec cette technique, mais elle peut s'adapter à n'importe quelle scène 3D sans processus manuel. La technique part généralement de données pseudoaléatoires, telles que celles d'un bruit blanc, qui peuvent être générées facilement en temps réel en fonction de la scène 3D..

L'approche manuelle est la plus directe, car elle permet d'introduire un style en utilisant des outils qui rappellent les techniques telles que le dessin ou la peinture. Ces outils sont très intuitifs, notamment pour les artistes qui ont l'habitude de manipuler les outils de la peinture physique. Le degré de contrôle du style par l'artiste est très fort et direct. Par contre, chaque trait de l'image finale aura été dessiné à un moment donné, ce qui rend le travail fastidieux. Si la qualité désirée est très élevée, il faudra souvent dessiner beaucoup de traits différents et adaptés à un objet précis. Il est bien évidemment possible de reproduire plusieurs fois un exemple donné par l'artiste, mais cette répétition peut être visible, ce qui nuit à l'impression de dessin manuel.

La dernière approche tend à combiner les avantages des deux premières en permettant un dessin manuel et une automatisation de l'application sans utiliser la simple répétition. À partir d'exemples de contours, constitués en général de courbes dessinées à la main, l'algorithme produit un nouveau contour similaire, qui peut alors être appliqué sur une courbe donnée sans aucune véritable répétition. La qualité est ainsi améliorée, mais l'apprentissage est une

problématique complexe, qui n'est pas toujours suffisamment efficace pour permettre de reproduire tous les styles. La thèse de P. Barla[103] présente ce type de solutions, qui prend pour base l'analyse d'images existantes réalisées manuellement, telles que des dessins, et génère une nouvelle image stylisée.

Finalement, la création d'une image intéressante est une symbiose entre une vision artistique et la technique qui va traduire et porter cette vision. Ces deux aspects de la création sont fortement liés et doivent former un tout. Que la traduction de la scène virtuelle cherche la crédibilité ou l'expression artistique, les choix artistiques et techniques font partie de l'image. Edmond Couchot souligne cette dualité :

> « La seule connaissance de ces pratiques et de leur théorie ne suffisait pas, toutefois, à créer «automatiquement» des œuvres dignes d'intérêt. Les peintres devaient être capables d'aller au-delà et faire preuve d'imagination, cette aptitude à concevoir de nouvelles images sans laquelle la plus belle théorie et la plus savante technique ne sont rien. Mais l'imagination d'un peintre, sa pensée vive, dans ce qu'elle a de plus inventif, est-elle totalement indépendante de la technique? Un peintre voit-il sans ses pinceaux, un cinéaste sans sa caméra? »[104]

103 Pascal Barla, « Modèles de représentation et d'acquisition pour le rendu expressif », *Institut national polytechnique (Grenoble).*, Thèse (2006).

104 Couchot, « Art et technique. L'émergence du numérique ».

2 Les techniques classiques

Le rendu typique « cartoon », également appelé « cel shading » est constitué principalement de deux éléments : une stylisation de l'éclairage et une accentuation des contours. Ces deux éléments forment un tout dans le style plastique de l'image, mais les techniques utilisées sont très différentes, nous allons donc les étudier séparément.

2.1 La stylisation de la lumière

Le « cel shading » est un terme anglais dérivé du monde de l'animation traditionnelle sur feuille de celluloïd et signifie littéralement « ombrage de celluloïd ». Il s'agit donc de donner du volume aux objets en utilisant les techniques d'ombrage qui étaient utilisées dans les dessins animés classiques. La caractéristique principale de ce type d'éclairage est l'utilisation d'une palette de couleurs et de niveaux d'ombrage réduite. Bien que ce nombre réduit de couleurs ait probablement été au départ un moyen de diminuer la quantité de travail nécessaire à la réalisation d'une animation, c'est également devenu un style à part entière. Le rendu de scènes 3D, en temps réel ou non, s'est inspiré de ce style et plusieurs techniques classiques ont été développées pour reproduire automatiquement ce type de rendu.

Figure V.3: Le rendu cartoon du jeu « Jet Set Radio ».
À gauche, le rendu avec ombrage de Gouraud, et à droite avec cel-shading.
Image emunova.net

Le but principal est de limiter le nombre de niveaux d'ombrage. Pour cela, nous pouvons bien évidemment utiliser les calculs par pixel (dans le pixel shader) afin d'opérer les opérations de seuillages nécessaires sur le modèle d'éclairage. Cependant, le type de rendu cartoon existait avant que les « pixel shader » ne soient disponibles, lorsque le « pipeline » fixe était la seule

solution pour obtenir une accélération graphique. À cette époque, la technique consistait en un calcul spécial des coordonnées de texture. Celles-ci étaient recalculées à chaque image sur le CPU, pour lui donner la valeur du produit scalaire entre la normale de la surface et la direction de la lumière. Cette opération était donc effectuée pour chaque sommet. La coordonnée de texture servait ensuite à indexer une texture à une seule dimension qui présentait un dégradé de couleur, avec d'un côté la couleur pour les points qui font face à la lumière et de l'autre ceux qui y sont opposés. Le matériel graphique se charge ensuite d'interpoler les coordonnées de textures entre les sommets, en utilisant donc les valeurs de la texture de dégradé. Cette technique est toujours utilisée, mais le calcul des coordonnées peut désormais se faire au vertex shader ou au pixel shader.

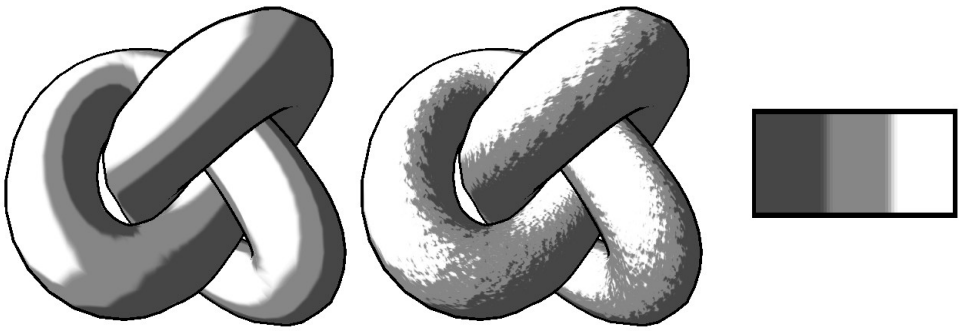


Figure V.4: Cel-shading classique.
Rendu classique (à gauche) et rendu cel-shading au pixel, avec prise en compte d'une texture de normale (au centre). À droite, la texture 1D qui sert à régler le dégradé de lumière.

Modèle issu de RenderMonkey

L'utilisation d'une texture pour stocker les variations de valeurs d'ombrage est une très bonne technique, car elle permet à l'artiste de contrôler librement et intuitivement l'ombrage d'un objet. Il dispose d'une texture de la taille qu'il souhaite et dont il peut découper les zones d'ombrage à volonté et même ajouter des dégradés entre les zones de couleurs unies.

La couleur de cette texture d'ombrage est souvent multipliée avec la texture diffuse de l'objet. Une solution simple pour contrôler précisément le résultat est de conserver l'une de ces deux textures en niveau de gris. En effet, la couleur peut provenir soit de la texture diffuse soit de la texture d'ombrage, mais utiliser un mélange des deux est souvent source de confusion. Chaque objet peut avoir sa propre texture d'ombrage, qui dépend par exemple du matériau représenté.

Pour les matières telles que le verre, les métaux et particulièrement les yeux des personnages, il est nécessaire d'avoir un reflet spéculaire. Le reflet spéculaire est une caractéristique

essentielle dans l'aspect de ces matières et qui aide à la compréhension de l'image par le spectateur. Pour cela, un nouvel appel à une texture peut être fait, en fonction de l'angle de réflexion de la lumière et de la caméra. Dans certains cas, il suffit d'utiliser une direction de reflet fixe par rapport à la caméra, voire même de présenter le reflet toujours face à la caméra, quelle que soit la direction de la lumière, pour obtenir un aspect réflexif suffisant.

Une autre possibilité assez simple est de créer deux textures pour chaque objet, l'une représentant la couleur dans l'ombre, et l'autre dans la lumière. Le mélange des deux se fait en utilisant la valeur de luminosité, qui peut elle aussi utiliser une texture pour contrôler sa valeur.

Il existe des techniques plus évoluées[105] pour la stylisation de l'éclairage et de la réflexion qui repose sur une augmentation du contraste des couleurs en fonction de la forme de l'objet et de la surface.

2.2 Les effets de contours

Nous avons vu que, au-delà du modèle d'éclairage, le style « cartoon » se reconnaît aux traits de contours qui accentuent les bords des objets. Ces contours rappellent le travail des dessinateurs de bandes dessinées et servent à la fois au style plastique, mais également à la lisibilité. Nous allons voir ici différentes manières classiques de produire ces contours.

2.2.a Contours par épaississement du maillage

La première technique consiste à afficher tout d'abord l'objet une première fois, mais en plus épais. Pour cela, deux solutions existent : afficher uniquement les lignes des arêtes de l'objet avec une taille de ligne importante et dans la couleur des contours (cette technique n'est plus vraiment d'actualité sur les cartes graphiques modernes), ou bien utiliser un vertex shader pour « pousser » les sommets de l'objet selon leurs normales au moment de l'affichage. Ici aussi, cette première couche sera dessinée avec la couleur des contours (noir par exemple).

Ensuite, il suffit de dessiner par dessus l'objet normalement pour obtenir une image avec les contours extérieurs de l'objet. Le mécanisme de « push » des sommets consiste à utiliser le vecteur normal qui est stocké avec chaque sommet pour l'ajouter à la position du sommet. La distance de déplacement permet d'obtenir des contours plus ou moins épais. L'épaisseur des

105 Romain Vergne, « Communication expressive de la forme au travers de l'éclairement et du rendu au trait » (2010).

traits peut être ajustée en fonction de la distance à la caméra afin d'obtenir une taille de contour qui paraisse fixe quelque soit la position de la caméra.

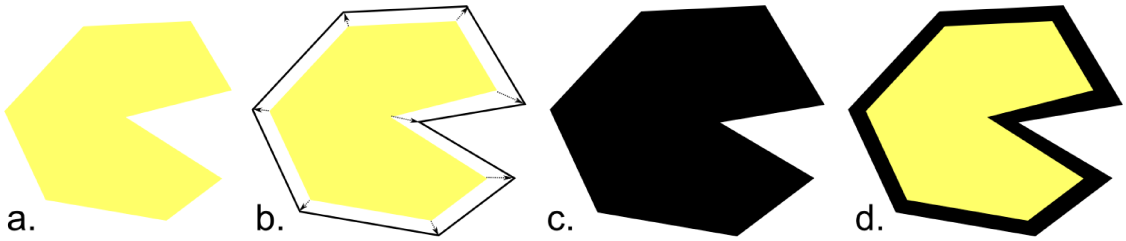


Figure V.5: Constructions des contours d'une forme par épaississement du maillage. La forme à détourer (a.) va voir ses bords s'écarter en fonction des normales des sommets (b.), puis dessinée en noir (c.). La forme originale est ensuite dessinée normalement par dessus (d.).

Cette technique est très efficace, mais nécessite quelques restrictions sur les maillages des objets concernés. En effet, ceux-ci doivent avoir des normales douces, c'est-à-dire qu'un sommet doit avoir la même normale pour toutes les faces dont il fait partie. Dans le cas contraire, la poussée dans la direction de la normale va créer des trous dans le maillage, qui deviennent très visibles si les contours sont épais. Les objets organiques tels que les mains des personnages vont très bien fonctionner, mais une caisse, par exemple, objet cubique par excellence, ne permettra pas d'obtenir des contours corrects avec des arêtes dures. Utiliser des normales douces irait à l'encontre de l'éclairage de l'objet, qui nécessite des bords durs pour figurer les arêtes de la caisse. Les deux solutions à ce problème sont soit de rajouter un chanfrein sur les bords des arêtes dures soit de stocker deux jeux de normales, un pour l'éclairage et le second pour les contours. Le chanfrein, qui est une petite face qui remplace une arête, permet d'avoir une normale différente des deux côtés de l'arête sans utiliser des normales dures.

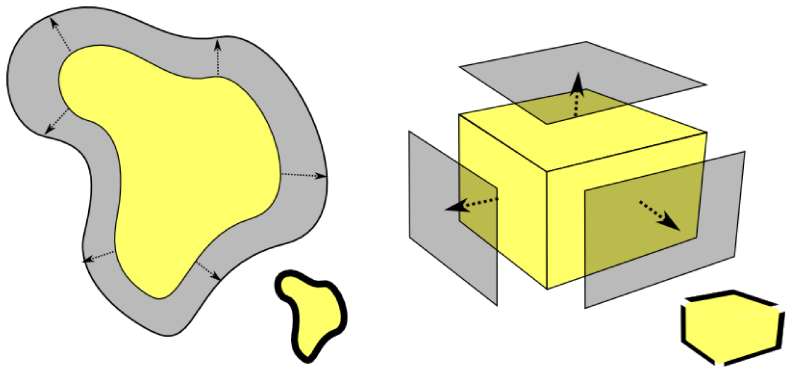


Figure V.6: Les artefacts liés à l'épaississement. À gauche, l'extrusion des bords sur une forme douce produit un contour correct. À droite, l'extrusion des bords sur une forme dure laisse des trous dans le contour.

La technique que nous venons de décrire peut être améliorée pour obtenir les contours intérieurs de l'objet, aux endroits concaves, qui se replient sur eux même. La technique basique ne dessine que les contours extérieurs, la frontière entre l'objet et le décor.

Pour cela, nous allons utiliser le z-buffer. Le dessin de l'objet avec les sommets poussés selon les normales doit être effectué avec du « front face culling », c'est-à-dire que seules les faces dont les normales ne sont pas orientées vers la caméra seront dessinées. L'écriture dans le z-buffer est activée pendant que l'objet est dessiné avec la couleur des contours. Ensuite, l'objet est affiché normalement, avec son éclairage et ses couleurs habituelles, et avec du « back face culling », c'est-à-dire cette fois l'affichage uniquement des faces orientées vers la caméra. Grâce au z-buffer et au « front face culling », l'objet de contour sera dessiné derrière l'objet normal et les différentes parties de l'objet seront triées entre elles, et donc auront des contours.

Toute une scène peut d'ailleurs être dessinée d'un seul coup sans devoir séparer les différents objets à détourer. Par contre, la technique pose problème lorsque la taille des contours est trop grande par rapport à la courbure moyenne des surfaces. Ainsi, les détails dans les visages des personnages sont souvent trop petits par rapport à la taille des contours souhaités. Les recoins de la surface comme les narines ou l'intérieur de la bouche vont être extrudés pour le dessin des contours jusqu'à se traverser les uns les autres, créant des artefacts indésirables.

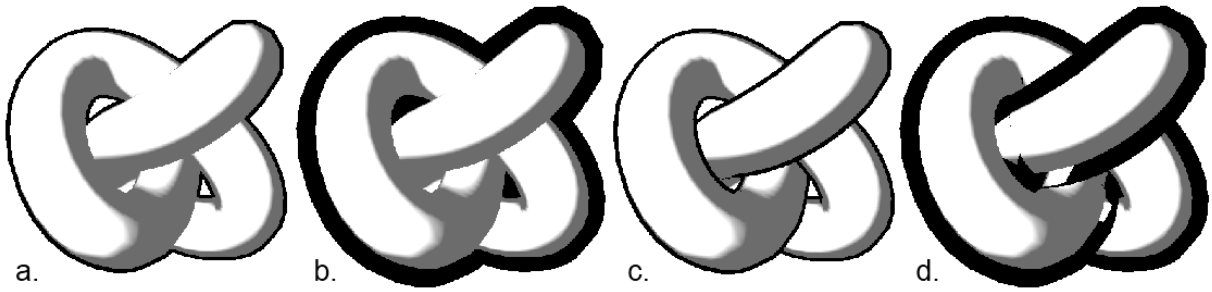


Figure V.7: Contours extérieurs et intérieurs en 3D.
Contours extérieurs d'un l'objet dessiné fin (a.) et épais (b.). Utilisation du Z-buffer pour l'obtention des contours extérieurs et intérieurs de l'objet, dessinés fin (c.) et épais (d.). Les contours intérieurs épais avec la méthode du Z-buffer créent des artefacts indésirables à certains endroits.

Modèle issu de RenderMonkey

Une solution simple à ce problème consiste à repousser les sommets de la passe de contours vers le fond de l'image, loin de la caméra, en manipulant leur coordonnée Z dans l'espace de projection de la caméra. Ainsi, les détails fins de l'objet auront des contours invisibles, alors que les grands dénivelés de profondeurs laisseront passer les contours. La distance de décalage vers l'arrière peut être dynamiquement calculée en fonction de l'éloignement de la

caméra par rapport à l'objet. En conjonction avec une taille de contour dynamique, cela nous permet d'obtenir des lignes simples, sans trop de contours intérieurs, lorsque la caméra est loin, puis de voir apparaître les détails au fur et à mesure que la caméra s'approche. La technique de l'épaississement du maillage est simple et très prisée du rendu typique « cartoon », mais manque par contre d'expressivité. L'échelle du polygone dans laquelle travaille cette technique est généralement bien trop grande pour exprimer un caractère propre. Avec suffisamment de subdivisions, la programmation de la carte graphique permettra de rendre le trait plus ou moins épais selon les endroits et de changer sa couleur. Il sera par contre bien difficile de lui donner l'aspect d'un contour formé de plusieurs traits, à la manière du dessin manuel.

2.2.b Contours au pixel en fonction de la normale de la surface

Une seconde technique consiste à utiliser directement la normale de l'objet dans le pixel shader pour dessiner la couleur du contour lorsque la normale de la surface est suffisamment perpendiculaire à la direction de la caméra. Ainsi, une sphère verra ses bords correctement détourés. Cette technique est utile, mais montre de grandes limites. Tout d'abord, elle ne fonctionne qu'avec des objets organiques, encore une fois, mais à un niveau plus fort que la technique d'épaississement des bords. Ici, il est nécessaire que les courbes des surfaces soient douces et suffisamment subdivisées pour que les contours puissent apparaître.

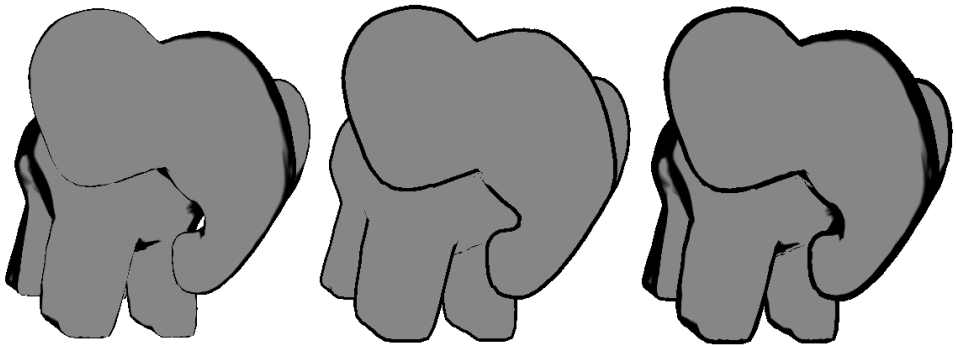


Figure V.8: Comparaison des techniques de dessins des contours. Affichage de contours selon la normale de la surface (à gauche), par épaississement du maillage (au centre) et une combinaison des deux techniques (à droite).

Modèle issu de RenderMonkey

La taille des contours est variable en fonction du degré de courbure de la surface, rendant impossible l'obtention d'une taille uniforme. Les objets tels que les caisses sont pratiquement impossibles à détourer par cette technique, puisque nous n'aurons pas de gradient de la

normale sur les bords de l'objet. Il est possible de combiner les deux techniques, les contours par épaississement du maillage et les contours en fonction de la normale afin d'adoucir la dureté des contours de la première technique tout en assurant une présence constante des contours, même sur les objets aux arêtes dures.

2.2.c Contours par extrusion des arêtes de bords

Une catégorie de techniques de dessin de contours a longtemps été réservée au calcul sur le processeur principal, sans pouvoir utiliser la puissance de la carte graphique. Il s'agit de sélectionner précisément les arêtes qui constituent un bord, c'est-à-dire un repli dans la surface du point de vue de la caméra. Nous pouvons trouver ces arêtes en comparant la direction de la normale des deux faces qui constituent une arête. La direction de la normale d'une face par rapport à la caméra définit si c'est une « front face » (vers la caméra) ou une « back face » (dos à la caméra). Si une face pointe vers l'avant et qu'une face adjacente pointe vers l'arrière, c'est que l'arête entre les deux est un repli dans la surface, donc un bord.

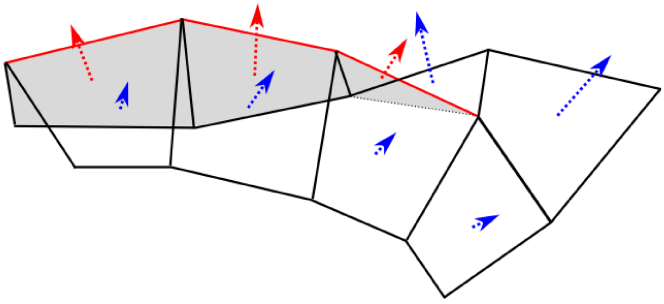


Figure V.9: Extraction des arêtes de contours d'un maillage.
En bleu, les normales des faces qui pointent vers la caméra et en rouge les autres. Les arêtes rouges séparent des faces des deux types et sont donc des arêtes de contours

Une fois toutes ces arêtes sélectionnées, nous pouvons créer un maillage constitué d'un rectangle le long de chaque arête et orienté vers la caméra. L'épaisseur du rectangle détermine l'épaisseur des contours. Cette technique permet d'obtenir des contours très manipulables. Il est notamment possible d'afficher une texture sur les rectangles afin de dessiner des contours moins réguliers, avec par exemple un aspect « coups de crayons » qui ouvre la voie vers des styles de rendus très travaillés.

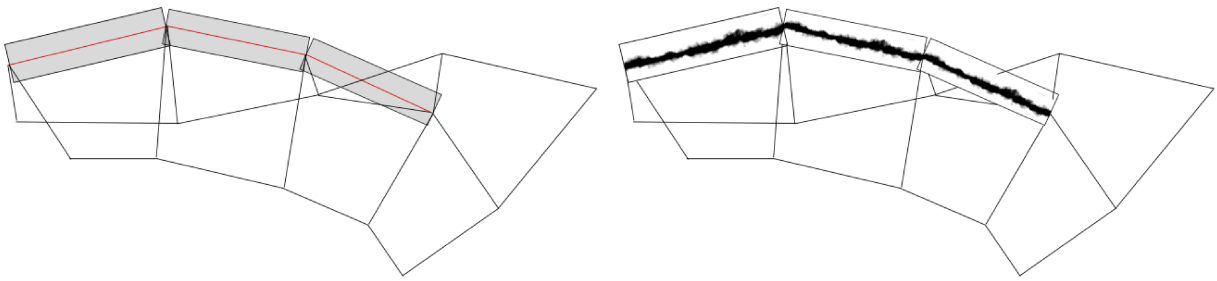


Figure V.10: Placement de rectangles sur les arêtes de contour (à gauche) et placement d'une texture dessinée manuellement sur les rectangles (à droite).

La technique de sélection des arêtes puis d'extrusion est maintenant applicable sur la carte graphique, grâce aux « geometry shader » de DirectX 11. Certaines recherches[106] permettaient déjà d'utiliser la carte graphique, mais de manière plus complexe. Les cartes graphiques récentes peuvent dorénavant intervenir au moment du traitement de chaque triangle, et décider des nouvelles primitives qui seront envoyées à l'étape suivante de traitement, celle des « vertex shaders ». Le « geometry shader » nous permets de comparer les triangles adjacents, de déterminer la présence d'une arête, puis de créer le rectangle de contour et de l'envoyer vers l'affichage. La technique sur le processeur principal pose des problèmes de performance dans la détection des arêtes de bords, qui est coûteuse et nécessite une structure d'accélération pour permettre d'établir rapidement les liens entre sommets, faces et arêtes. À ce propos, un article très intéressant de Fabian « ryg » Giesen, membre du groupe de « demo-maker » Farbrausch, est disponible sur son blog[107]. Il traite de la théorie et de la pratique du stockage efficace des données topologiques par le biais des demi-arêtes (half-edge), ce qui permet d'effectuer rapidement la détection des arêtes de bord.

Il est également nécessaire d'envoyer le maillage de contour ainsi calculé à chaque image depuis la mémoire principale vers la mémoire de la carte graphique, ce qui est lent et occupe à la fois la carte graphique et le processeur principal. Au contraire, la technique récente à base de « geometry shader » n'a pas besoin de transfert d'information avec le processeur principal et effectue tous ses calculs sur le processeur graphique de manière massivement parallélisée. Il est tout de même nécessaire de stocker des informations additionnelles spécifiques dans le maillage, qui nous permettront de retrouver les voisins de chaque triangle. Cet ajout oblige à appliquer un traitement spécial aux objets qui seront détourés. Les informations de connectivités sont par contre nécessaires pour d'autres techniques récentes telles que la « tesselation », qui consiste à subdiviser automatiquement le maillage d'une surface afin d'ajouter du détail en fonction de l'éloignement de la caméra. Il est donc probable que les moteurs graphiques des jeux futurs fourniront ces données dans tous les cas et qu'ainsi, l'utilisation de contours par extrusion des arêtes sera très simple à mettre en place.

2.2.d Contours par filtre de détection

La dernière famille de techniques de dessin des contours que nous allons étudier dans cette partie est celle de la détection bidimensionnelle des contours par l'application d'un filtre.

106 M. McGuire et J. F. Hughes, « Hardware-Determined Feature Edges », *ACM* (2004).

107 Giesen Fabian « ryg », « The ryg blog », 2012, http://fgiesen.wordpress.com/.

Le filtre « sobel »

La situation la plus simple pour étudier ce cas est de produire l'image finale avec des couleurs franches, puis d'appliquer par dessus un filtre de détection de contours comme le « sobel ». Un tel filtre travaille sur une fenêtre d'un certain nombre de pixels et calcule le gradient de couleur. Si celui-ci est très fort, c'est qu'il y a une variation de couleur brutale et donc que nous sommes en présence d'un pixel de bord. Le filtre de « sobel » s'applique en deux passes successives, chacune détectant les contours selon un des axes de l'image. Une fois le filtre appliqué, l'image contient les contours.

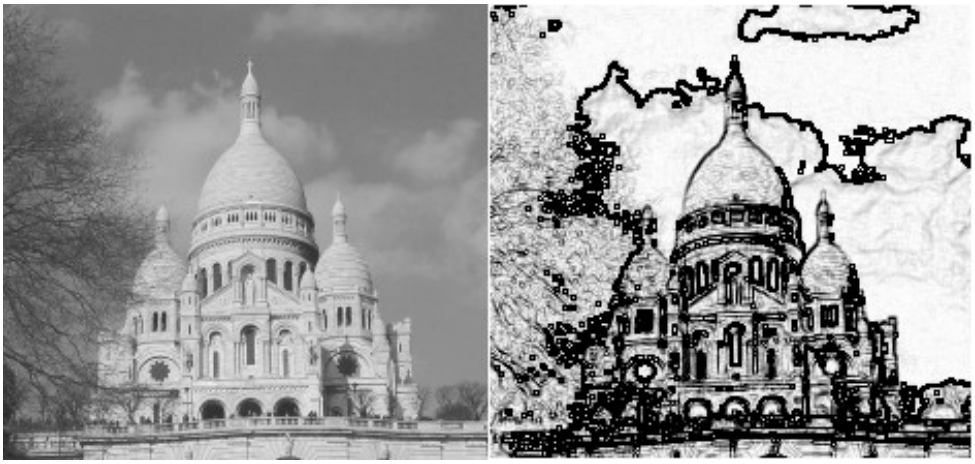


Figure V.11: Exemple d'application du filtre de sobel sur une photographie
Image Wikipedia (Greudin)

Cette technique à l'avantage d'être orthogonale à la création de l'image, car elle travaille sur l'image finale et ne se préoccupe pas des techniques de création de celle-ci. Pour améliorer le résultat obtenu, nous avons plusieurs techniques dérivées qui utilisent d'autres informations que la couleur de l'image finale pour effectuer la détection de contours. En effet, si nous utilisons directement les pixels de l'image finale, les contours n'apparaîtront qu'aux endroits où la couleur varie brutalement. Impossible alors d'obtenir les bords intérieurs d'un objet de couleur unie.

Utilisation de la profondeur et de la normale

La solution est d'utiliser les images de profondeur (z-buffer) et de normales (orientation de la surface en espace-écran ou en espace-monde) dans la détection des contours. Ainsi, nous ne nous intéressons plus aux variations dans la couleur de l'image, mais dans la profondeur et dans la normale. Le gradient de profondeur nous permet d'obtenir des contours sur les bords des objets, mais pas aux points de contact entre deux surfaces. C'est pour cette raison que

nous utilisons également la texture de normale, car celle-ci nous permet de détecter les changements brutaux d'orientation des surfaces, tels que les points de contact entre un objet et le sol par exemple, qui sont indétectables par une différence de profondeur.

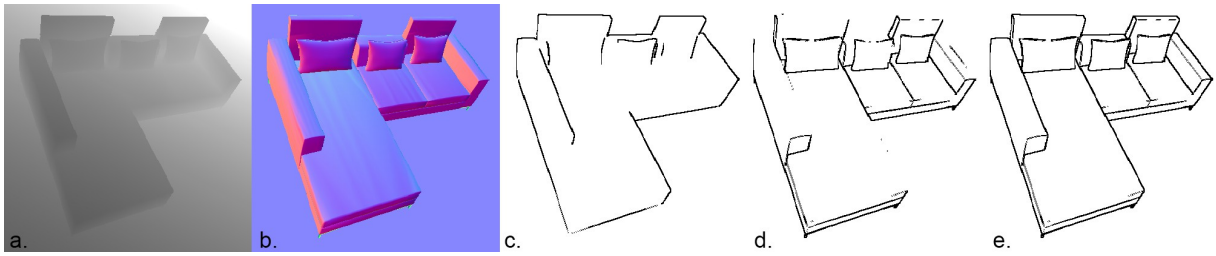


Figure V.12: Filtre de détection de contours.
Image de profondeur (a.) et de normale (b.) et les détections de contours respectives (c. et d.)
Combinaison des détections avec profondeur et avec normale (e.).
Modèle 3D de R.Taylor - archive3D.net

Grâce à ces deux textures, la détection de contours est très robuste et s'adapte à la plupart des situations. Le calcul de contours plus épais qu'un pixel nécessite l'utilisation d'un filtre avec une fenêtre d'action plus grande, donc plus lourde, ou bien les applications successives d'un filtre d'épaississement des contours. Le filtre d'épaississement s'occupe d'effectuer une expansion des contours. À chaque passe, un pixel normal devient un pixel de contour si l'un de ses voisins fait déjà partie du contour. Chaque passe agrandit le contour d'une taille de deux pixels (un pixel de chaque côté du contour).

Utilisation de couleurs unies

Pour aller plus loin, il est également possible de produire une image en couleur qui sert uniquement à la détection de contours. Dans cette image, chaque objet sera dessiné avec une couleur uniforme, mais différente des autres objets. La détection de contours dessine alors parfaitement les contours extérieurs des objets. Cette technique à l'avantage de pouvoir dessiner des lignes plus épaisses qu'un pixel avec seulement une passe. Pour cela, à la place d'utiliser un filtre sobel nous allons simplement échantillonner la texture de couleurs uniformes à trois endroits : la position du pixel calculé, un échantillon décalé sur la droite et un échantillon décalé vers le bas. Plus le décalage est grand, et plus le contour sera épais, pour un coût de traitement constant. Le contour ne sera pas parfait, notamment aux « coins » des objets, qui ne seront pas détectés convenablement, ou bien pour les objets trop petits pour la taille du filtre. Pour une taille de contour raisonnable, de quelques pixels, cette technique est très efficace.

Nous pouvons également utiliser des dégradés doux afin de détecter les contours intérieurs des objets[108].

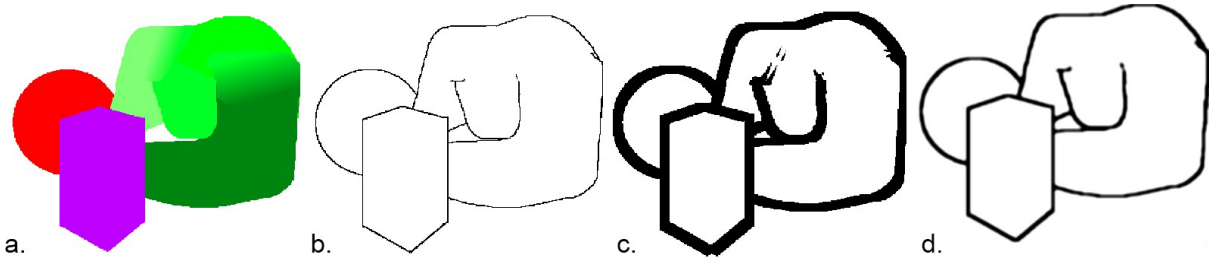


Figure V.13: Extraction des contours à partir d'un rendu spécifique.
Le rendu, constitué essentiellement d'aplats et de dégradés (a.). Extraction de contours d'un pixel de large (b.) et de 8 pixels de large (c.). La version à 8 pixels de large présente des artefacts aux coins des objets. Les contours peuvent être adoucis par l'application d'un léger flou gaussien sur des contours de 2 pixels de large (d.).

Faire varier le contour

L'ajout de détails tels que des variations d'épaisseurs, une ondulation du trait, la subdivision du contour en petits traits irréguliers se fait généralement en utilisant une texture de variation placée en 2D sur l'écran. Cette texture fournit du détail qui n'apparaîtra qu'aux endroits où sont présents les contours. L'inconvénient majeur de cette technique est sa grande dépendance à l'espace-écran, c'est-à-dire à la position de la caméra. Ainsi, si la caméra se déplace, les pleins et déliés, le grain, les déformations, vont se déplacer avec la caméra, donnant un effet qui n'est pas agréable à l'œil et perturbe le sens initial des traits[109]. Ce problème est appelé généralement « rideau de douche » (shower door en anglais) en référence à la paroi granuleuse des vitres de douches, qui déforme la lumière par réfraction, mais dont les ondulations suivent le mouvement de la porte, et non ceux du décor derrière. Les informations bidimensionnelles que nous utilisons sont plaquées sur l'écran. Les moindre mouvements de la caméra où des objets vont révéler le manque de corrélation entre les détails plaqués sur l'écran et la spatialité de la scène 3D.

Pour cette raison, les détails sont souvent modifiés à chaque image, ce qui masque effectivement le manque de corrélation, mais impose un scintillement permanent des contours. Cette solution fonctionne bien si les détails sont légers, comme du grain doux qui perturbe finement le trait, mais est trop perturbante pour des modifications plus profondes de l'image. Celles-ci ont besoin d'une continuité temporelle d'une image sur l'autre afin que l'œil

108 Alex Harvill, « Effective toon-style rendering control using scalar fields », in *ACM SIGGRAPH 2007 sketches*, SIGGRAPH '07 (New York, NY, USA: ACM, 2007).

109 V. Ostromoukhov, « Survol de techniques de rendu non-photoréaliste (NPR) » (2000).

puisse avoir le temps de les interpréter. La modification des détails à chaque image se fait par exemple par un décalage 2D tiré aléatoirement des coordonnées de texture utilisées pour les détails.

Plusieurs détections de contours peuvent être additionnées avec un décalage différent en espace-écran afin d'obtenir des contours en plusieurs traits et plus complexes, et l'ajout d'un flou gaussien puis d'une augmentation du contraste adoucit les traits dans leur forme et dans leur trajectoire.

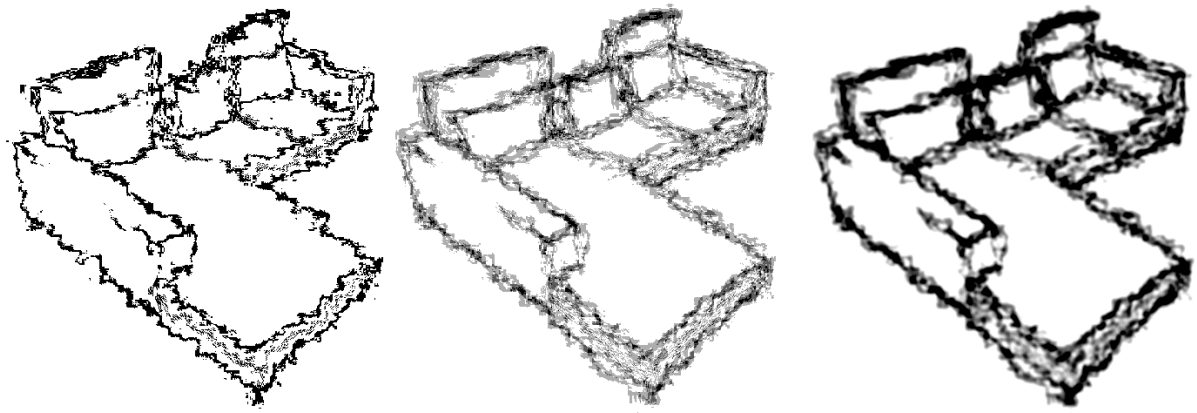


Figure V.14: Application d'une déformation sur les contours.
De gauche à droite : détection des contours avec déformation en espace-écran, moyenne de trois détections avec des décalages différents, ajout d'un filtrage gaussien et amplification du contraste pour un effet « encre de chine ».

Modèle 3D de R.Taylor - archive3D.net

Plusieurs recherches s'intéressent à la création procédurale de textures, telles que des textures de bruits[110]. Le but est d'obtenir un bruit bidimensionnel qui s'adapte aux évolutions dans le temps. Ainsi, le bruit reste attaché aux objets 3D au cours de leurs mouvements. Afin de compenser l'étirement de cette texture de bruit au cours du temps, le bruit se renouvelle régulièrement. Les résultats sont bons, mais se limitent à des effets procéduraux tels que du grain ou une distorsion 2D. Les hachures et les autres styles nécessitant une direction d'application sont plus complexes et fonctionnent moins bien. La thèse de P. Bénard[111] décrit notamment l'utilisation du « gabor noise » pour produire un bruit plus complexe tout en conservant une cohérence temporelle et une capacité à s'adapter à la distance de vue pour toujours produire un même style.

110 Michael Kass et Davide Pesare, « Coherent noise for non-photorealistic rendering », in *Proceeding SIGGRAPH '11* (ACM SIGGRAPH, 2011), 1.

111 Pierre Bénard, « Stylisation temporellement cohérente d'animations 3D basée sur des textures », *Université de Grenoble*, Thèse (2011).

3 Les techniques modernes du rendu expressif

3.1 WYSIWYG NPR : un outil de travail du rendu expressif

Nous avons déjà discuté de l'importance d'impliquer les artistes dans la création du style de rendu. Il s'agit également d'être capable de manipuler le graphisme expressif d'une scène de manière interactive, en obtenant un retour direct de nos actions et de nos décisions. Le projet WYSIWYG NPR[112] des universités Princeton et Brown suit exactement cette idée, en proposant un outil dans lequel il est possible de peindre directement sur les objets 3D d'une scène et le logiciel adaptera cette peinture lors des changements d'angles de caméra. Pour cela, les opérations de dessin sont regroupées en quatre sections. La première s'occupe des contours et des frontières des objets. La seconde est dédiée à la représentation des creux et des bosses dans la surface. La troisième concerne les hachures qui représentent les parties ombrées des objets. Enfin, la dernière est celle des traits qui ne correspondent à aucune caractéristique de la surface, mais sont peints sur l'objet. Ils sont vus comme des « décalcomanies » plaquées sur la surface.

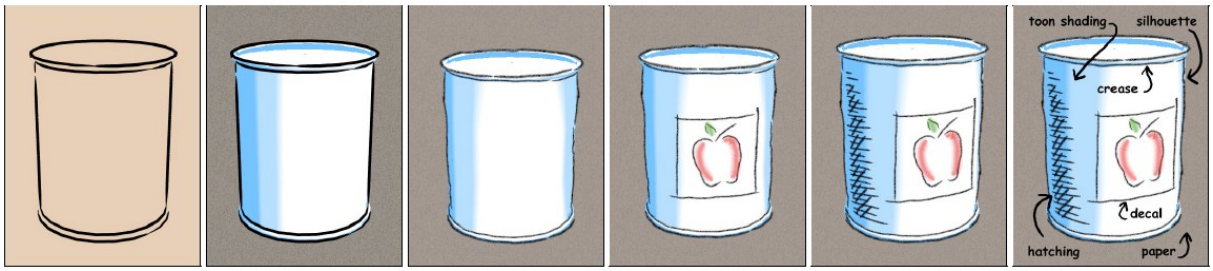


Figure V.15: La technique de WYSIWYG NPR.
De gauche à droite : Modèle avec contours basiques, ajout d'un éclairage, stylisation des contours, ajout d'une « décalcomanie », ajout des hachures et résumé des caractéristiques.

En plus de ce travail des traits, il est également possible de manipuler directement l'éclairage de la surface, afin d'obtenir un dégradé de couleur en fonction de la lumière qui va servir de fond derrière le dessin. Les traits des contours sont simplement créés en dessinant sur l'un des bords. Le contour est ensuite automatiquement adapté aux autres bords. Il est possible d'ajouter autant d'exemples de contours que voulu pour améliorer le résultat. Les hachures sont également dessinées directement sur l'objet, et le programme appliquera une variation de la largeur des hachures, ainsi que du nombre de hachures afin de conserver le même ton

112 R. D. Kalnins et al., « WYSIWYG NPR: Drawing strokes directly on 3D models », *ACM Transactions on Graphics* 21, nº 3 (2002): 755–762.

moyen en fonction de l'éloignement de la caméra. Si elles sont loin de l'objet, certaines hachures disparaissent et dans le cas contraire, de nouvelles hachures apparaissent entre les hachures peintes.

Figure V.16: Exemples de styles graphiques atteignables grâce au projet WYSIWSG NPR.

Il est également possible de peindre manuellement les différents niveaux de hachures en fonction de l'éloignement. L'ensemble des outils du projet est présenté dans une interface commune qui permet de mettre l'outil dans les mains d'un artiste, sans nécessiter d'interventions extérieures d'un programmeur et sans opérations complexes. Un outil[113] dérivé de ces recherches a été mis à disposition de tous, ce qui est une très bonne initiative. Il est ainsi possible d'expérimenter soit même la création de styles graphiques à l'aide de cet outil. Cela donne également une seconde vie au projet, hors du domaine réservé du laboratoire et des publications. Une thèse de R.D. Kalnins sur le sujet est également consultable[114].

Les résultats obtenus sont très intéressants et permettent effectivement de créer très simplement un style de rendu expressif qui s'adapte automatiquement lors des changements de caméra et de l'animation des éléments de la scène 3D. Néanmoins, certains éléments ne sont pas développés par cette recherche, comme les ombres portées. De plus, l'utilisation d'une texture de papier ou de toile pour ajouter de la matière et du grain à l'image est limitée pour eux à l'espace-écran, et donc glisse sur les objets, sans proposer de cohérence avec l'espace 3D et les mouvements de la caméra.

3.2 Journey

Le jeu vidéo « Journey » est sorti en 2012 sur le PlayStation Network. Le studio de développement « Thatgamecompany » nous propose une œuvre majeure dans l'utilisation du

113 « j o t », s. d., http://jot.cs.princeton.edu/.

114 R. D. Kalnins, « Wysiwyg npr: interactive stylization for stroke-based rendering of three-dimenisional animation » (2004).

média vidéoludique en tant que forme d'art. Le parti pris graphique est pour beaucoup dans l'expérience de jeu et soutient à la fois la narration et l'expérience humaine qui sont les principaux points qui distinguent « Journey » des autres jeux. Nous allons donc analyser ce jeu et tenter de comprendre ce qui le rend intéressant.

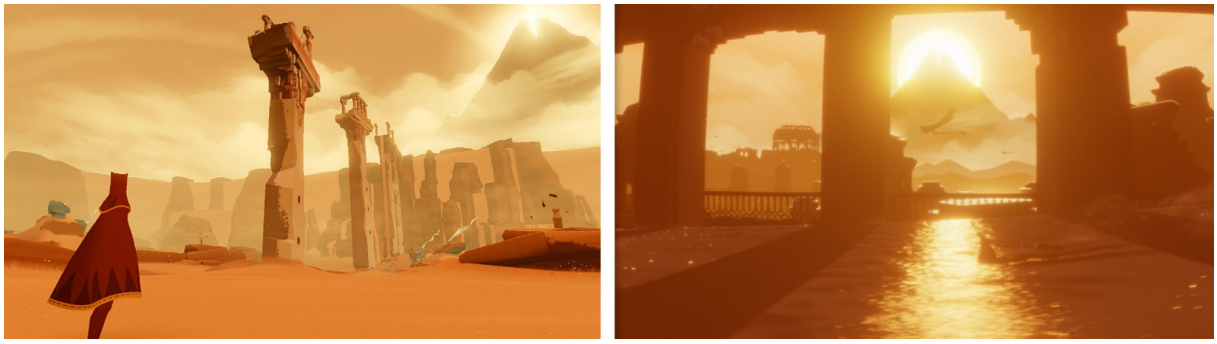


Figure V.17: Le jeu vidéo « Journey » (2012)

Le précédent jeu du même développeur et du même directeur créatif, Jenova Chen, préfigure certains des thèmes abordés dans « Journey ». Il s'agit de « Flower », une expérience dans laquelle nous incarnons le vent, qui souffle des pétales de fleurs pour progresser dans les décors et leur redonner vie. Déjà, très peu de textes sont utilisés, et la narration passe essentiellement par des indices visuels et par l'évolution des émotions ressenties chez le joueur. L'aspect visuel privilégie les contrastes entre couleurs, les ambiances fortes et en adéquation avec la narration implicite du jeu. L'essentiel de l'expérience repose ainsi sur la découverte et l'exploration de différentes atmosphères et l'émerveillement devant les modifications que le joueur peut y apporter.

Dans « Journey », le joueur incarne directement un personnage, perdu au milieu du désert. Celui-ci est à la fois très identifiable et curieusement neutre. Son visage est symbolisé par deux yeux lumineux, tout son corps est enveloppé dans une robe. L'ambiance du jeu repose sur cet équilibre entre simplicité et complexité. La simplicité est présente dans l'utilisation d'aplats de couleurs alors que les effets de brumes, de vents de sable ou de neige ajoutent une touche de complexité visuelle.

Le jeu propose une expérience multijoueur innovante sur plusieurs points. Tout d'abord, aucun choix n'est proposé au joueur quant aux personnes qui l'accompagneront dans l'aventure. Au contraire, la plupart des joueurs sont véritablement surpris lorsqu'ils font la rencontre de leur compagnon de voyage et qu'ils se rendent compte que ce n'est pas une intelligence artificielle, mais un humain. Le second point innovant de l'expérience de jeu consiste en une absence

quasi totale de moyens de communication entre les deux joueurs. Ils ne peuvent que se déplacer ou émettre de petits sons abstraits. Grâce à ce manque d'expression directe des joueurs, l'accent se place sur le non-dit, sur la suggestion et sur l'exploration de l'univers de jeu. En résumé, aucun élément extérieur n'est autorisé à perturber l'expérience esthétique et émotionnelle des joueurs, pas même la culture personnelle des participants.

La première technique graphique employée dans ce jeu est celle de la simplification des formes, des ombres et des textures pour privilégier l'utilisation d'aplats de couleurs et de dégradés. Ensuite, nous pouvons remarquer l'utilisation d'une simulation de fluide. Elle sert principalement pour les dunes de sable, qui se creusent sous les pas du personnage et sont entraînées par le vent. Les fumées et les effets de particules sont également des utilisations de simulations de fluides plus ou moins complexes. Ces simulations apportent de la complexité à l'image et surtout à son mouvement, tout en conservant un sens et un naturel pour le joueur, qui est habitué à ce type de mouvements issus de la réalité. Enfin, les matières utilisées présentent une apparence simple au premier abord, mais se révèlent par moment impressionnant de qualité et d'originalité. Elles proposent par exemple une réflexion spéculaire du soleil.

« Journey » possède plusieurs points communs avec l'expérience que nous avons menée sur le rendu expressif, et est pour nous un parfait exemple de l'application de techniques singulières visant à l'obtention d'un rendu de l'image intéressant, identifiable. Ce rendu est également adapté à la transmission de la narration souhaitée par les développeurs, c'est-à-dire sans texte, uniquement par les changements visuels dans l'ambiance et par les connotations suggérées par les mouvements des personnages, des décors et des créatures.

3.3 Okami

Le jeu vidéo « Okami » (2006) du studio Clover est inspiré des œuvres de Katsushika Hokusai, peintre japonais du début du 19ème, et du courant artistique qu'il représente, le ukiyo-e, un style d'estampes japonaises.

Figure V.18: Les dessins d'ambiance du jeu vidéo « Okami » (2006)

Deux aspects sont intéressants ici. Tout d'abord d'un simple point de vue technique, le jeu vidéo Okami est un des premiers jeux vidéo avec un rendu expressif aussi prononcé et singulier. Les techniques utilisées sont originales et le rendu est très particulier. Le titre se démarque du point de vue visuel, particulièrement au niveau de l'aspect encré de l'image, dans le style du ukiyo-e. L'univers est constamment en mouvements, les bords noirs ondulent, nous pouvons observer des effets de coulures (par exemple lorsque le joueur utilise le pinceau dans le jeu), et beaucoup d'effets de fumées lors des apparitions et disparitions des monstres du jeu.

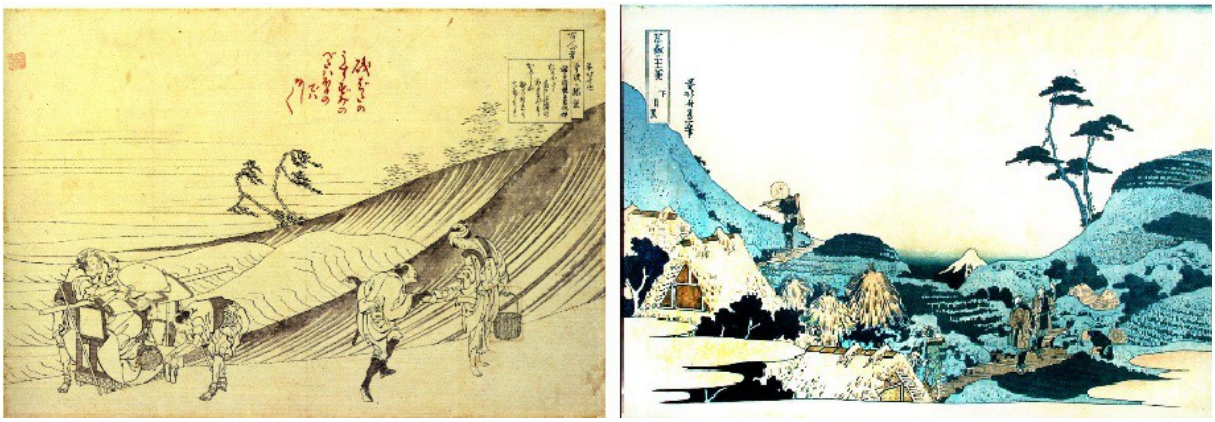


Figure V.19: Deux peintures de Katsushika Hokusai

4 Création de techniques expressives personnelles

4.1 La profondeur de champ : l'effet « rêve »

Au cours du développement d'un prototype de jeu pour la société DONTNOD, nous avons eu besoin d'un effet particulier. Le but principal était de pouvoir présenter la même scène avec deux ambiances visuelles différentes. En effet, le prototype se décomposait en deux parties, l'une se passant dans la « réalité » d'une scène virtuelle, et l'autre dans le souvenir de cette « réalité ». La différence devait être sensible directement par l'image. Les connotations et sensations qui accompagnent le « souvenir » et les liens avec l'ambiance des rêves nous ont mis sur la piste des techniques de flous. Les souvenirs sont rarement parfaits, ils comportent des parties plus détaillées sur lesquelles le « rêveur » se concentre alors que des pans entiers de la scène peuvent n'être qu'esquissés. La personne ne remarque pas ces zones vides, car elles n'ont aucune importance dans le déroulement de l'action.

4.1.a Lien entre le « rêve » et la profondeur de champ

Le lien avec le flou de profondeur de champ est assez logique. Les deux effets consistent à rendre floues certaines parties de l'image afin de concentrer le regard et l'attention du spectateur. L'aspect plastique est également important pour la réussite de l'ambiance « rêve », notamment par le travail du flou. Les deux effets, rêve et profondeur de champ, ont donc été couplés à la fois pour les raisons de similarités déjà évoquées, mais aussi pour des raisons de performance. Nous utilisons le principe de la profondeur de champ en rendant plus ou moins flou un élément en fonction de sa distance à la caméra. Le plan de focus est mis automatiquement sur les éléments qui sont considérés comme importants, notamment les personnages, et le reste de l'image est dominée par le flou et l'effet de souvenir.

La déformation de l’écran, dans le cadre de l’évocation d’un effet « souvenir », est souvent utilisée dans le domaine du cinéma ou du jeu vidéo. Ces effets utilisent des flous dans certaines parties de l’image ainsi que des déformations plus ou moins prononcées. Le principal reproche que nous pouvons faire à ces effets est qu’ils manquent de cohérence temporelle et spatiale. En effet, lors des rotations de caméra par exemple, ces effets ont tendance à rester accrochés à la caméra, au lieu de suivre le décor. C'est l'effet « rideau de douche » dont nous avons déjà parlé. Le but principal de la recherche a donc été d’améliorer

la stabilité de ces effets par rapport à la caméra, en développant des techniques en adéquation avec le temps réel.

Figure V.20: Utilisation de l'effet « rêve » pour un prototype.

Notre effet est constitué essentiellement de déformations bidimensionnelles de l'image de la scène. Il s'agit donc d'un post-process pur. Afin que ces déformations puissent garder une cohérence lors du déplacement de la caméra, nous devons la rendre dépendante de données qui sont extraites de la scène 3D. Ces déformations devront également ne pas présenter de répétitions temporelles ou spatiales trop visibles. Pour cela, les objets utilisent leur position pour calculer la direction de déformation. Les bords des objets, c'est-à-dire les discontinuités dans l'image de la position vue depuis la caméra, peuvent par contre poser problème. En effet, les deux côtés d'un même bord appartiennent à des objets et donc des positions différentes. Les deux côtés n'auront donc pas la même direction de distorsion ni le même mouvement. Nous verrons, dans la partie sur notre technique de contours texturés, différentes pistes de résolutions de ce problème.

4.1.b Création d'une texture pseudovolumétrique

Pour faire varier la direction de déformation en fonction de la position dans l'univers 3D, le plus simple serait d'utiliser une texture volumétrique 3D qui contiendrait des valeurs de directions plus ou moins aléatoires. Malheureusement, le moteur utilisé ne gère pas efficacement les textures 3D. De plus, la taille mémoire que prendrait une texture 3D de qualité suffisante pour nos besoins serait trop importante. Une texture 3D de $256 \times 256 \times 256$ pixels (en RGBA8) représente 64 Mo de données brutes.

Pour pallier cela, nous avons mis au point une technique qui utilise une texture 2D contenant un bruit (par exemple un bruit de Perlin) différent sur chacune des 4 composantes RGBA

(Red, Green, Blue et Alpha). Trois appels différents sont ensuite faits en projetant l'espace 3D sur les 3 plans de bases : XY, YZ et ZX. Les trois résultats sont ensuite additionnés puis normalisés, et les canaux Red et Green sont utilisés pour indexer une dernière fois dans une texture de bruit. Ce système permet de minimiser les artefacts visuels sur les trois axes. Chaque axe contribue au choix des coordonnées finales, et le caractère non linéaire du dernier appel à une texture brouille les pistes et masque l'artificialité de l'algorithme.

Dans notre cas, nous ne possédons pas l'information de normale de la surface au moment du calcul de l'effet. Si celle-ci est disponible, il est possible d'interpoler entre les trois appels liés aux trois plans en fonction de la normale. L'idée est d'utiliser le plan de projection de la texture qui est le plus orthogonal à la normale de la surface, puisque c'est le plan qui présentera le moins de déformations et d'étirements.

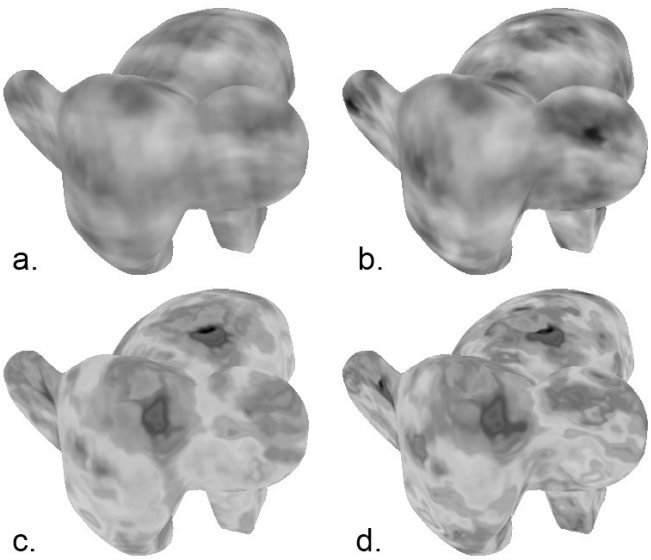


Figure V.21: Génération d'un bruit 3D à partir d'une texture 2D.
a. : Somme normalisée de trois appels selon les axes XY, YZ et ZX. Des lignes indésirables sont visibles.
b. : Somme normalisée des trois appels pondérés selon la direction de la normale pour minimiser les déformations. Les artefacts ont disparus.
c. : Utilisation des axes XY du résultat de (a.) pour échantillonner une nouvelle fois la texture. Les artefacts sont invisibles.
d. : Même opération mais avec le résultat de (b.) Les résultats c. et d. sont qualitativement très similaires.
Modèle issu de RenderMonkey

La valeur donnée par cette texture pseudovolumétrique va permettre de moduler l'intensité d'une déformation sinusoïdale bidimensionnelle placée sur l'écran. L'association de ces deux mouvements, l'un spatialisé en 3D et l'autre en 2D sur l'écran, va à la fois conserver une cohérence spatiale, et en même temps donner de la diversité et du contrôle sur le résultat final. La fréquence de variation de la direction et l'amplitude peuvent ainsi être fixée en espace-écran. Les différentes textures utilisées peuvent être déplacées en fonction du temps afin d'apporter du mouvement.

4.1.c Liaison entre la déformation et le flou

En plus de la déformation, les éléments de l'image vont devenir flous. L'intensité du flou va être directement reliée à l'intensité de la déformation. Ainsi, en même temps qu'une partie de l'image commence à se déformer à cause de l'effet, elle va également devenir floue. L'algorithme de profondeur de champ est donc modifié pour prendre en compte également le facteur de déformation dans le choix du degré de flou à appliquer. La saturation et la colorimétrie des parties déformées peuvent également être modifiées. Selon les paramètres, nous pourrons obtenir des effets proches d'une ancienne photographie (tons sépias), ou bien d'une sorte de carbonisation de l'image si les couleurs sont saturées ou encore d'une scène immergée dans l'eau. Dans ce dernier cas, le mouvement de l'image rappelle fortement celui d'un fluide.

Figure V.22: L'effet "rêve" déforme et rend flou différentes parties du décor en fonction du temps.

Grâce à cet exemple, nous avons pu voir une extension possible de l'effet de profondeur de champ. Le temps réel nous offre les moyens de naviguer librement dans l'espace et de jouer avec ce genre d'effets pour essayer, en tant que joueur, d'en comprendre les principes.

4.2 Intégration des fluides dans le rendu expressif

Dans le chapitre sur les fluides, nous avons pu voir un éventail de techniques qui prennent pour base la physique réelle et d'autres qui cherchent simplement à reproduire l'effet visuel de manière purement déterministe. Nous introduisons ici une technique utilisant une partie physique et une autre partie purement déterministe. Nous nous intéressons essentiellement à l'apport plastique des formes et des mouvements du fluide dans l'optique d'un rendu expressif.

4.2.a Fluides sans simulation de la vélocité

L'idée principale est d'utiliser la technique de l'advection, qui permet de transporter de l'encre par exemple. Une texture de vélocité est utilisée et sert à indiquer la direction dans laquelle va bouger l'encre. Cette texture ne sera, par contre, pas calculée par une simulation « physique » d'un liquide transport comme c'est le cas habituellement. En remplacement, nous utilisons simplement une série de sinusoïdes basées sur le temps et sur la position dans l'espace-écran du pixel à calculer.

Figure V.23: Application d'un effet de fluide en espace-écran afin de briser les contours nets de la surface.

L'effet fonctionne très bien sur les premiers tests, avec un seul personnage et une texture simple. Bien sûr, le mouvement du fluide ne suit pas une logique physique, mais l'association d'un mouvement cyclique, par le biais des sinusoïdes, et d'un transport itératif de l'encre va créer à la fois des vagues de fluides et de petites volutes très esthétiques. Cette technique est peu coûteuse et très contrôlable. L'advection de l'encre peut être faite simplement par un processus d'advection arrière, sans que l'obligation de compressibilité du fluide doive être respectée. L'encre est ajoutée en continu par l'objet 3D, est transportée par les oscillations paramétriques, puis disparaît au cours du temps. Le contrôle par l'artiste provient principalement du réglage des sinusoïdes. La fréquence des variations de vélocité peut être réglée à la fois spatialement et temporellement. Il est donc possible d'obtenir des déplacements d'encre rapides ou lents, qui changent de directions plus ou moins souvent, et un chaos plus ou moins prononcé dans l'image, simplement par le biais de ces réglages.

4.2.b Les liens entre l'espace-écran et l'espace monde

Nous souhaitons utiliser les fluides pour déformer l'image de la scène, et notamment les contours. Pour cela, il est nécessaire d'avoir des liens entre les éléments 3D (personnages, décors) de la scène et le fluide positionné sur l'écran.

Ainsi, l'encre est déposée sur la grille de fluide, soit par des objets spécifiquement dédiés, soit par l'image de la scène 3D elle-même. Nous avons choisi de déposer de l'encre en utilisant les zones très sombres (les traits noirs essentiellement) et les parties les plus éclairées de l'image.

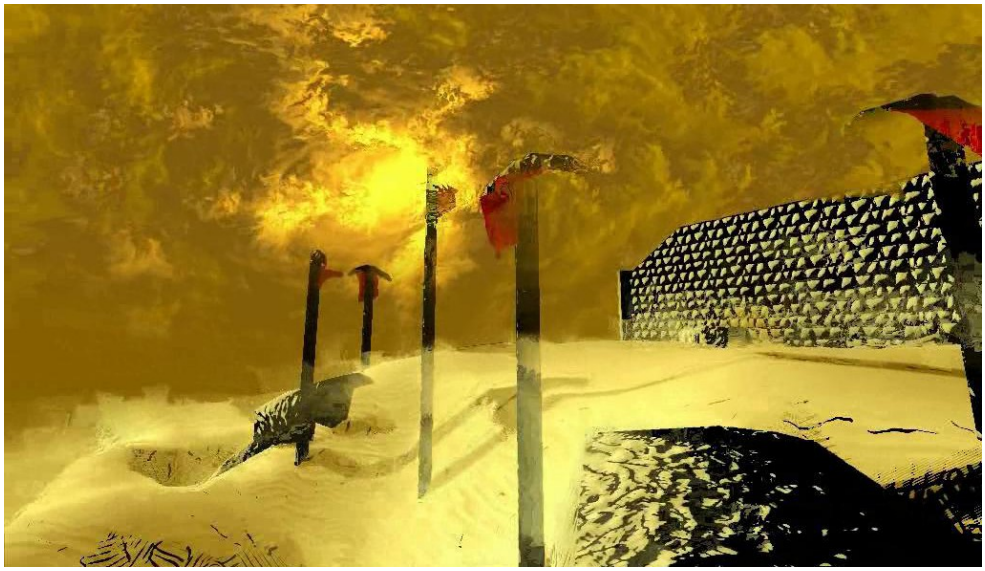


Figure V.24: Projet interactif regroupant les fluides en espaces-écran et les effets de rendu au trait.

Il est en effet important que l'encre soit déposée de manière ponctuelle, à travers de petites surfaces, et non par blocs importants. Ces blocs unis bougeraient d'un seul tenant, car l'encre ne pourrait se disperser autour. C'est essentiellement ce mécanisme de dispersion de l'encre par advection qui donne des effets plastiques intéressants.

En plus d'ajouter de l'encre, il est possible d'en influencer le déplacement en fonction de la scène 3D. L'idée est de disposer des objets 3D qui ne seront pas rendus dans la scène, mais dans une texture à part. Celle-ci indiquera au fluide 2D placé sur l'écran la direction et la vitesse avec laquelle le fluide doit se déplacer. Ces objets fixeront également la quantité d'encre déposée par le décor. Cela nous permet de créer des zones de la scène avec beaucoup d'encre, d'autres avec moins, et de jouer sur l'orientation générale et la vitesse du mouvement du fluide. Ces zones conserveront une cohérence spatiale lors des mouvements de la caméra, puisqu'elles sont placées dans la scène 3D sous forme d'objets invisibles. Ceux-ci vont ajouter de la vélocité au fluide en utilisant leur propre mouvement, ou bien des directions fixes. Ces

directions seront transférées dans l'espace-écran afin de donner une direction bidimensionnelle cohérente au fluide. Dans notre application, nous avons utilisé des maillages animés dont les sommets parcourent des sinusoïdes. La vélocité apportée va donc se présenter sous forme de vagues successives.

L'inconvénient majeur de cette technique, qui utilise un fluide en deux dimensions posé sur l'écran, se dévoile dès que la caméra se déplace. En effet, si la caméra tourne, l'encre qui a déjà été déposée sur l'écran reste figée, et persiste quelques instants avant de se dissoudre. Le fluide 2D ne prend en effet pas en compte les mouvements de la caméra.

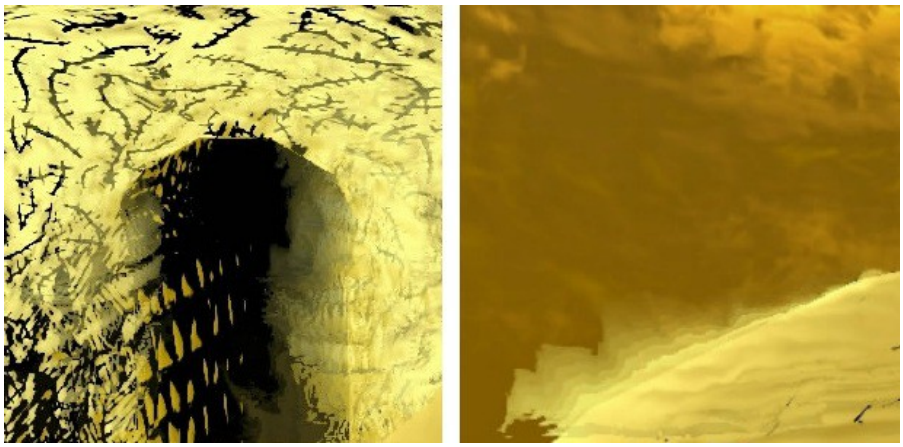


Figure V.25 : Détails illustrant l'apport des fluides sur l'image. Les couleurs peuvent déborder de leurs cadres et se propager alentour.

Une solution partielle est de simuler le fluide, non pas dans une texture 2D, mais dans une cube-map, c'est-à-dire un assemblage de 6 textures 2D qui forment un cube autour de la caméra. Lorsque nous tournons la caméra, le fluide reste fixe, car le fluide est simulé à 360 degrés autour de la caméra, grâce à la cube-map. Cela corrige le problème lors de la rotation de la caméra, mais pas lors d'un changement de position. Néanmoins, les traînées sont moins perceptibles lorsque le joueur avance ou recule. Par contre, pour avoir une bonne résolution pour le fluide, il faut utiliser une cube-map très lourde, par exemple de 1024x1024x6 pixels, ce qui impacte les performances négativement. Une solution intermédiaire est de simuler le fluide dans une texture légèrement plus grande que la taille de l'écran, et de compenser les rotations de la caméra en déplaçant le plan de simulation, afin que les pixels du fluide puissent suivre les mouvements de la scène vue depuis la caméra.

4.3 Le travail de la matière hachurée

Un des éléments très identifiables du style « dessiné » est l'utilisation de hachures et de traits pour représenter les ombres et les dégradés. Les hachures sont constituées d'un groupe de lignes, produites souvent par un coup de pinceau ou de crayon, qui forment un ensemble cohérent partageant des caractéristiques telles que la direction ou l'épaisseur. Les hachures sont des indices visuels illustrant l'éclairage, les propriétés d'un matériau, ou encore la forme de la surface. Ce type de visuel est directement hérité du dessin traditionnel et du croquis. On le trouve également beaucoup dans la bande dessinée.

4.3.a Les « Tonal Art Map »

Le principe fondamental de la technique présentée ici est d'utiliser plusieurs versions d'une même texture, chacune correspondant à un degré différent de luminosité de la surface. Ainsi, chaque version sera légèrement plus sombre que la précédente. Ce type de texture se nomme une Tonal Art Map. Le programme utilise ensuite pour chaque pixel, l'une ou l'autre version de la texture, selon la quantité de lumière reçue par le pixel. C'est un principe très simple, mais qui peut donner de nombreux effets différents selon le nombre et le type de texture utilisée. Le degré de luminosité de la texture est obtenu par l'application de hachures plus ou moins resserrées et non par l'utilisation de niveaux de gris.

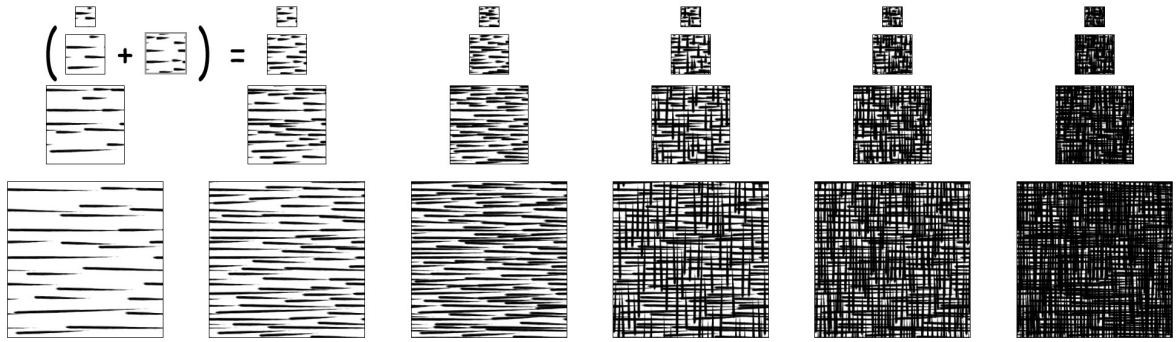


Figure V.26: La Tonal Art Map utilisée dans la publication originale de E. Praun et al.

Dans l'algorithme initial des Tonal Art Maps[115], les textures de hachures représentent simplement une matière, mais elles ne sont pas directement mises en relation avec la forme de la surface sur laquelle elles vont être appliquées. Ce sont des textures génériques qui ne sont pas adaptées aux formes d'un objet précis. Elles sont généralement produites par un processus automatique qui utilise des images de lignes dessinées manuellement pour créer toutes les

115 E. Praun et al., « Real-time hatching », in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 2001, 581.

textures, de sorte qu'elles soient répétables et qu'elles représentent un certain ton de gris perceptuel[116]. Pour affiner le rendu, certaines recherches se concentrent sur le calcul de la courbure de la surface afin de trouver un sens principal dans lequel appliquer les hachures. Cette direction principale donnée à la texture améliore la crédibilité du rendu et se rapproche plus de la manière dont un dessinateur organiserait ses hachures. Il s'agit également d'automatiser l'application du rendu hachuré afin de pouvoir l'appliquer sur n'importe quel objet sans nécessiter d'intervention manuelle. Les limites entre les zones formées par des courbures différentes nécessitent également un traitement particulier. Il peut consister par exemple en une interpolation entre deux textures de hachures, chacune alignée selon l'un des sens de courbure. Une des textures disparaît petit à petit et l'autre la remplace.

4.3.b Dessin manuel des textures

Nous avons supprimé cette recherche de la courbure en la remplaçant par un dessin entièrement manuel des textures de hachures, associées à leurs objets. Certaines surfaces n'ont pas besoin de traitement particulier, tel que les murs. Dans ces cas-là, une texture générique est créée, puis appliquée sur les différents murs par manipulation des coordonnées de texture, de la même manière que pour les textures classiques. Cette décision nous permet d'obtenir une corrélation beaucoup plus forte entre la surface et les hachures, en suivant manuellement la courbure de la surface et en accentuant volontairement les creux et les bosses. Par contre, le travail manuel est nécessaire et chaque objet nécessitera sa propre texture. La création des textures de hachures est grandement facilitée par l'utilisation d'un logiciel tel que Zbrush, qui permet de peindre les hachures directement sur le modèle 3D. Pouvoir observer l'objet tout en dessinant les hachures améliore la productivité et produit un résultat plus cohérent et crédible.

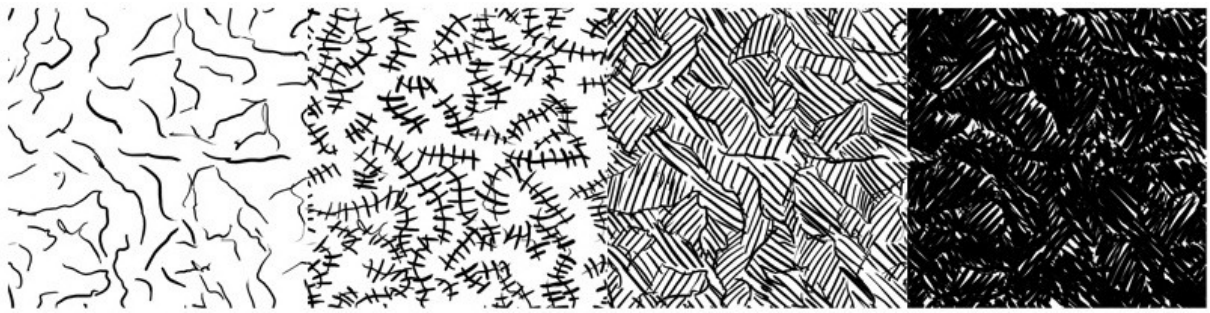


Figure V.27: Exemple de Tonal Art Map réalisée pour notre application. La luminosité moyenne diminue de gauche à droite.

116 Matthew Webb et al., « Fine tone control in hardware hatching », in *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, NPAR '02 (New York, NY, USA: ACM, 2002), 53–ff.

Au final, ce type de travail de création de textures en adéquation avec un objet et ses coordonnées de textures est une étape classique de la création d'un décor à laquelle les artistes 3D sont habitués, ce qui la rend aisément intégrable par une équipe de développement de jeux vidéos.

4.3.c Transitions entre les niveaux de luminosité

Nous avons utilisé 4 niveaux de textures de hachures, toutes en noir et blanc. La première version (la plus claire) ne dessine que quelques contours, un trait pour chaque creux dans la surface par exemple. Les versions ultérieures vont rajouter à chaque fois une couche de hachures, ce qui va assombrir au fur et à mesure la texture. Il est important de rajouter les hachures et non de recommencer à zéro avec d'autres hachures complètement différentes à chaque version. Ainsi, nous pouvons passer d'une texture à l'autre avec une transition douce et logique, car certaines hachures restent persistantes entre les niveaux de luminosité.

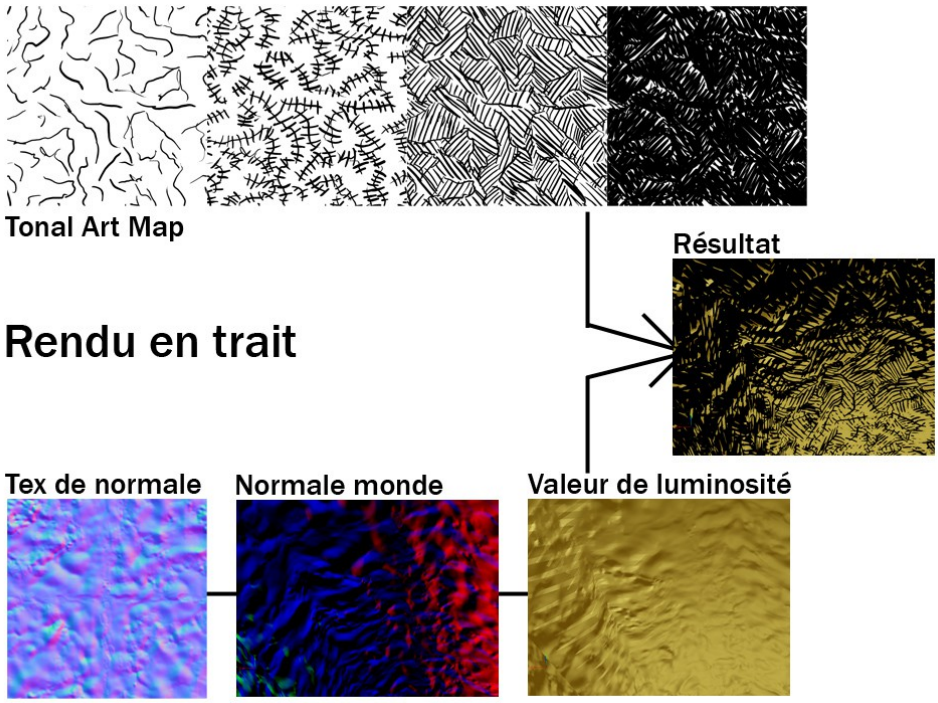


Figure V.28 ; Le principe du rendu au trait et la gestion de la lumière.

Il existe plusieurs manières de gérer les transitions entre les textures, ici nous sélectionnons simplement en fonction du degré de luminosité l'une des 4 textures ainsi que la texture suivante. Ces deux textures sont ensuite modulées linéairement selon le pourcentage de luminosité. Nous pouvons aussi utiliser soit l'une soit l'autre, sans transition, ce qui donne une séparation très visible entre les niveaux de luminosité.

Figure V.29: Application d'une Tonal Art Map sur un modèle de statue.
Modèle issu de RenderMonkey

Le rendu obtenu avec cette technique produit un résultat un peu trop flou, car de nombreux pixels vont prendre une teinte grisâtre au cours de leur passage du blanc au noir, entre deux textures. Afin d'obtenir un rendu bien délimité et contrasté, il nous suffit de multiplier fortement la valeur jusqu'à obtenir une saturation, et donc une séparation nette entre noir et blanc. C'est un processus d'augmentation du contraste. L'aspect flou disparaît au profit d'une grande netteté, mais également de hachures plus variables dans leurs épaisseurs et présentant des courbes plus organiques. Les traits sont plus ou moins fins selon la luminosité, et les croisements entre les hachures forment des courbes très douces.

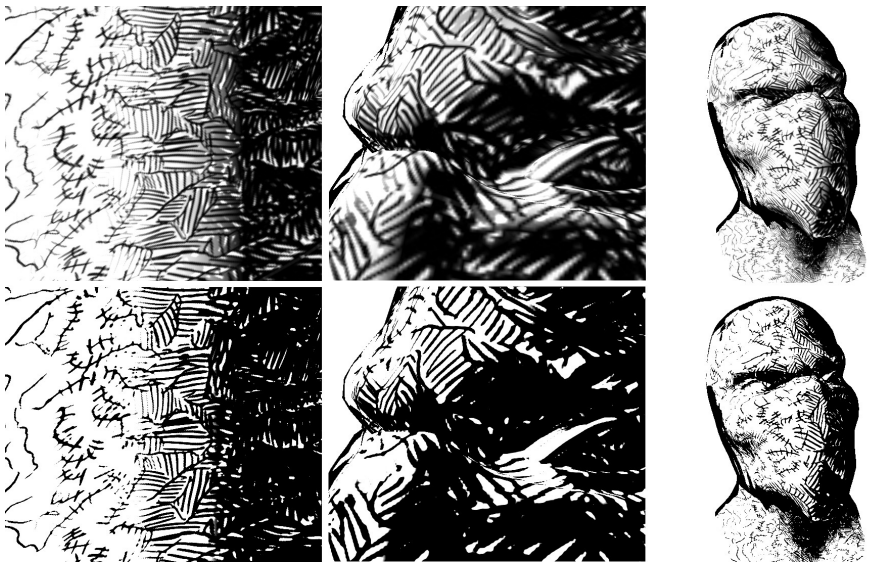


Figure V.30: Application d'une Tonal Art Map.
Application sur un dégradé de luminosité (colonne de gauche), sur un gros plan (colonne du centre) et sur un objet complet (colonne de droite). La ligne d'en haut montre le résultat brut du mélange des textures. La ligne d'en bas ajoute un contraste pour éviter l'aspect flou.
Modèle issu de RenderMonkey

Par contre, ce procédé de contraste ne fonctionne qu'entre les hachures à l'intérieur du même objet, et les courbes ne pourront pas déborder sur les autres objets. Pour obtenir ce débordement, nous pourrions ajouter une passe de flou sur l'image finale calculée, puis appliquer l'augmentation du contraste ensuite. Cette technique est par contre beaucoup plus lourde qu'un simple contraste par objet, qui peut se faire directement dans le pixel shader en ajoutant quelques instructions.

Les maillages utilisés en temps réel sont souvent assez faibles en polygones et peu détaillés. Dans ce cas-là, cette technique de rendu produira des lignes de séparations assez franches entre un niveau de luminosité et le suivant. Pour remédier à cela, nous utilisons une texture de normale (normal map), qui va servir à modifier la normale de la surface lors du calcul de luminosité du pixel. Cela nous permet aussi d'avoir beaucoup de détails sur un simple plan, par exemple pour figurer les pierres d'un mur. En adéquation avec l'utilisation des Tonal Art Maps, la texture de normale ajoute des variations qui permettent de diversifier l'aspect des surfaces. Étant donné que les hachures sont plus prononcées dans les creux, la texture de normale va encore accentuer les variations entre creux et bosses.

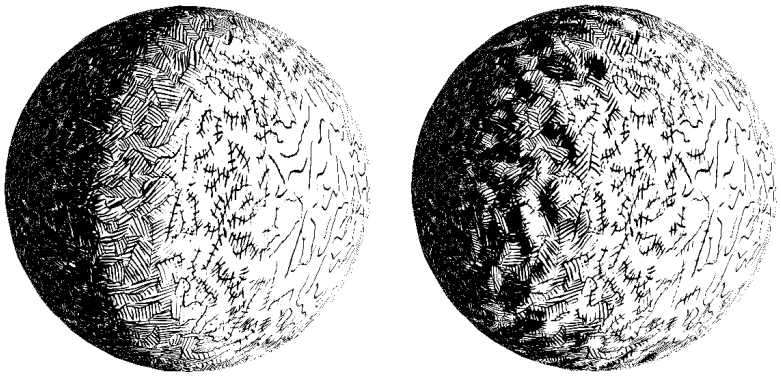


Figure V.31: Les problèmes de transitions entre les niveaux de luminosité des Tonal Art Maps. À gauche, application de l'algorithme sur une surface trop simple. Les transitions de l'ombre à la lumière semblent artificielles. À droite, utilisation d'une texture de normale pour calculer la luminosité de la surface. Les transitions sont beaucoup plus naturelles.

4.3.d Stockage des textures

Il existe plusieurs manières de stocker les textures multiples des « Tonal Art Maps ». La première manière est de poser les 4 images des niveaux de luminosité les unes à côté des autres dans une grande texture. Nous utilisons alors un décalage des coordonnées de texture pour échantillonner la valeur des deux textures à utiliser pour le pixel courant. L'échantillonnage du pixel dans la texture peut poser quelques problèmes. En effet, l'échantillonnage sur la carte graphique prend en compte les pixels adjacents afin de fournir

une valeur douce, interpolée bilinéairement, en utilisant le mécanisme de « mip-mapping ». Étant donné que nous passons d'un bout à l'autre de la texture en fonction du niveau de luminosité, l'échantillonnage est totalement faux, car il utilisera un niveau de mip-map avec une très petite résolution. Il faut alors renoncer au mip-mapping, même si avec les shader 3.0, il est possible de fournir explicitement les dérivés utilisées dans le calcul de l'échantillonnage. L'autre possibilité est de faire quatre appels à la texture au lieu de seulement deux, afin de ne pas avoir à sauter d'une partie à l'autre de la texture. Les quatre textures peuvent alors être présentées comme des textures séparées.

Figure V.32 : Utilisation d'une Tonal Art Map sur des personnages en mouvement.

Une technique plus simple est souvent plus rapide est d'utiliser des textures 3D. Chaque section de la texture contiendra une des versions de la texture 2D. Le calcul du mélange entre une texture et la suivante est alors gratuit, car il est effectué directement par la carte graphique.

Le résultat obtenu avec les Tonal Art Maps est très contrôlable et peut permettre d'obtenir un résultat esthétique proche d'une bande dessinée. Un problème subsiste néanmoins lorsque la caméra s'éloigne de l'objet texturé.

En effet, nous aurons le même phénomène qu'avec n'importe quelle texture, c'est-à-dire un clignotement très désagréable. En effet, pour chaque pixel affiché au final, plusieurs pixels de la texture pourront être choisis. Le choix étant plus ou moins aléatoire, le pixel clignote au moindre déplacement de la caméra. Pour palier cela, nous utilisons traditionnellement du

« mip-mapping », c'est a dire le stockage de versions réduites en taille de la texture qui sont utilisées en fonction de l'éloignement de la caméra. Si la technique fonctionne bien dans le cas de l'échantillonnage d'une texture 2D, l'application dans le cas de l'utilisation de textures 3D pose problème pour nous.

Le « mip-mapping » est en effet appliqué sur les trois coordonnées de la texture 3D. Si nous avons 4 subdivisions sur l'axe des Z, afin de stocker nos 4 niveaux de luminosité, nous n'en aurons plus que 2 au prochain niveau de « mip-map », puis un seul. L'utilisation du « mip-mapping » limite ainsi le nombre de niveaux de luminosité disponibles, ne laissant qu'un seul niveau lorsque la caméra est trop éloignée de la surface. La création des versions réduites des textures de hachures est également un challenge. En effet, les textures sont composées de traits individuels. Lorsque la taille de la texture diminue, les traits deviennent trop petits par rapport à la résolution, et se mélangent entre eux, deviennent flous. Au final, la texture tend à devenir un gris uniforme.

Figure V.33: Simplification du calcul en fonction de la distance. Au premier plan, les Tonal Art Maps sont utilisées alors qu'au fond c'est un simple seuillage de la luminosité qui assombrit les creux de la surface.

La solution acceptable ici est de décider nous même des textures à utiliser pour la création des versions en taille réduite. Nous n'utilisons que les versions les plus claires de la texture, qui sont composées de moins de traits que les plus sombres. Lorsque la caméra s'éloigne, la texture devient ainsi de plus en plus simple, avec moins de traits, jusqu'à disparaître. Pour

augmenter la qualité, il est possible de créer à la main des versions moins denses, avec des traits moins nombreux, mais plus épais, adaptés à des textures de petites résolutions. Afin de conserver une notion d'éclairage même à grande distance, alors que la texture de hachures utilisée par le « mip-mapping » est totalement blanche, nous noircissons artificiellement les parties ombrées de l'objet à partir d'une certaine distance, sans plus utiliser la texture. À cette distance, les détails stockés dans la texture de normale suffisent à générer un niveau de complexité visuelle proche des hachures utilisées sur les parties plus proches.

Figure V.34: La gestion de la lumière et des ombres portées produit une image complexe.

4.3.e La gestion des ombres portées

La gestion des ombres portées des objets les uns sur les autres est également un aspect important dans notre style graphique. Les ombres portées servent en effet à rendre la scène 3D plus cohérente et compréhensible, notamment en rendant perceptibles les surfaces de contacts entre un objet et le sol. Les personnages semblent par exemple flotter s'ils n'ont pas une ombre sous leurs pieds qui les rattachent au sol et nous indique leur hauteur. Nous avons privilégié la technique classique des textures d'ombres (Shadow maps) qui consiste à effectuer un rendu de la profondeur vue depuis une caméra située à la position de la lumière. Cette texture nous permet alors de savoir si un point donné est devant ou derrière le décor tel que vu par la lumière, et donc s'il est ou non dans l'ombre. Cette technique classique a été décrite dans de

nombreux ouvrages et notamment dans notre travail de Master[117], en parallèle avec la gestion de l'éclairage en 3D par la technique du « deferred lighting ».

La principale contrainte de cette technique d'ombrage provient du crénelage lié à la résolution de la texture d'ombre. Si celle-ci est trop basse, des carrés apparaissent et apportent un aspect très pixelisé au rendu. Dans notre cas particulier, le style graphique envisagé nous permet d'utiliser une solution très simple qui va masquer ces artefacts. Il s'agit d'un exemple de symbiose entre la technique et le style graphique, les défauts de la première devenant des qualités du second. Concrètement, nous appliquons un léger décalage sur l'endroit ou nous allons faire le test de profondeur dans la texture d'ombre. Ce décalage aura une forme sinusoïdale, ce qui fonctionne parfaitement avec un rendu hachuré tel que le nôtre. La forme sinusoïdale évoque des hachures, et masque agréablement les contours durs aux bords carrés caractéristiques des textures d'ombre. Cette technique n'est pas possible dans le cas d'un rendu classique de type « photoréaliste », même si un dérivé est parfois utilisé en remplaçant la sinusoïde par un bruit. Il s'agit de « l'échantillonnage fluctuant » (Jitter sampling en anglais) qui remplace le crénelage par du bruit, souvent différent à chaque image.

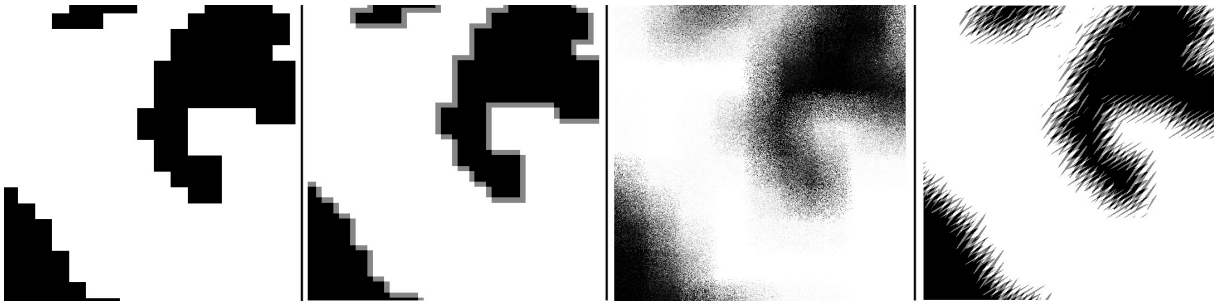


Figure V.35: Adaptation de la technique d'ombrage au rendu hachuré.
De gauche à droite : une ombre binaire, la superposition de deux ombres binaires, l'ajout d'un « jitter » de bruit sur les coordonnées de textures des deux ombres binaires et l'utilisation de sinusoïdes à la place du bruit pour respecter le style hachuré.

Pour obtenir des hachures dans les deux sens, nous pouvons faire deux appels à la texture, l'un utilisant une sinusoïde sur X et l'autre sur Y, les deux axes de la texture d'ombre. Le degré d'ombre sera alors la moyenne de ces deux résultats. Tout d'abord, ce décalage était calculé dans le repère de la caméra, mais lorsque celle-ci bougeait, les hachures changeaient continuellement de forme, ce qui était perturbant. La solution est d'utiliser la seule valeur qui changera pour chaque pixel calculé tout en restant cohérente avec le déplacement de la caméra, c'est-à-dire la position du pixel calculé dans le repère global. Cette position est bien

117 Antoine Zanuttini, « Recherches en rendu non réaliste temps réel » (2008).

évidemment exprimée en trois dimensions, avec trois coordonnées, or nous avons besoin d'un décalage en seulement deux dimensions, les deux axes de la texture d'ombre.

Une équation simple avec le calcul suivant nous permet de réduire les trois dimensions à deux et donne déjà de bons résultats :

*décalage.xy = sin(position.xy * position.z)*

Figure V.36: Les contours des ombres portées sont hachurés de manière procédurale afin de masquer la faible résolution des « shadow maps ».

4.3.f Ajout des couleurs de fond

Jusqu'à présent, nous nous sommes intéressés aux hachures noires uniquement. Il est bien sûr possible de faire varier les couleurs des hachures, ainsi que leur degré d'opacité. Néanmoins, les tests effectués ne nous ont pas convaincus et les traits sont donc restés d'un noir pur. Les espaces de vides laissés entre les traits noirs deviennent par conséquent les seuls indices de couleurs des matières et des lumières. Les contours et les hachures servent à mettre en valeur et à contraster ces couleurs.

Au départ, nous souhaitions un rendu en aplats de couleurs, chaque objet se limitant à deux ou trois couleurs sans dégradé entre celles-ci. Une couleur principale bien sûr, puis la couleur en présence de lumière et une couleur dans l'ombre, entre les hachures. Cependant, le rendu paraissait un peu pauvre, et le mélange des couleurs des différentes lumières avec la couleur du matériau pour produire une seule couleur est assez délicat à contrôler. Nous avons donc

finalement décidé d'utiliser un éclairage tout en dégradés subtils. Les couleurs restent très saturées, mais se teintent des couleurs des lumières environnantes tout en ne tombant jamais dans le noir absolu, réservé aux contours et hachures, ou dans le blanc pur. Les couleurs claires tirent vers le jaune plus que vers le blanc, afin de souligner l'atmosphère très chaude.

Le contraste augmente entre les zones éclairées et les zones sombres, puisque les premières ont peu de hachures et un fond de couleur clair alors que les secondes ont des hachures en grand nombre et très resserrées, en plus d'un fond sombre. Le contraste local au sein d'une même zone en est diminué, apportant une forte cohérence et rendant l'image plus compréhensible. La variation de la densité des hachures n'est pas toujours suffisante pour spécifier un degré de luminosité donné. L'alternance de blanc et de noir très franc des hachures est peu lisible, alors que l'alternance d'une couleur brune et des traits noirs permet d'atteindre des tons apparents plus sombres et moins perturbés par le contraste des hachures.

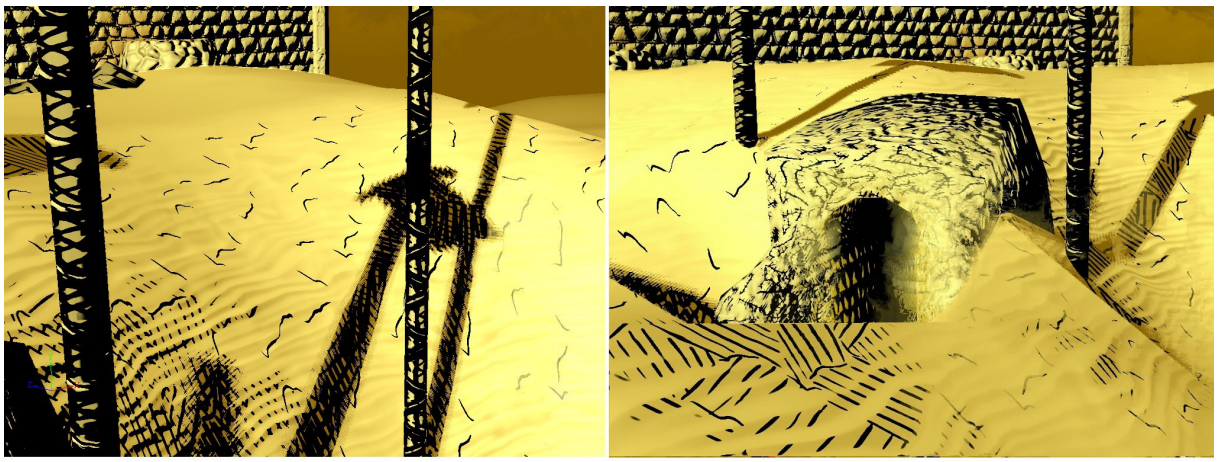


Figure V.37: Détails de l'application interactive que nous avons mis en place pour nos tests de rendu graphique.

Le sujet de la réflexion, qui a été abordé longuement dans le chapitre III, n'a pas pu être associé au rendu expressif dans cette étude, mais constitue un sujet qui n'a pas encore été très étudié. La prise en compte des réflexions diffuses ambiantes (c'est-à-dire la radiosité) peut largement contribuer au rendu expressif, comme c'est le cas pour certains jeux (Team Fortress 2 par exemple). La réflexion spéculaire ambiante, et notamment l'utilisation de la rugosité des matériaux pour faire varier le degré de flou de la réflexion, pourrait également permettre l'apparition de nouveaux styles de rendu expressifs.

La génération des « Tonal Art Maps » et des textures de normale associées a été faite grâce à Photophop et au plug-in « nVidia normal map », ainsi que part une utilisation ponctuelle de zBrush pour la peinture des hachures directement sur les objets 3D importants tels que les

personnages. Les textures utilisées sur les murs ont été créées directement en 2D et ont la particularité d'être répétables, c'est-à-dire qu'il est possible de placer côte à côte plusieurs fois la texture sans percevoir les frontières. Cela nous permet de créer une seule texture, pour un mur de briques ou un sol rocheux par exemple, puis d'appliquer la texture sur toute une portion de décors, sans devoir peindre chaque mur individuellement. La scène temps réel présentée dans cette partie se contente ainsi d'une dizaine de textures différentes (chacune déclinée dans les différents niveaux de luminosité). La diversité dans l'image proviendra principalement de l'utilisation de la lumière pour sculpter différentes zones sans pour autant devoir créer d'autres textures. La lumière est en effet un très bon outil pour donner du caractère à des décors. C'est un outil très rapide et efficace du point de la production, car le placement des lumières peut s'effectuer très vite.

L'insertion dans Virtools et la gestion personnalisée des « mip-maps » a été faite par l'intermédiaire du format .dds grâce à DxTex, disponible dans le SDK directX.

4.4 Techniques de représentation graphique des contours

4.4.a Détection des contours et application d'une texture

La nouvelle technique mise au point ici va chercher à mélanger subtilement des données calculées en 2D et en 3D. La première étape est d'effectuer une simple détection de contours, que nous enregistrons dans une texture. Nous allons également enregistrer dans cette texture une information de progression le long du contour (nommée ici « progression »). Cette valeur va représenter la première dimension lorsque nous ferons appel à une texture de contour. Ensuite, plusieurs passes successives vont se charger de faire une expansion de ce contour, c'est-à-dire que chaque passe va simplement modifier les pixels adjacents à un contour. Ils vont prendre la valeur du contour le plus proche. Pour que nous puissions distinguer les pixels selon leur distance par rapport au contour initial, chaque passe va faire diminuer une seconde

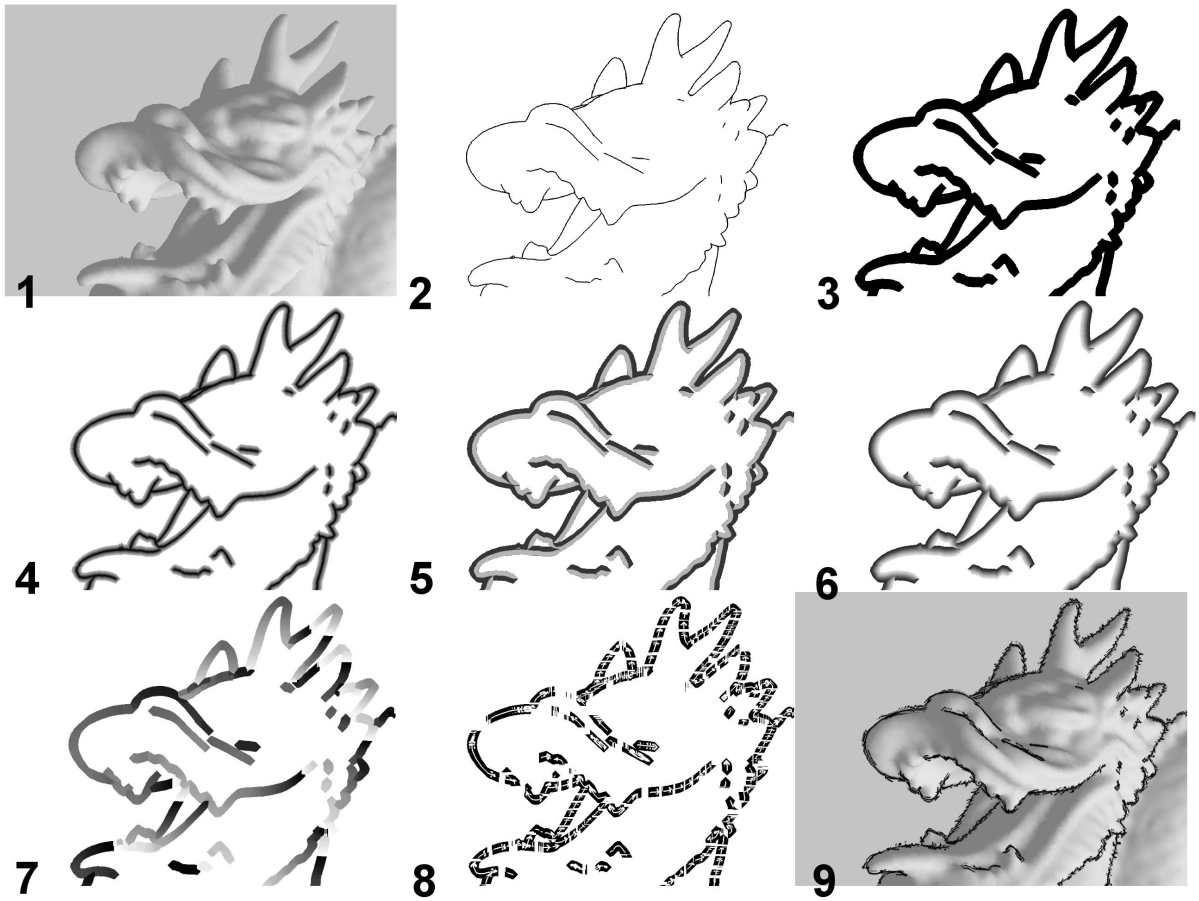


Figure V.38: Les étapes de calcul de l'effet de contours.
1 : Rendu de départ, 2 : Détection des contours en fonction de la profondeur, 3 : Épaississement des contours, 4 : Dégradé en fonction de la distance au contour, 5 : Séparation des deux cotés du contour, 6 : Dégradé de l'extérieur vers l'intérieur du contour, 7 : Valeur de progression le long de la texture, 8 : Texture d'exemple appliquée sur le contour, 9 : Image finale avec rendu et contours texturés.

valeur aux pixels de contours quelle ajoute. Grâce à cette valeur (nommée ici « distance au contour »), il sera déjà possible de dessiner très simplement des contours plus ou moins épais..

Pour pouvoir faire appel à une texture complète en deux dimensions, nous allons utiliser la distance au contour, et rajouter une information qui dépend du sens par rapport au contour. Comme le contour est la frontière entre deux zones, l'une plus proche et l'autre plus éloignée, les pixels de contours situés du côté le plus proche vont être marqués comme positifs et les plus éloignés comme négatifs. Grâce à cela, il est possible de faire la différence entre les deux faces du contour, et donc d'afficher une texture qui ne soit pas symétrique. Nous utiliserons les deux dimensions constitués par la progression le long du contour (en X) et par la distance signée au contour (en Y) pour dessiner une texture qui se répétera sur les bords de l'objet 3D.

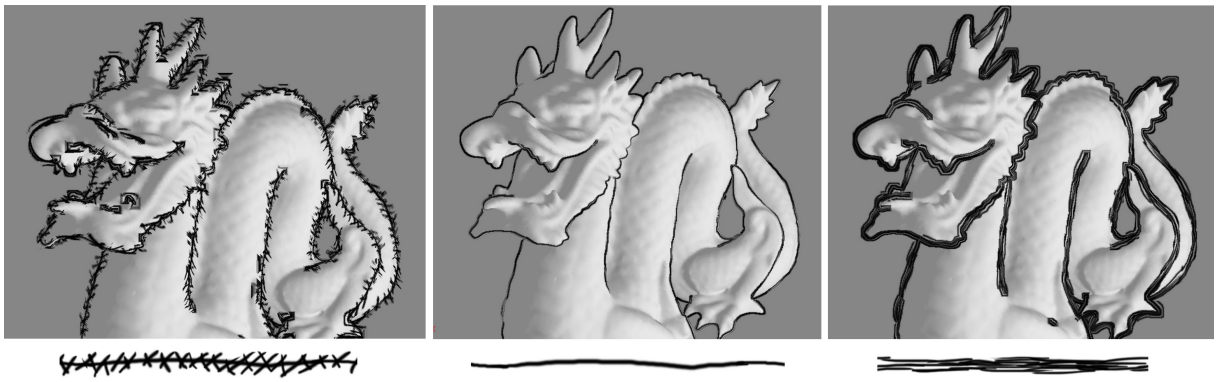


Figure V.39: Exemples de contours différents appliqués sur le même objet.
La texture utilisée en tant que contour est affichée en dessous de chaque rendu.
Modèle issu du Stanford 3D Scanning Repository

Pour que l'information de progression du contour soit cohérente avec la scène 3D, et donc consistante lors du changement d'angle de caméra, nous calculons cette valeur à partir de la position dans l'espace monde du pixel de contour. Évidemment, un contour n'a pas en soi de position, c'est la barrière entre deux zones, il va donc falloir prendre l'une de ces deux zones en référence, ici c'est la plus proche de la caméra qui sera utilisée. À partir des trois coordonnées (X, Y et Z) du contour, il va donc falloir sortir une seule valeur qui puisse boucler, puisqu'elle va servir à indexer dans une texture. La caractéristique principale que cette valeur doit avoir est de toujours évoluer quel que soit le point de vue et de rester très continue pour ne pas avoir de cassures dans la texture. En respectant ces deux contraintes, la texture se répétera régulièrement et sans brisures. La formule idéale n'existe probablement pas, mais en pratique, des formules très simples donnent des résultats déjà intéressants. Ainsi, la simple addition des trois composantes (XYZ) suffira dans notre cas. L'utilisation de

fonctions sinusoïdales peut également aider à ajouter de la diversité et un caractère non linéaire dans le transfert de la valeur de position depuis l'espace 3D vers la valeur de progression, qui s'exprime dans un espace 1D, le long de la texture de contour.

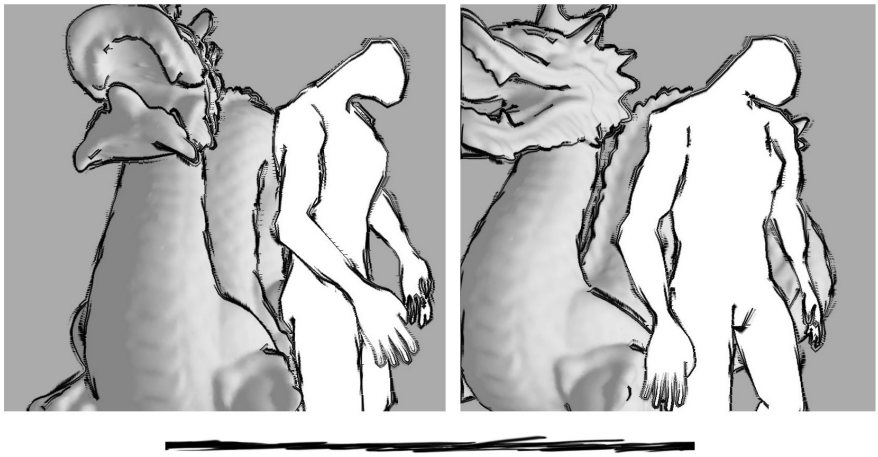


Figure V.40: Exemple d'application d'un contour texturé sur plusieurs objets à la fois.

De nombreux points restent encore à travailler, notamment les soucis de crénelages et de pixellisations qui interviennent fréquemment et rendent les contours très secs et brouillons. Le phénomène peut être masqué en partie avec une texture de contour suffisamment bruitée. Pour rendre les contours plus clairs, il faudrait également appliquer des filtres sur les contours pour éliminer les bords trop petits et rendre les contours plus lisses. L'application d'un filtre gaussien en espace-écran sur la valeur de progression permet par exemple de supprimer une bonne partie du crénelage.

4.4.b Application d'une texture de contours en fonction de la normale

Nous avons vu qu'il est possible d'utiliser une détection de contours, puis d'épaissir le trait afin de retrouver des coordonnées de textures à partir des étapes d'épaississement. Nous avons également créé une technique plus simple pour obtenir un résultat proche, mais sans la lourdeur de l'étape de détection de contour et d'épaississement.

En effet, le facteur de Fresnel d'un objet 3D peut être utilisé comme gradient, en remplacement du processus d'épaississement. Le facteur de Fresnel, dans le domaine du rendu temps réel, est un calcul qui indique si une normale donnée est en direction ou non de la caméra.

Voici la formule couramment utilisée :

$$Fresnel = |\vec{N} \cdot \vec{C}|^p$$

N : vecteur normal à la surface
C : direction de la lumière

Sur les objets aux formes organiques, tels qu'un visage ou bien une sphère, les normales sont douces et n'offrent pas de brusques changements de direction. Cela nous permet d'obtenir toujours, près des bords de l'objet, un dégradé du facteur de Fresnel, passant de 1 à l'intérieur de l'objet jusqu'à 0 sur les bords extrêmes. Les objets anguleux ne fonctionnent pas aussi bien, car les discontinuités dans la direction de la normale peuvent laisser des bords importants de l'objet sans variation du facteur de Fresnel.

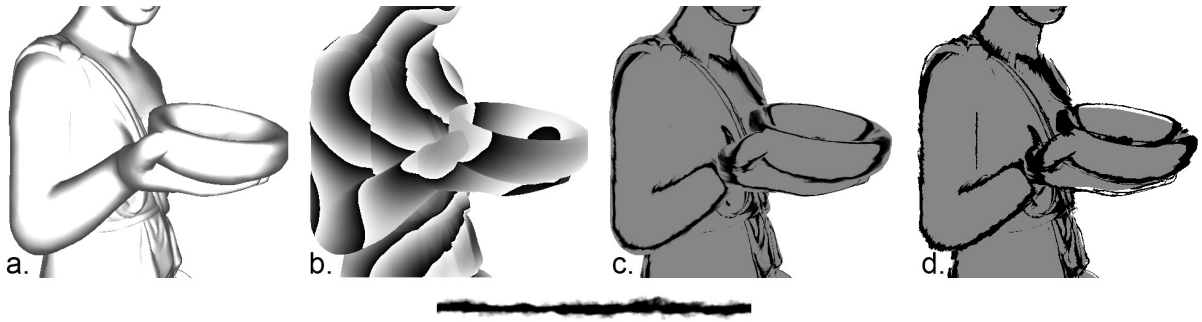


Figure V.41: Utilisation du facteur de Fresnel pour calculer les coordonnées de textures des contours.
a. Première coordonnée : facteur de Fresnel, limité aux bords de la surface
b. Seconde coordonnée : valeur 1D se modifiant doucement en fonction de la position 3D
c. Texture de contour appliquée sur les bords
d. Suppression des pixels au delà du contour et épaississement du maillage
En dessous : texture de contour utilisée
Modèle issu de RenderMonkey

Le facteur de Fresnel, une fois retaillé afin de limiter l'effet de contours aux seuls bords de la surface, offre un dégradé qui peut servir de première coordonnée pour indexer notre texture de contour. L'obtention de la seconde coordonnée peut se faire de la même façon que pour la technique précédente, c'est-à-dire en utilisant des fonctions mathématiques pour traduire la position 3D d'un pixel en une seule variable évoluant doucement.

Au niveau des calculs, cet algorithme est très simple et peu coûteux et permet effectivement d'appliquer une texture de traits répétable sur les contours de l'objet. Par contre, l'effet ne s'échappe pas des contours rigides des polygones, et se termine toujours abruptement sur les bords de l'objet. Afin d'améliorer cet aspect de l'algorithme, il est possible de décider, dans le

pixel shader, de détruire certains pixels du contour, qui ne sont alors pas rendus. Le principe est le même que celui du masque alpha, c'est-à-dire l'utilisation dune valeur de la texture pour rendre totalement transparent certains pixels. Ici, la texture utilisée est celle appliquée sur les bords, et seule la partie extérieure du contour est effacée. Cela nous permet d'obtenir effectivement des bords extérieurs découpés plus finement que le niveau du polygone. Par contre, les bords intérieurs des objets complexes peuvent poser problème, car les concavités dans la surface seront interprétées comme des contours, et donc seront découpées, laissant un vide dans l'objet. Pour remplir ce vide, nous pouvons dessiner une première fois l'objet normalement, puis dessiner une seconde fois l'objet, avec la technique de contour, mais en épaississant le maillage, selon la technique d’expansion des sommets en direction de la normale que nous avons présentée précédemment. Le premier rendu s'occupe de la couleur de l'objet en lui même et le second ajoute uniquement les contours, qui peuvent alors laisser des trous sans souci. Nous retrouvons les inconvénients de l'utilisation du facteur de Fresnel, qui ne produit pas des contours d'une épaisseur fixe, mais qui varie en fonction de la courbure de la surface. La taille de l’expansion des sommets ne conviendra pas non plus à tous les contours de l'objet, laissant apparaître quelques trous entre la surface et le contour.

5 Conclusion

Dans cette partie, nous avons étudié le sujet du rendu expressif, qui s'inspire des œuvres de la peinture et du dessin afin de proposer un degré d'expression plastique plus poussé. Nous avons pu voir que le rendu expressif cherche également à donner plus de contrôle à l'artiste en lui proposant des outils qu'il peut manipuler à l'aide de techniques classiques du dessin. Par ce biais-là notamment, le rendu expressif rejoint la recherche en rendu dit « photoréaliste ». Ce dernier a également besoin d'un contrôle avancé sur la matière première de la 3D, les polygones, mais aussi sur les autres effets qui permettent de s'échapper des contraintes du rendu temps réel classique. Les deux domaines sont en train de converger et de s'échanger des techniques et des outils. De nombreux jeux vidéo s'inspirent du rendu expressif pour styliser leur rendu et éviter la comparaison avec la réalité photographique.

Après avoir présenté les algorithmes « classiques » du rendu expressif ainsi que plusieurs projets du domaine qui ont été plus loin, nous sommes passés à l'intégration des techniques présentés dans les trois chapitres précédents, par différentes expériences.

La profondeur de champ a été ainsi utilisée pour le contrôle d'un effet de « rêve ». L'image se déforme, se floute et voit ses couleurs changer en fonction de la distance à la caméra.

Des fluides se sont ensuite vus appliqués sur l'écran afin de transporter de l'encre virtuelle déposée par l'image, au grès du mouvement apporté par certains objets de l'espace 3D. L'association des couleurs de la scène 3D et de la simulation de fluide en 2D sur l'écran produit un effet particulier et très adapté au rendu expressif.

Le travail de la matière virtuelle a été étudié dans le cadre du rendu hachuré, avec la prise en compte de la lumière pour moduler l'aspect des traits et la couleur de la surface. Nous avons présenté un algorithme très simple et adapté au contrôle du rendu par les graphistes.

Enfin, les effets de contours ont été étudiés. Nous avons introduit à cette occasion nos propres techniques permettant d'appliquer une image de contour sur les bords d'un objet 3D. Grâce à ces techniques, les contours peuvent être plus complexes et expressifs, ils peuvent sortir des limites des polygones de la surface et participer à l'expressivité de l'image.

Grâce à ce chapitre, nous avons pu voir que le rendu calculé par l'ordinateur consiste essentiellement à représenter une scène virtuelle en utilisant certains de ses attributs pour les transformer en caractéristiques de l'image, sans que le lien soit forcément simple ou

mimétique. Le rendu ne se limite donc pas à la représentation fidèle d'une scène virtuelle. Les objectifs de transmission d'informations lisibles, d'expression de sentiments ou d'idées abstraites font également partie des possibilités du rendu expressif, mais également des possibilités de l'image 3D en général, y compris lors de l'utilisation d'un « style » photoréaliste.

Conclusion générale

Au cours de notre étude, nous sommes passés par de nombreux constats et avons abordé de multiples solutions, en apportant nos propres algorithmes pour chacun des sujets présentés. Il est temps maintenant d'établir un bilan de notre exploration.

Rappelons que notre objectif était double. Il s'agissait tout d'abord d'approfondir chacun des thèmes présentés, d'effectuer un bilan théorique et technique puis d'apporter une réponse personnalisée, adaptée et innovante. Chacun des thèmes est en soi un défi qui n'a pas encore reçu de solution standardisée dans le domaine du temps réel, mais fait l'objet de recherches intensives. Ils laissent donc une large part à l'innovation et à l'intuition dans la démarche de création.

Notre second objectif était d'analyser l'impact de chacun des thèmes présenté sur la représentation par l'image. La question du « style » graphique a été discutée, et particulièrement la distinction entre « photoréalisme » et rendu « expressif », distinction dont nous avons montré l'absence de véritable sens. Les deux « styles » s'influencent l'un l'autre, s'empruntent des caractéristiques et finalement travaillent dans le même but, la création d'une image intéressante pour le joueur. Le rendu expressif se base souvent sur les caractéristiques de la réalité et les retranscrit différemment à l'écran. Le rendu dit « photoréaliste » cherche surtout à être crédible plutôt que fidèle, et surtout à éviter la fameuse « uncanny valley ». L'« uncanny valley » est cette zone fatidique de la courbe de « crédibilité » dans laquelle tombent les images qui se veulent de plus en plus proches de la réalité cinématographique, mais qui ne l'atteignent pas tout à fait. Les images produites sont alors désagréables, car à la fois familières et artificielles. La solution adoptée généralement passe par la stylisation de l'image, la caricature discrète. Cette solution cherche également à donner du contraste à l'image, des éléments intéressants et travaillés. L'image devient plus subjective qu'une capture neutre de la réalité.

Après avoir présenté le domaine du graphisme dans le jeu vidéo, ainsi que le contexte de recherche, nous nous sommes intéressés à trois sujets qui sont habituellement rattachés au photoréalisme.

La profondeur de champ

Nous nous sommes intéressés au flou de profondeur de champ, un héritage principalement lié au cinéma et à ses contraintes techniques, mais qui est devenu un véritable outil pour l'expressivité artistique et narrative. La profondeur de champ joue un rôle important dans la perception de la spatialité de la scène, mais est également un moyen de structurer et d'améliorer la lisibilité d'une image. C'est aussi une modification profonde de l'image qui porte sa propre plasticité, à la fois abstraite et réaliste. Nous avons également constaté que le flou de profondeur de champ pouvait être un frein à l'immersion du joueur en limitant son contrôle sur le champ de vision.

Une des caractéristiques du flou optique liée aux caméras réelles est la forme de la lentille, qui apparaît notamment sur les points très lumineux de l'image. Cette forme, ou « bokeh », fait l'objet de nombreuses études afin d'être intégrée dans les applications en temps réel. Nous avons également montré la connexion entre le flou de profondeur et la réduction du crénelage dans l'image. Un flou de bonne qualité avec une prise en compte de la forme de la lentille nécessite en effet une image avec peu de crénelages spatial et temporel. La mise en commun du flou de profondeur, du flou de mouvement et de l'anticrénelage est une piste de recherche intéressante pour l'amélioration de la qualité et des performances de ces trois effets.

Nous avons apporté notre contribution par le biais d'une technique innovante de flou de forme hexagonale. Cette technique est essentiellement indépendante de la taille du flou désirée et permet d'utiliser un flou plus correct par dispersion des couleurs à la place du rassemblement habituel.

La profondeur de champ est un outil en constante amélioration qui est pour le moment lié à la reproduction des caméras réelles. Nous avons vu qu'elle peut jouer un rôle important dans la construction d'une image intéressante et convaincante.

Matières, lumières et réflexions

La reproduction des matières et des effets de surfaces passe essentiellement, nous l'avons vu, par la question de la lumière et de sa réflexion. C'est en effet la lumière qui révèle la plupart des caractéristiques des surfaces (couleur, rugosité, humidité…). Avoir la possibilité, pour un artiste du numérique, de reproduire les matières réelles lui permet de créer des images convaincantes, mais aussi de travailler ses matières en fonction de caractéristiques intuitives. Le retour visuel instantané du temps réel est mis à profit par exemple pour « peindre » la rugosité d'une surface directement sur l'objet 3D. La lisibilité et la compréhension de l'image peuvent également être améliorées par l'utilisation de matières photoréalistes, de même que la spatialité de la scène et la reconnaissance des matières. Nous avons présenté et commenté les choix effectués au niveau de la description de la surface, dans sa couleur, rugosité, normale...

La réflexion diffuse peut être stockée puis réutilisée lors des mouvements de caméra. La question du calcul dynamique des rebonds lumineux diffus, appelée habituellement radiosité, fait toujours l'objet de recherches intensives. La réflexion spéculaire ambiante, c'est-à-dire la réflexion de l'environnement en fonction de la matière et de la position de la caméra, ne dispose pas encore d'algorithmes standardisés permettant de la stocker puis de la retranscrire rapidement. Nous nous sommes donc intéressés au sujet de la reproduction de la réflexion spéculaire ambiante, et surtout à l'une de ses spécificités, la parallaxe.

Nous avons proposé deux nouveaux algorithmes. Le premier va effectuer un stockage particulier du décor afin de privilégier les réflexions rasantes et l'influence de la parallaxe. L'interpolation entre les échantillons stockés nous permet d'obtenir une réflexion floue précalculée à faible coût. L'algorithme se limite à la réflexion planaire.

Notre second et principal algorithme adopte une approche par « proxies », c'est-à-dire des versions de la scène très simplifiées. Nous utilisons une texture d'environnement (sous la forme d'une cube-map) projetée sur un volume, ainsi que des plans texturés pour reproduire plus simplement les décors. Le mélange des différents proxies en fonction de l'emplacement de la caméra nous permet d'adapter la réflexion à l'ambiance locale. Nous avons apporté plusieurs innovations, avec notamment une méthode permettant d'afficher nos proxies dans une résolution faible, mais avec peu de crénelage visible. Nous avons également mis en place un calcul en temps réel de la pyramide de mip-map de la texture de réflexion afin de gérer le flou de la réflexion en fonction de la rugosité de la surface. L'ensemble de la technique est très rapide afin de ne pas impacter les performances du jeu.

Notre prochain but concernant ce sujet sera la gestion de la réflexion sur plusieurs plans. Nous chercherons également à améliorer la qualité du flou de la réflexion.

Les fluides

Le sujet des fluides diffère des deux thèmes précédents, car il amène directement du mouvement dans l'image. Dans le cas de la profondeur de champ, de la matière et de la réflexion, l'image se modifiait principalement par un déplacement de la caméra. Le rendu d'une image isolée devient moins important dans le cas des fluides, au profit des caractéristiques temporelles et de l'impression de mouvement qui est ressentie sur plusieurs images successives.

Nous avons mis l'accent sur les simulations de fluides basées sur la physique, qui permettent une grande complexité dans le mouvement et sont très adaptées aux mondes virtuels. Les simulations vont en effet pouvoir réagir aux interactions entre le jeu, les personnages et les décors.

Nous avons remarqué que les étendues d'eau parcourues par des courants présentent des caractéristiques à plusieurs échelles. Le mouvement global de l'eau peut être simulé à une échelle très large, alors que les remous de l'eau nécessitent une plus grande fréquence. Nous avons donc travaillé principalement sur le sujet de l'advection de textures sur une surface d'eau globalement plane. Le but est d'ajouter des détails à petites échelles par le biais de textures. La simulation à grande échelle sera utilisée pour mettre en mouvement ces textures. Ce type de technique permet d'avoir une impression de mouvement et de détail avec une simulation très rapide, à petite résolution.

Au cours de nos recherches, nous avons mis en avant l'idée de la séparation du fluide en zones d'advection continue, c'est-à-dire où la texture se déplace en un seul bloc. L'utilisation d'une grille hexagonale s'est révélée le meilleur équilibre entre la régularité des zones et la crédibilité du mouvement. Grâce à cette technique, nous obtenons un algorithme très rapide qui permet de simuler le fluide sur le processeur principal, puis d'ajouter du détail grâce à la carte graphique.

Notre technique est très efficace, mais montre tout de même ses limites si le fluide présente des mouvements très rapides qui ne pourront être perceptibles au travers de zones de petite taille. L'agrandissement des zones améliore la perception de la vitesse, mais réduit la

précision des mouvements du fluide, puisque chaque zone se déplacera à la même vitesse, formant de grandes plaques au lieu d'un flux visuellement continu. Néanmoins, nous avons trouvé un équilibre entre la taille des zones et la vitesse moyenne du fluide qui est efficace dans notre application. Les pistes de recherches futures s'orientent vers l'adaptation de la taille des zones en fonction de la vitesse locale du fluide, par exemple par une approche reposant sur des grilles à plusieurs échelles.

Nous avons présenté brièvement les méthodes utilisées pour afficher une surface d'eau crédible. Elles reposent notamment sur le mélange entre la couleur de l'eau, la texture de normale advectée, la couleur de la lumière, la réflexion et la réfraction de l'environnement. Nous avons mis en place une gestion de l'écume qui demande encore des recherches pour atteindre un rendu tout à fait crédible.

L'utilisation des fluides pour donner du mouvement à l'image est très efficace, et apporte une grande crédibilité au monde virtuel par ses réactions aux collisions du décor ou des personnages. La surface de l'eau nous présente un aspect toujours changeant qui mélange la lumière, la réflexion et la réfraction en fonction du courant. L'aspect physiquement réaliste de la simulation créé une image immédiatement identifiable aux fluides. L'image semble alors naturelle et compréhensible malgré une grande complexité visuelle.

Le rendu expressif

Enfin, nous nous sommes intéressés au rendu expressif, qui n'est pas en soi une technique, mais plutôt une vision de l'image numérique qui repose essentiellement sur la volonté de l'artiste de faire passer un message, une ambiance et d'obtenir une image plus personnelle. Le rendu expressif va peut être au-delà du rendu dit « non-photoréaliste » en étant capable d'inclure des effets complexes et qui reposent sur des règles issues de la réalité, tels que la profondeur de champ, la matière et les fluides.

Nous avons ainsi présenté des expériences d'intégration de chacun de ces effets dans le cadre du rendu expressif.

La profondeur de champ nous a permis de réaliser un effet évoquant le souvenir, en jouant sur le flou, les couleurs et la distorsion de l'image. Nous disposons des mêmes contrôles qu'une caméra, permettant de faire le point sur une distance et des objets particuliers. Les parties qui

sont hors de la zone de focus deviennent alors imprécises et se mélangent en fonction d'une sorte de fluide qui se déplace.

Le transport d'une encre virtuelle sur l'écran nous a permis d'illustrer visuellement l'apport des techniques de simulation de fluides au rendu expressif. Le fluide perturbe l'image initiale et brise la régularité du rendu classique de la 3D temps réel. Il apporte également de la vie et du mouvement à l'image. De plus, c'est une indication d'un élément de la scène virtuelle qui serait sinon invisible, le vent. Il parcourt les décors et emporte les couleurs de l'image avec lui en formant de courtes traînées.

En ce qui concerne la matière, nous nous sommes principalement intéressés à l'utilisation de hachures qui s'adaptent en fonction de la lumière qui atteint la surface. Les hachures peuvent être dessinées « à la main » puis appliquées facilement sur la surface. L'utilisation d'une texture de normale permet d'intégrer une notion de relief et de matière dans les surfaces hachurées.

Enfin, un des effets majeurs du rendu expressif a été introduit : le dessin des contours. Nous avons mis en place une technique inédite permettant d'appliquer une image dessinée à la main sur les contours d'un objet 3D en temps réel. La cohérence temporelle est assurée, bien que la texture de contour subisse des étirements une fois appliquée. Les contours ainsi réalisés vont bien sûr accentuer les caractéristiques principales de la surface, mais aussi brouiller les frontières de l'objet en proposant par exemple une délimitation composée de plusieurs traits indépendants.

Les thèmes qui ont été abordés dans cette étude sont à la fois issus de mes propres préoccupations par rapport à l'image temps réel, mais également d'une constatation des manques ressentis au cours du développement professionnel d'un jeu vidéo. Ces manques concernent par exemple la qualité de certains effets déjà utilisés, tels que la profondeur de champ, la matière, la réflexion ou les fluides.

Nous avons pu expérimenter, avec succès, l'émulation entre technique et art au cours du processus de développement grâce à l'équipe artistique de l'entreprise DONTNOD. Les techniques présentées ont ainsi été testées par les équipes en charge de l'éclairage, des effets spéciaux ou des scènes cinématographiques et ont évolué par itérations successives. La technique est ainsi en constante évolution en fonction des besoins et des idées de chaque corps

de métier. Elle cherche aussi à donner des outils plus directs et intuitifs, qualités qui sont souvent perdues lors du passage d'un médium physique à un médium virtuel. Notre étude a profité de cette collaboration pour apporter des innovations dans différents domaines de l'image 3D temps réel tout en montrant les liens forts entre le photoréalisme et le rendu expressif dans la représentation de scènes virtuelles.

Ouverture

Au cours de cette étude, nous avons essentiellement abordé chaque sujet individuellement, puis apporté quelques pistes d'intégration dans le domaine du rendu expressif. La réunion de tous ces effets conjointement dans une même œuvre mériterait une analyse plus poussée. Les influences de chaque effet sur l'image et les manières de les harmoniser pour créer une ambiance commune et cohérente sont, pour nous, laissées à l'artiste qui utiliserait les résultats de nos différentes recherches. Nous pensons que les avancés dans le domaine du rendu passent par l'amélioration des techniques, mais que celles-ci doivent ensuite pouvoir s'ouvrir aux artistes. Cette ouverture peut passer par la collaboration entre techniciens et artistes au cours même de la recherche technique, ou bien par la création d'outils qui rendent possible une appropriation et un contrôle de l'image par l'artiste.

Si un des objectifs qui à orienté l'évolution du rendu informatique 3D temps réel a été la recherche de la fidélité au réel, nous constatons actuellement une ouverture à des styles graphiques diversifiés. À côté des styles issus des arts picturaux émergent ceux de l'image 3D temps réel en se basant sur des qualités de rendu qui lui sont propres, mais aussi sur l'interactivité numérique et sur l'exploration libre de la scène tridimensionnelle par le spectateur. La puissance de la programmation permet, de plus, de tisser une infinité de liens entre un monde virtuel et sa représentation en image animée sur l'écran. L'image 3D temps réel porte ainsi des promesses de visualisations artistiques inédites.

Table des figures

Références

Baker, Dan. « Spectacular Specular - LEAN and CLEAN specular highlights ». *Firaxis Games* (GDC 2011).

Barla, Pascal. « Modèles de représentation et d'acquisition pour le rendu expressif ». *Institut national polytechnique (Grenoble).* Thèse (2006).

Barsky, B.A., M.J. Tobias, D.P. Chu, et D.R. Horn. « Elimination of artifacts due to occlusion and discretization problems in image space blurring techniques ». *Graphical Models* 67, nº 6 (2005): 584–599.

Beckmann, P., et A. Spizzichino. « The Scattering of Electromagnetic Waves from Rough Surfaces ». *The Macmillan Company, New York* (1963).

Bénard, Pierre. « Stylisation temporellement cohérente d'animations 3D basée sur des textures ». *Université de Grenoble*. Thèse (2011).

Bertalmio, M., L. Vese, G. Sapiro, et S. Osher. « Simultaneous structure and texture image inpainting ». *Image Processing, IEEE Transactions on* 12, nº 8 (2003): 882–889.

Bjorke, Kevin. *GPU Gems -Chaptitre 19 : Image-Based Lighting*. Addison-Wesley Professional, 2004.

Blinn, J.F. « Models of light reflection for computer synthesized pictures ». In *ACM SIGGRAPH Computer Graphics*, 11:192–198, 1977.

Bridson, R., J. Houriham, et M. Nordenstam. « Curl-noise for procedural fluid flow ». In *ACM Transactions on Graphics (TOG)*, 26:46, 2007.

Bridson, R., et M. Müller-Fischer. « Fluid simulation ». In *ACM SIGGRAPH 2007 courses*, 1–81, 2007.

Bruneton, E., et F. Neyret. « A Survey of Nonlinear Prefiltering Methods for Efficient and Accurate Surface Shading ». *IEEE Transactions on Visualization and Computer Graphics* 18, nº 2 (février 2012): 242-260.

Burrell, Tim. « Advected River Textures ». *Dalhousie University (Canada)*. Master (2008).

Burrell, Tim, Dirk Arnold, et Stephen Brooks. « Advected river textures ». *Comput. Animat. Virtual Worlds* 20, nº 2-3 (juin 2009): 163–173.

Castaño, Ignacio. « Seamless Cube Map Filtering », 2012. http://the-witness.net/news/2012/02/seamless-cube-map-filtering/.

Cook, R. L., et K. E. Torrance. « A reflectance model for computer graphics ». *ACM Transactions on Graphics* (1982): 1(1):7-24.

Cook, R.L. « Stochastic sampling in computer graphics ». *ACM Transactions on Graphics (TOG)* 5, nº 1 (1986): 51–72.

Couchot, Edmond. « Art et technique. L'émergence du numérique ». *revue La pensée*, nº 268. collectif Arts de l'ère numérique (1989).

Crane, K., I. Llamas, et S. Tariq. « Real-time simulation and rendering of 3d fluids ». *GPU Gems* 3 (2007): 633–675.

Crassin, C., F. Neyret, M. Sainz, S. Green, et E. Eisemann. « Interactive Indirect Illumination Using Voxel Cone Tracing ». In *Computer Graphics Forum*, 30:1921–1930, 2011.

Crassin, Cyril. « GigaVoxels: A Voxel-Based Rendering Pipeline For Efficient Exploration Of Large And Detailed Scenes ». *Université de Grenoble*. Thèse (2011).

Crassin, Cyril, Fabrice Neyret, Sylvain Lefebvre, Miguel Sainz, et Elmar Eisemann. « Beyond Triangles : Gigavoxels Effects In Video Games ». *SIGGRAPH 2009: Talks*. ACM SIGGRAPH '09 (2009).

Crow, F.C. « Summed-area tables for texture mapping ». *ACM SIGGRAPH Computer Graphics* 18, nº 3 (1984): 207–212.

Cupisz, Robert. « Light probe interpolation using tetrahedral tessellations ». *Conférence GDC 2012* (2012).

Czuba, Bartosz (Behc). « Box Projected Cubemap Environment Mapping », 2010. http://www.gamedev.net/topic/568829-box-projected-cubemap-environment-mapping/.

Donzallaz, Pierre-Yves, et Tiago Sousa. « Lighting in Crysis 2 », Conférence GDC 2011.

Dunbar, D., et G. Humphreys. « A spatial data structure for fast Poisson-disk sample generation ». In *ACM Transactions on Graphics (TOG)*, 25:503–508, 2006.

Durand, F. « An invitation to discuss computer depiction ». In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, 111–124, 2002.

Enrique, S. « Real-Time Rendering Using CUReT BRDF Materials with Zernike Polynomials ». *Topics on Computational Vision and Graphics* (2004).

Fabian « ryg », Giesen. « The ryg blog », 2012. http://fgiesen.wordpress.com/.

Fleming, R. W., R. O. Dror, et E. H. Adelson. « Real-world illumination and the perception of surface reflectance properties ». *Journal of Vision* 3, nº 5 (2003).

Fuchs, Philippe, Guillaume Moreau, Jean-Marie Burkhardt, et Sabine Coquillart. *Le traité de la réalité virtuelle : Volume 2, Interfaçage, immersion et interaction en environnement virtuel*. Presses de l'Ecole des Mines, 2006.

Futuremark. « 3DMark 11 Whitepaper », 2011. http://www.3dmark.com/wp-content/uploads/2010/12/3DMark11_Whitepaper.pdf.

Gotanda, Yoshiharu, et Tatsuya Shoji. « Real-time Physically Based Rendering - Implementation ». *TriACE* (CEDEC 2011).

Gourlay, Michael J. « Fluid Simulation for Video Games », 2010. http://software.intel.com/en-us/articles/fluid-simulation-for-video-games-part-1/.

Graham. « Screen space reflections », 2010. http://forum.beyond3d.com/showthread.php?t=56095.

Gribel, C. J, R. Barringer, et T. Akenine-Möller. « High-quality spatio-temporal rendering using semi-analytical visibility ». *ACM Transactions on Graphics (TOG)* 30, nº 4 (2011): 54.

Gritz, Larry, et Eugene d' Eon. « The Importance of Being Linear ». *GPU Gems 3, Addison-Wesley* (2008): 529–542.

Hable, John. « Filmic Games », 2010. http://filmicgames.com/archives/75.

———. « Uncharted 2 : HDR Lighting ». Naughty Dog - Conférence GDC, 2010.

Harris, M. « Fast fluid dynamics simulation on the GPU ». In *GPU gems*, 1:637–665. Addison Wesley, 2004.

Harvill, Alex. « Effective toon-style rendering control using scalar fields ». In *ACM SIGGRAPH 2007 sketches*. SIGGRAPH '07. New York, NY, USA: ACM, 2007.

Heckbert, P.S. « Filtering by repeated integration ». *ACM SIGGRAPH Computer Graphics* 20, nº 4 (1986): 315–321.

Heidrich, W., P. Slusallek, et H.P. Seidel. « An image-based model for realistic lens systems in interactive computer graphics ». In *Graphics Interface*, 68–75, 1997.

Hensley, J., T. Scheuermann, G. Coombe, M. Singh, et A. Lastra. « Fast Summed-Area Table Generation and its Applications ». In *Computer Graphics Forum*, 24:547–555, 2005.

Hensley, Justin, Thorsten Scheuermann, Montek Singh, et Anselmo Lastra. « Interactive Summed-Area Table Generation for Glossy Environmental Reflections », s. d.

Herzog, R., E. Eisemann, K. Myszkowski, et H. P Seidel. « Spatio-temporal upsampling on the GPU ». In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, 91–98, 2010.

Hill, Stephen. « Specular Showdown in the Wild West », 2011. http://blog.selfshadow.com/2011/07/22/specular-showdown/.

Ikits, Milan, Joe Kniss, Aaron Lefohn, et Charles Hansen. « Volume Rendering Techniques ». In *GPU gems*. Vol. 1. Addison Wesley, 2004.

« j o t », s. d. http://jot.cs.princeton.edu/.

Jensen, L. S., et R. Golias. « Deep-water animation and rendering ». In *Game Developer's Conference (Gamasutra)*, 2001.

Kalnins, R. D., L. Markosian, B. J. Meier, M. A. Kowalski, J. C. Lee, P. L. Davidson, M. Webb, J. F. Hughes, et A. Finkelstein. « WYSIWYG NPR: Drawing strokes directly on 3D models ». *ACM Transactions on Graphics* 21, nº 3 (2002): 755–762.

Kalnins, R. D. « Wysiwyg npr: interactive stylization for stroke-based rendering of three-dimenisional animation » (2004).

Kaplanyan, A., et C. Dachsbacher. « Cascaded light propagation volumes for real-time indirect illumination ». In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, 99–107, 2010.

Kass, Michael, et Davide Pesare. « Coherent noise for non-photorealistic rendering ». In *Proceeding SIGGRAPH '11*, 1. ACM SIGGRAPH, 2011.

Kasyan, Nickolay, Nicolas Schulz, et Tiago Sousa. « Secrets of CryENGINE 3 Graphics Technology ». Crytek, Conférence Siggraph 2011.

Kawase, Masaki. « Anti-Downsized Buffer Artifacts ». Présentation en japonais, CEDEC, 2009. www.daionet.gr.jp/~masa/archives/CEDEC2009_Anti-DownsizedBufferArtifacts.ppt.

Kosloff, T.J., M.W. Tao, et B.A. Barsky. « Depth of Field Postprocessing For Layered Scenes Using Constant-Time Rectangle Spreading ». *Graphics Interface Conference* (2009).

Kovesi, Peter. « Arbitrary Gaussian Filtering with 25 Additions and 5 Multiplications per Pixel » (2009).

« La France compte 25,4 millions de gamers », 2009. http://www.afjv.com/press0910/091008_joueurs_en_france.htm.

Lafortune, E. P. F., S. C. Foo, K. E. Torrance, et D. P. Greenberg. « Non-linear approximation of reflectance functions ». In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 117–126, 1997.

Lagarde, Sébastien, 2012. http://seblagarde.wordpress.com/.

Lambert, J. H. « Photometria sive de mensure de gratibus luminis, colorum umbrae ». Eberhard Klett (1760).

Lauritzen, Andrew, et Michael McCool. « Summed-Area Variance Shadow Maps ». *GPU Gems 3* (2007): 157–182,.

Lee, Sungkil, Elmar Eisemann, et Hans-Peter Seidel. « Depth-of-field rendering with multiview synthesis ». *ACM Transactions on Graphics* 28, nº 5 (décembre 1, 2009): 1.

Lönroth, P., et M. Unger. « Advanced Real-time Post-Processing using GPGPU techniques » (2008).

Martin, Marcel. *Le langage cinématographique*. Collection Septième art. Paris: Ed. du Cerf, 1994.

Masnou, Simon, et Jean-Michel Morel. « Level Lines Based Disocclusion ». *Proc. Int. Conf. Image Processing,* (1998): 259-263.

Max, Nelson, et Barry Becker. « Flow visualization using moving textures ». *Proceedings of the ICASW/LaRC Symposium on Visualizing Time-Varying Data* (s. d.): 1995.

McGuire, M., et J. F. Hughes. « Hardware-Determined Feature Edges ». *ACM* (2004).

Meier, Barbara. « Computers for artists who work alone ». *ACM SIGGRAPH Computer Graphics* 33, nº 1 (février 1, 1999): 50.

Merriam, Websters. *Webster's Ninth New Collegiate Dictionary*. Revised. Merriam-webster+ Inc, 1983.

Mittring, Martin, et Bryan Dudash. « The Technology Behind theDirectX 11 Unreal Engine"Samaritan" Demo ». Epic Games/Nvidia, GDC 2011.

Nehab, D., A. Maximo, R. S Lima, et H. Hoppe. « GPU-efficient recursive filtering and summed-area tables ». *ACM Trans. on Graphics (SIGGRAPH Asia)* 30 (2011): 6.

Ng, Ren. « Digital Light Field Photography ». *Stanford University*. Thèse (2006).

Nilsson, Jonas. « HDR tone-mapping in real time » (2007).

NVIDIA. « The Naked Truth Behind NVIDIA's Demos » (2005).

Olano, M., et D. Baker. « LEAN mapping ». In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, 181–188, 2010.

Oren, M., et S.K. Nayar. « Generalization of Lambert's reflectance model ». In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, 239–246, 1994.

Ostromoukhov, V. « Survol de techniques de rendu non-photoréaliste (NPR) » (2000).

Pichard, Cyril, Sylvain Michelin, et Olivier Tubach. « Photographic Depth of Field Blur Rendering ». *Proc. WSCG 2005* (2005).

Pla-Castells, M., I. García-Fernandez, et R. J. Martinez-Dura. « Physically-based interactive sand simulation ». *Eurographics 2008-Short Papers* (2008): 21–24.

« Poisson Disk Generator », 2010. http://www.coderhaus.com/?p=11.

Potmesil, M., et I. Chakravarty. « A lens and aperture camera model for synthetic image generation ». *ACM SIGGRAPH Computer Graphics* 15, n° 3 (1981): 297–305.

Praun, E., H. Hoppe, M. Webb, et A. Finkelstein. « Real-time hatching ». In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 581, 2001.

« Quel Solaar : Love », s. d. http://www.quelsolaar.com/.

Quilez, Iñigo. « Shader Toy », 2009. http://www.iquilezles.org/apps/shadertoy/.

Ray, Sidney F. *Applied Photographic Optics: Lenses and Optical Systems for Photography, Film, Video, Electronic and Digital Imaging*. Focal Press, 2002.

Reeves, W. T., D. H. Salesin, et R. L. Cook. « Rendering antialiased shadows with depth maps ». In *ACM SIGGRAPH Computer Graphics*, 21:283–291, 1987.

Reinhard, E., M. Stark, P. Shirley, et J. Ferwerda. « Photographic tone reproduction for digital images ». *ACM Transactions on Graphics* 21, n° 3 (2002): 267–276.

Riguer, G., N. Tatarchuk, et J. Isidoro. « Real-time depth of field simulation ». *ShaderX2: Shader Programming Tips and Tricks with DirectX* 9 (2003): 529–556.

rousiers, Charles De. *Rendu réaliste de matériaux complexes*. Grenoble, 2011. http://www.theses.fr/2011GRENM055.

Santella, A., et D. DeCarlo. « Visual interest and NPR: an evaluation and manifesto ». In *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, 71–150, 2004.

Scherzer, Daniel, Chuong H. Nguyen, Tobias Ritschel, et Hans-Peter Seidel. « Pre-convolved Radiance Caching ». In *Computer Graphics Forum*. Vol. 31, 2012.

Seims, Joshua. « Putting the artist in the loop ». *ACM SIGGRAPH Computer Graphics* 33, n° 1 (février 1, 1999): 52.

Sims, Karl. « Choreographed Image Flow ». *The Journal of Visualization and Computer Animation* 3 (1992): 31-43.

Sisson, Dylan. « Soirée PIXAR RenderMan ». Conférence, novembre 10, 2009.

Sousa, Tiago. « Anti-Aliasing Methods in CryENGINE 3 » (Siggraph 2011).

Strengert, M., M. Kraus, et T. Ertl. « Pyramid methods in gpu-based image processing ». *Proceedings Vision, Modeling, and Visualization 2006* (2006): 169–176.

Swoboda, Matt. « Numb Res by CNCD & Fairlight », 2011. http://directtovideo.wordpress.com/.

Szirmay-Kalos, L., B. Aszódi, I. Lazányi, et M. Premecz. « Approximate Ray-Tracing on the GPU with Distance Impostors ». In *Computer Graphics Forum*, 24:695–704, 2005.

Tessendorf, Jerry. « Simulating Ocean Water ». Proceedings of SIGGRAPH 2001, 2001.

Tuong-Phong, B. *Illumination for Computer-Generated Images*, 1973.

Vergne, Romain. « Communication expressive de la forme au travers de l'éclairement et du rendu au trait » (2010).

Vlachos, Alex. « Water Flow in Portal 2 ». In *Siggraph*. Valve, 2010.

Vrellis, Petros. « Petros Vrellis on Vimeo », 2012. http://vimeo.com/user10348450.

Webb, Matthew, Emil Praun, Adam Finkelstein, et Hugues Hoppe. « Fine tone control in hardware hatching ». In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, 53–ff. NPAR '02. New York, NY, USA: ACM, 2002.

Wees, William C. *Light Moving in Time: Studies in the Visual Aesthetics of Avant-Garde Film*. University of California Press, 1992.

West, Mick. « Practical Fluid Dynamics », 2008. gamasutra.com.

Yu, Q., F. Neyret, E. Bruneton, et N. Holzschuch. « Scalable real-time animation of rivers ». In *Computer Graphics Forum*, 28:239–248, 2009.

Yu, Qizhi, Fabrice Neyret, Eric Bruneton, et Nicolas Holzschuch. « Spectrum-preserving texture advection for animated fluids ». *INRIA*, 2009.

Zanuttini, Antoine. « Recherches en rendu non réaliste temps réel » (2008).

Zhu, Y., et R. Bridson. « Animating sand as a fluid ». In *ACM Transactions on Graphics (TOG)*, 24:965–972, 2005.

Du photoréalisme au rendu expressif en image 3D temps réel dans le jeu vidéo : programmation graphique pour la profondeur de champ, la matière, la réflexion, les fluides et les contours.

Cette étude vise à sortir de l'esthétique standardisée des jeux vidéo par l'apport de nouvelles techniques de représentation pour l'image numérique temps réel.

Le rendu dit « photoréaliste » manque souvent de contrôle et de flexibilité pour l'artiste qui cherche à aller au-delà de la fidélité au réel. La crédibilité et l'immersion passent alors par une stylisation de l'image pour proposer un visuel convaincant et esthétique. Le rendu dit « expressif » va plus loin en prenant en compte le regard personnel de l'artiste tout en se basant sur des caractéristiques et des phénomènes issus du réel pour les détourner. Nous montrerons que le photoréalisme et le rendu expressif se rejoignent et se complètent sur de nombreux points.

Trois thèmes différents seront présentés du point de vue du photoréalisme et accompagnés de la création de techniques personnelles. Le thème du flou de profondeur de champ nous amènera à prendre en compte la forme de la lentille de la caméra virtuelle à travers l'algorithme des *Hexagonal Summed Area Tables*. Nous aborderons ensuite la matière, la lumière et surtout la réflexion spéculaire ambiante et l'importance de la parallaxe pour celle-ci. Notre troisième thème sera celui du mouvement des fluides et de l'advection de textures en fonction du courant afin d'ajouter du détail facilement et efficacement.

Ces mêmes sujets seront ensuite intégrés au rendu expressif et utilisés comme des outils d'expression pour l'artiste, à travers la création d'un effet « rêve », de fluides en espace-écran et du travail de la matière hachurée. Enfin, nous présenterons nos créations spécifiquement dédiées au rendu expressif et à la stylisation des contours.

Mots-clés : jeux vidéo / infographie / temps-réel (informatique) / algorithmes / traitement d'image - techniques numériques