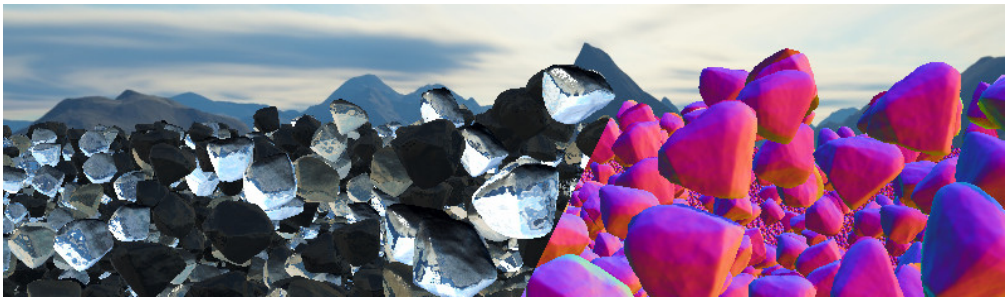
École normale supérieure
École normale supérieure de Cachan
Télécom ParisTech

Rapport de stage

Modèles hybrides géométrie-matériau-éclairage pour le rendu temps réel multi-échelles

Élie Michel

5 mars 2018



Stage réalisé du 1er avril au 15 août 2017 dans l'équipe Image du laboratoire LTCI de Télécom ParisTech, sous la direction de Pr. Tamy Boubekeur

Table des matières

Chapitre 1

Introduction

Le stage défendu dans le présent rapport s'inscrit dans le domaine de l'informatique graphique, dont le but est la synthèse d'images, ainsi que du contenu qu'elles représentent.

La synthèse d'image, ou *rendu* d'image, est *a priori* un problème de simulation de la lumière acquise par un capteur photographique. Mais son critère d'évaluation n'est pas tant l'exactitude physique que la perception de la scène. On peut se permettre des écarts tant que le rendu est *plausible*. Néanmoins, rester le plus proche possible des phénomènes physiques a de nombreux avantages : de stabilité, d'intuitivité des paramètres, et de généralité.

L'informatique graphique regroupe les modèles et techniques permettant de décrire, et de créer, le contenu de la scène virtuelle dont on veut simuler la prise de vue. Il est nécessaire de pouvoir modéliser la façon dont la lumière se diffuse, se réfléchit, évolue entre les sources et le capteur virtuel. Il existe une grande diversité de modèles, selon l'origine de leur donnée (acquise ou créée), leur usage (rendu en temps réel ou non) et le phénomène à modéliser (objet opaque, effet volumétrique, etc.).

En plus de l'information purement visuelle, les modèles contiennent de l'information structurelle permettant leur création et leur modification. La possibilité d'interaction avec le contenu est un enjeu capital, que ce soit pour des usages artistiques, pour les besoins de la visualisation claire de données réelles, ou pour la simulation d'évolution. Par exemple, le modèle peut inclure des propriété mécaniques afin de générer des animations par simulation physique.

Le travail présenté ici porte sur les modèles utilisés pour le rendu, et en particulier sur leur dépendance au point de vue dont est rendue la scène. En effet, en fonction du niveau de détail auquel on observe un objet, son comportement vis à vis de la lumière peut fortement changer. Par exemple, une surface rugueuse plane ne peut plus être modélisée par un plan à l'échelle microscopique. À l'inverse, on ne peut pas se permettre de modéliser les feuilles individuellement lors du rendu d'une forêt en vue d'avion.

Ce besoin d'adaptation du modèle est particulièrement important dans le cadre du rendu en temps réel. Le budget temps alloué à chaque image est alors une fraction de seconde, typiquement entre $1/25$ à $1/120$ seconde (rendu binoculaire), et un maximum de calcul doit être fait en amont, lors de la création du modèle ou d'une phase de chargement. Avoir un *a priori* sur les conditions de vue permet d'anticiper une grande partie des calculs et rend possible le rendu en temps réel de

FIGURE 1.1 – Étapes de construction d'une image du court métrage de synthèse *The Third The Seventh*. À gauche : modèle 3D. Au centre : Rendu intermédiaire. À droite : Image finale. ©*Alex Roman*

certaines scènes autrement trop complexes. Mais si un modèle est voué à être vu sous différents angles, ou à différentes distances, il faut donc le préparer plusieurs fois et correctement effectuer la transition entre les différents points de vue.

On recherche donc à concevoir des modèles hybrides pour le rendu temps réel, tirant partie des spécificité des différentes méthodes existantes en fonction du point de vue, et à assurer la transition entre ces méthodes.

La suite de cette partie introductive présente le contexte dans lequel est mené ce type de recherche, sur le plan technique et scientifique comme sur le plan industriel et artistique. La partie 2 expose les bases des techniques de rendu et de modélisation, tout d'abord de façon générale, puis en s'attardant sur les modèles hybrides. La partie 3 décrit plus précisément le déroulement du stage, ses différentes phases, et les techniques qui y ont été étudiées de façon plus approfondies. La partie 4 conclut ce rapport en proposant de multiples directions d'extension.

1.1 Contexte industriel

Le contexte d'application des techniques de synthèse d'image conditionne les objectifs du domaine. Bien que les travaux de recherche doivent se réserver le droit d'aller à l'encontre des pratiques en place, une nouvelle méthode ne doit pas inutilement les casser. Nous nous intéressons donc ici aux principales applications de l'informatique graphique, et à leurs conséquences techniques qu'il est utile de prendre en compte.

1.1.1 Domaines d'application

Applications artistiques

La synthèse d'image joue un rôle clef dans les domaines artistiques de l'image, qu'elle soit fixe, animée ou interactive. Ces domaines incluent le cinéma d'animation, où chaque image est synthétique (figure 1.1), et les effets spéciaux numériques (ou *VFX*[1]), où des images de synthèse sont intégrées à des images réelles et réciproquement. On y trouve également le jeu vidéo, qui

1. *Visual Effects*. En Anglais, le terme *Special Effects* désigne les effets spéciaux non numériques uniquement, c'est-à-dire ceux qui interviennent pendant le tournage, et non en post-production.

FIGURE 1.2 – Visualisation 3D de scène urbaine avec Google Maps. *©2017 Google et partenaires*

ajoute la contrainte de générer toutes les images en temps réel, car ce sont les actions du joueur qui déterminent la scène à rendre.

Ces domaines artistiques sont liés à des industries de production, dont le marché demande un renouvellement et un perfectionnement permanent des techniques. Le grand public s'attend à être toujours plus impressionné par les prouesses techniques des plus grosses productions cinématographiques ou vidéoludiques. Les studios les plus importants sont donc incités à investir dans la recherche. En plus des production artistiques et de divertissement, les mêmes techniques et outils peuvent être utilisés pour la production publicitaire et la communication marketing.

Visualisation et conception

Une autre application de la synthèse d'image est la visualisation et l'aide à la conception. La visualisation concerne parfois des logiciels grand public comme par exemple la vue 3D de Google Earth (figure 1.2), et sert à des outils spécialisés, comme les outils de conception assistée par ordinateur (CAO). L'ingénieur concepteur doit pouvoir manipuler interactivement la pièce qu'il est en train de dessiner, et le designer doit avoir un aperçu précis de l'apparence finale de l'objet, en fonction du procédé d'usinage.

La CAO bénéficie également des techniques de modélisation et de traitement de géométrie issues de l'informatique dans un but autre que le rendu d'image. Par exemple, l'impression 3D (figure 1.3) met en jeu des modèles numériques dont la visée n'est pas une image mais un objet matériel.

L'architecture aussi utilise des outils de synthèse d'image. L'architecte, et son client, veulent être assistés dans la conception géométrique, mais aussi dans la simulation de l'éclairage, de la façon dont la lumière pénètre un bâtiment en fonction de l'heure ou dont la lumière artificielle se diffuse. Les clients peuvent aussi vouloir faire des visites virtuelles de bâtiments, en particulier

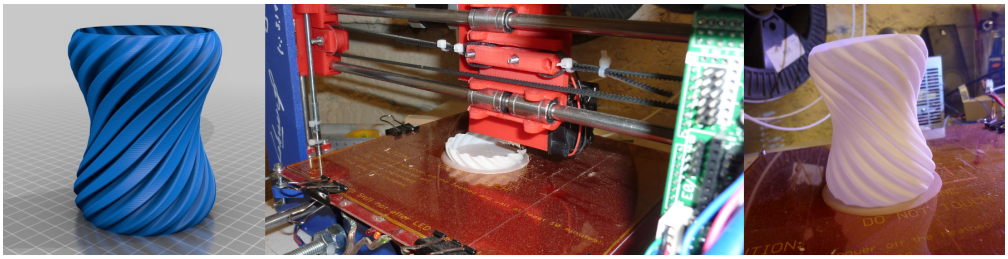


FIGURE 1.3 – Impression 3D, du modèle 3D (à gauche) à l'objet usiné (à droite). *Modèle 3D : ©hakalan*

lorsqu'ils achètent sur plan. Une autre pratique en plein essor est le *home staging* virtuel, qui consiste à refaire la décoration intérieure d'une propriété dans le but de la vendre, mais seulement de façon virtuelle par soucis d'économie.

Enfin, la visualisation de donnée tridimensionnelle est importante dans les domaines scientifiques comme le médical, pour l'étude de scanners et d'images issues d'IRM, la visualisation de données astronomiques, etc. Mais ces applications nécessitent également des mesures précises qui dépassent la seule appréciation visuelle et relèvent donc parfois plus du domaine du traitement du signal.

1.1.2 Étapes de mise en application

Moteurs

Lorsque la recherche propose de nouvelles techniques, ce sont généralement les développeurs d'outils qui s'en saisissent les premiers. Des ingénieurs intègrent ces algorithmes à des solutions techniques appelées *moteurs de rendu*, *moteur de jeu* ou *logiciel de modélisation*. Le but de ces moteurs est de donner accès aux techniques et algorithmes de façon interactive aux *artistes* qui souhaitent réaliser une oeuvre.

Artistes

L'artiste est celui qui crée du contenu, qui manipule les modèles. Les artistes peuvent travailler seuls, mais souvent ces moteurs visent les productions pour lesquelles il est nécessaire de distribuer la création du contenu artistique entre de nombreux artistes, jusqu'à plusieurs centaines. Bien qu'ils deviennent parfois plus des artisans que des artistes, le terme *artiste* reste utilisé pour les désigner, que ce soit au sein des studios ou dans la littérature.

L'artiste doit pouvoir se concentrer sur sa tâche de création, aussi les techniques cherchent-elles à simplifier leur interface, à la rendre intuitive. Des articles issus des domaines de productions comme [BS12] (Disney, cinéma d'animation et VFX) ou [Kar13] (Unreal Engine, moteur de jeu) insistent bien sur ce point. C'est ce qui participe au succès ces dernières années des méthodes dites physiquement réalistes (*Physically Based*), dont les paramètres sont souvent moins nombreux et correspondent à des notions simples à interpréter (rugosité, métallicité, etc.), plutôt que d'être des lettres grecques dans des formules mathématiques (voir section **??**).

L'artiste construit des modèles, en partie par lui-même grâce à des outils, et en partie par acquisition de données réelles. Les enjeux de recherches concernent donc à la fois la création de contenu, et la modification, la simplification, la retouche, de données réelles. Les données mesurées peuvent être des photographies servant de textures, des séquences issues de tournages à incruster, mais aussi par exemple des relevés topologiques, ou des captures de mouvements (*motion captures*) de squelettes humains ou de visages, dont il faut transférer l'animation vers des avatars virtuels et adapter à des silhouettes différentes. Le contenu à créer peut être très riche, regroupant géométrie, matériaux, textures, animations, simulations (de fluides, de fumées, de foules).

Diffusion

Une fois les techniques intégrées aux moteurs, puis le contenu créé par les artistes, ce dernier est visionné par des spectateurs, des joueurs ou des utilisateurs selon le domaine. Le médium de diffusion joue un rôle, car il détermine le type de contenu à créer. Par exemple, pour le cinéma, l'artiste connaît à l'avance le point et l'angle de vue, et peut donc optimiser la scène pour, alors que dans le cas d'un jeu vidéo il faut prévoir un rendu raisonnable sous tous les angles de vue.

Les média de diffusion évoluent, et il est nécessaire de s'y adapter. Cet enjeu est plus immédiat dans le jeu vidéo, qui propose des contrôleurs différents en fonction des plateformes. Entre le jeu pour ordinateur, avec ou sans contrôleur spécial (joystick, volant), le jeu pour console de salon, qui tente régulièrement d'innover (Nintendo Wii, Microsoft Kinect) et mise sur la différence de médium pour attirer joueurs et développeurs, les jeux portables (console de poche, téléphone tactile), les contraintes sont très variables. Peu de jeux sont réellement disponibles pour toutes ces plateformes.

Mais la diffusion cinématographique évolue aussi. La taille d'affichage change (HD, 4K, etc.), la profondeur de couleur également. De plus, de nombreux films sont désormais tournés en 3D stéréoscopique, et les productions sentent venir le besoin de traiter des film en réalité virtuelle, c'est-à-dire en tournés à 360 °. On distingue d'ailleurs la *réalité virtuelle*, composée de scènes intégralement virtuelles, de la *réalité augmentée*, intégrant des éléments virtuels en temps réel aux scènes réelles. Cette dernière concerne plus des outils d'aide ou de visualisation, ainsi que des jeux vidéos, tandis que la première s'applique aussi au cinéma.

Pour certains usages, le médium de diffusion peut être spécialisé. Par exemple, les spectacles de son et lumière mettent souvent en jeu des projection sur des écrans non plans, comme des bâtiments ou des fumées, nécessitant d'être pris en compte très tôt dans la réalisation des scènes virtuelles.

1.1.3 Besoins et contraintes

Les applications rétro-agissent sur la recherche par deux vecteurs : les besoins et les contraintes. Les besoins sont ceux présentés précédemment. Les contraintes sont celles d'intégration des nouvelles méthodes au processus pratique de création et production existant, souvent désigné par le terme *pipeline*.

Contraintes

Même s'il s'agit plutôt du travail des développeurs de moteurs, les chercheurs essayent de prendre en compte ce processus. Comme dans beaucoup de domaines informatiques, une rupture

de rétro-compatibilité ou une transition trop brutale coûte cher. Non seulement en migration des outils de production et d'organisation qui se branchent sur les moteurs (partage des créations entre artistes d'une même production, gestion d'emploi du temps, etc.), mais aussi en formation des artistes, qui doivent alors changer leurs habitudes et perdent donc temporairement la fluidité de manipulation de l'outil.

L'intégration aux *pipelines* est un critère que l'on retrouve donc directement dans les articles de recherche, qui mettent parfois en avant la possibilité de transition entre les méthodes établies et la contribution. Bien entendu, certaines innovations nécessitent tout de même de casser ce processus. La contrainte consiste surtout à ne pas le faire inutilement, par méconnaissance du contexte d'application.

Besoins

L'état de la recherche conditionne, réciproquement, les possibilités de création. Certains effets sont bannis *a priori* par les artistes, dont ils savent leurs outils incapables. Ce phénomène est particulièrement bien illustré par la façon de créer de Pixar [VHK+15], qui choisi le scénario lui-même en fonction de ce qu'il est possible ou non de faire, en particulier pour ses courts métrages.

Par exemple, *Toy Story* raconte l'histoire de jouets principalement en plastique, car lorsqu'il est sorti, en 1995, les matériaux plastiques étaient les plus satisfaisants. Depuis ce film, qui a marqué l'histoire des longs métrages d'animation, de nombreuses évolutions techniques ont amené les scénarios à s'enrichir. *Toy Story 2* montre par exemple beaucoup plus de scènes présentant des humains, qui sont d'autant plus dures à modéliser que notre perception y est sensible. D'autres films jouent avec les nuages, la fourrure, des animations de plus en plus complexes.

La technique impose encore à ce jour un grand nombre de contraintes, notamment pour l'animation. Générer des mouvements humains complexes, comme par exemple un habillement ou déshabillement, reste un vrai défi, même à partir de données acquises, et de nombreux artistes font un travail très manuel.

En plus de définir ce qui est possible et ce qui ne l'est pas, la technique peut influencer le travail artistique, en rendant plus intuitive la manipulation des modèles menant à la solution la plus plausible, par exemple. C'est, encore une fois, un intérêt des méthodes physiquement réalistes.

1.1.4 Acteurs

Cette section présente quelques uns des acteurs industriels les plus influents en termes de recherche et d'applications.

Cinéma

L'application de la synthèse d'image au cinéma se fait dans les studios d'animation ou de VFX. Nous venons de citer Pixar, qui cherche à développer au maximum les techniques de synthèse d'image et écrit des scénarios les mettant en valeur. Disney investit également beaucoup dans la recherche de nouvelles techniques, mais en étant aussi orienté vers les effets spéciaux et le rendu photo-réaliste. De ce côté, deux studios de VFX d'envergure sont Industrial Light and Magic (ILM), issu des production des premiers *Star Wars*, de Goerges Lucas, et Weta Digital, fondé par Peter Jackson pour *Le Seigneur des anneaux*.

Les studios contribuent à l'écosystème technique de l'informatique graphique. Par exemple, ILM a développé le format d'image OpenEXR, adapté aux besoins des effets spéciaux, et désormais très largement utilisé dans ce domaine. Pixar a publié en 2013 le format de description de scène USD, visant aussi à être adopté par la communauté.

Jeu vidéo

De la même façon, certains studio de jeux vidéos ont une politique d'innovation active, en particulier ceux qui commercialisent leur moteur de jeu. Ces studios et éditeurs interviennent généralement à l'annuelle Game Developpers Conference (GDC). On y trouve des éditeurs comme Electronic Arts, Activision/Blizzard, Ubisoft, Microsoft Studio ou Square Enix.

Des acteurs plus techniques participent également, en particulier Nvidia, constructeur de cartes graphiques. Comme pour le cinéma, l'industrie produit des formats de fichier et des standards partagés, notamment via le consortium Khronos, qui spécifie entre autres OpenGL.

CAO

Les deux plus gros éditeurs de logiciel de CAO sont Dassault Systèmes (Catia, Solidworks) et Autodesk (Inventor, AutoCAD). Ce dernier a également racheté une grande partie des logiciels de modélisation utilisés par les studios, comme Maya, 3ds Max, ou Softimage (qu'il a arrêté au profit de Maya).

1.2 Contexte technique

1.2.1 Moteurs et suites logicielles

Comme évoqué précédemment, les divers algorithmes sont réunis au sein de *moteurs* ou suites logicielles. On distingue d'une part les logiciels de modélisation, comme par exemple Blender (projet open source), Houdini (SideFX), ou Maya (Autodesk). Ces logiciels servent à construire les modèles, qui sont ensuite fournis au moteur de rendu. Ce moteur peut calculer un rendu *hors-ligne*, comme Renderman (Pixar), Arnold (SolidAngle), Cycles (Blender) ou Mitsuba (projet open source), ou en temps réel, comme les moteurs inclus dans les moteurs de jeux.

D'autres logiciels incluent des éléments de synthèse d'image assez poussés comme les logiciels de composition vidéo, par exemple After Effects (Adobe) ou Nuke (The Foundry).

Moteurs de jeux

Le moteur de jeu désigne à la fois un éditeur de modèle permettant de construire un jeu, dans tous ses aspects, et à la fois le moteur de rendu *runtime* qui sera partie intégrante du jeu une fois achevé.

Il permet d'éditer les modèles visuels, de les animer, mais aussi de décrire leur logique sous-jacente et de programmer leur comportement. À bien des égards, il est un environnement de développement intégré (IDE) spécialisé dans le jeu, ou du moins les applications 3D interactives, manipulant des modèles en plus du code.

Remarque. *Construire à la fois les données, le modèle, et le programme les exploitant, est une pratique qui commence à se répandre en machine learning, avec des outils comme Azure ML*[2] *par exemple.*

Comme les autres moteurs, le moteur de jeu permet d'isoler le rôle du développeur d'algorithmes de rendu du rôle d'artiste, créateur de contenu. Mais dans le cas d'un jeu, une partie du contenu lui-même est logique, relève aussi du développement. Le moteur permet alors de faire collaborer artiste et développeur sur la création du contenu, et différencie le développeur de moteur du développeur de contenu. [Low14] considère l'apparition de cette distinction entre deux rôles de développement à 1993, avec la sortie du *DOOM engine* de John Carmack, qui marque l'histoire technique du jeu vidéo.

Cependant, certaines personnes ou studios favorisent parfois encore un développement plus vertical, partant de zéro et dans lequel chacun garde en tête l'intégralité des contraintes techniques imposées par le médium. Cette tendance est semble-t-il plus présente dans les studios japonais qu'européens.[3]

Parmis les moteurs de jeux commerciaux les plus utilisés, on peut citer Unreal Engine (Epic Games), Unity, CryENGINE (Crytek), Source (Valve). Les plus grosses productions utilisent parfois leur propre moteur, mais grand nombre de jeux reposent sur l'un de ces quatre exemples.

Plugins, middlewares et outils spécialisés

Les moteurs définissent une architecture souple, au sein de laquelle il doit être aisé de modifier une étape donnée. C'est le rôle des *plugins*. Les moteurs et logiciels de modélisation favorisent la création, et la distribution, de plugins, par le biais d'une interface de programmation (très souvent en Python). L'industrie de la composition vidéo a même défini un standard, *Open Effects*[4], qui permet à un même plugin d'être utilisable dans plusieurs suites logicielles (appelées les *hôtes*) différentes, voire concurrentes.

Certains éditeurs, ou particuliers, se sont ainsi spécialisés dans l'écriture de plugins. Il y a en comparaison beaucoup plus de plugins que de plateformes.

Ainsi, les moteurs et logiciels de modélisation définissent principalement le *pipeline*. Un algorithme qui remet en cause ce dernier nécessite donc une modification du moteur, par ses développeurs, et non uniquement la mise à jour (ou création) d'un plugin.

Dans le cadre des moteurs de jeu, on distingue les plugins des *middlewares*. Les premiers sont des outils pour l'équipe de création du jeu, tandis que les seconds jouent un rôle dans le jeu lui-même, et nécessitent donc d'être adapté"es aux contraintes du temps-réel et du support de diffusion final. Les plugins sont ajoutés au moteurs de construction, les middlewares sont ajoutés au jeu lui-même. Les middlewares sont en général accompagnés d'un plugin pour les configurer.

2. https://studio.azureml.net/
3. Source : Teppei Takehana, animateur de *Metal Gear Solid 4* (Kojima, studio japonnais), puis directeur d'animation de *Beyond : Two Souls* (Quantic Dream, studio français), lors d'une séance du séminaire *Ateliers Interactifs* à la Fémis, printemps 2017
4. http://openeffects.org/

Dans les exemples de middleware, on peut citer ceux qui altèrent les animations à la volée en introduisant de l'aléa humain, par exemple en ajoutant des faux pas, en adaptant la démarche au relief, aux obstacles. On peut également citer ceux qui génèrent du contenu à la volée de façon procédurale, comme les outils d'Allegorithmic [5].

Ainsi, de même que beaucoup d'artistes peuvent contribuer simultanément à une même production, une oeuvre peut composer avec des briques techniques développées complètement indépendamment.

Moteurs de recherche

Un chercheur vise à remettre en cause le pipeline, à expérimenter ses modifications. Il doit donc écrire des éléments de moteurs favorisant la flexibilité. Le moteur de recherche peut être soumis à des contraintes beaucoup plus fortes, ne peut pas se permettre d'hypothèses du type « on ne fera pas ça en production » pour éviter certaines contraintes, et cherche au contraire à explorer les limites des modèles.

Par exemple, un moteur de jeu cohérent ne s'intéressera pas à pouvoir charger dans son éditeur un objet de toute façon trop lourd pour pouvoir être affiché dans le jeu final. Le chercheur peut au contraire vouloir charger cet objet, et se spécialiser sur l'amélioration de son rendu sans avoir à réécrire la procédure de chargement.

J'ai donc été amené, lors de mon stage, à écrire mes propres éléments de moteur, de rendu temps-réel. C'est de plus un exercice très didactique puisqu'il oblige naturellement à apprendre et s'approprier tous les aspects du processus de rendu.

1.2.2 Infrastructure matérielle

Après cet aperçu du contexte logiciel de la synthèse d'image, il est important de donner des précisions sur le support matériel utilisé. En effet, ce support doit être utilisé au maximum de ses capacités par le logiciel, les enjeux de performance étant importants. Ces enjeux sont en fait tels que les besoins logiciels ont rétro-agi sur le matériel pour créer les cartes graphiques, ou GPU (*graphical processing unit*).

Carte graphique

De même que le seul moyen de décoder certains formats vidéos en temps réel a été d'ajouter des instructions spécifiques aux jeux des processeurs, le rendu 3D en temps réel serait bien plus difficile sans avoir écrit une partie du *pipeline* de rendu directement dans la silice. La nature particulière de ces algorithmes ne permettait pas de se contenter d'ajouter des instructions spécifiques au processeur, mais a obligé à construire une unité de calcul complètement différente, organisée à sa façon.

Le GPU ne se programme pas comme le CPU, car une partie de son processus est figée. Seules certaines étapes sont programmables. Certaines opérations sont plus efficaces sur l'un, et d'autres sur l'autre. Mais les échanges de mémoire entre les deux sont lents, donc on ne peut pas simplement faire faire à chacun uniquement ce qu'il fait de mieux.

5. https://www.allegorithmic.com/

Bien qu'à l'origine conçue pour le rendu en temps réel, la forte capacité de parallélisation de la carte graphique a ensuite trouvé son rôle dans les algorithmes de rendu hors-ligne. L'évolution du matériel a pris cela en compte, de même qu'elle a pris en compte les usages non graphiques des GPUs qui ont pu émerger, comme l'entraînement de réseaux de neurones.

Fermes de rendu

Alors que le rendu en temps réel a généralement pour contrainte de devoir s'exécuter sur une unique machine disponible chez le grand public, le rendu hors-ligne, pour générer les images d'un film d'animation ou d'effets spéciaux, peut chercher à exploiter un grand nombre de machines à la fois. On parle alors de *ferme de rendu*.

La façon la plus simple de paralléliser le rendu d'une séquence d'images sur plusieurs machines est simplement de rendre simultanément les différentes trames, une par machine. Mais même cette situation simple peut introduire des défis, comme le besoin d'accès simultané par toutes les machines aux données de la scène.

Et on peut chercher à partager un certain nombre de calculs entre les différents rendus. Il est dommage d'exécuter de façon totalement indépendante deux rendus d'une même scène où la caméra s'est déplacée d'un demi-millimètre. Certains algorithmes cherchent ainsi à exploiter un réseau de machine de la façon la plus efficace.

1.3 Contexte scientifique

1.3.1 Bref historique

1.3.2 Conférences et communauté scientifiques

Les deux groupes majeurs d'organisation de conférence en informatique graphique sont ACM SIGGRAPH, et Eurographics. Chacun possède sa conférence principale, puis des symposiums et conférences spécialisées sont organisés par l'un, l'autre ou les deux.

La conférence annuelle la plus attendue dans le domaine est certainement SIGGRAPH. C'est une conférence très sélective et l'occasion à la fois pour les chercheurs et les industriels d'annoncer des nouveauté, de nouveaux produits ou de nouvelles techniques.

SIGGRAPH possède sont pendant asiatique, SIGGRAPH Asia, et la conférence d'Eurographics est l'équivalent européen. Parmis les conférences plus spécialisées, on trouve I3D, organisé par SIGGRAPH, pour le rendu interactif ; EGSR, orienté rendu ; EG/SIGGRAPH Symposium on Geometry Processing ; SCA (Symposium on Computer Animation), orienté animation, organisé de façon jointe par l'ACM et Eurographics.

Les publications se font souvent par le biais des conférences, et le principal journal est la revue CGF (Computer Graphics forum). Il est à noter qu'il existe un journal orienté *Open Access* et ne préemptant pas de droits aux auteurs, le JCGT. Pour une liste plus complète des conférences et de toutes les publications qui y sont faites, voir [Hua].

Les sponsors qui s'intéressent à un domaine de recherche sont souvent un indicateur intéressant, donc on peut citer, entre autres, pour l'informatique graphique : NVIDIA, Intel, Google, Disney research, Unreal, etc.

Chapitre 2

Rendu et modélisation

Maintenant que nous avons vu de nombreuses raisons de s'intéresser aux problèmes de la synthèse d'image, il est temps d'entrer un peu plus dans les détails techniques. Ce chapitre sert à la fois à introduire le lecteur aux théories et techniques du rendu d'image, et à la fois à rapporter l'apprentissage que j'ai fait de celles-ci lors de mon stage.

Après cette section 2.1 définissant formellement le problème du rendu et les techniques associées, la section 2.2 décrit la modélisation de la scène, c'est-à-dire la façon dont on spécifie à l'algorithme de rendu le contenu qu'il doit traiter. On y décrit les modèles les plus usuels jusqu'à certains plus ésotériques, en expliquant à quelles contraintes ils répondent. Cette introduction permet finalement de correctement exprimer le sujet de ce stage.

2.1 Rendu

Nous allons commencer par une partie un peu théorique, en section 2.1.1, pour y définir un peu plus ce que l'on cherche, en définissant les modèles physiques sur lesquels on se base. C'est aussi l'occasion de définir les notations utilisées par la suite. Cette sous-partie permet de formaliser ce qu'est le problème du rendu, et est suivie de la section 2.1.2, plus pratique, qui donne une vue d'ensemble des différentes techniques permettant de le résoudre.

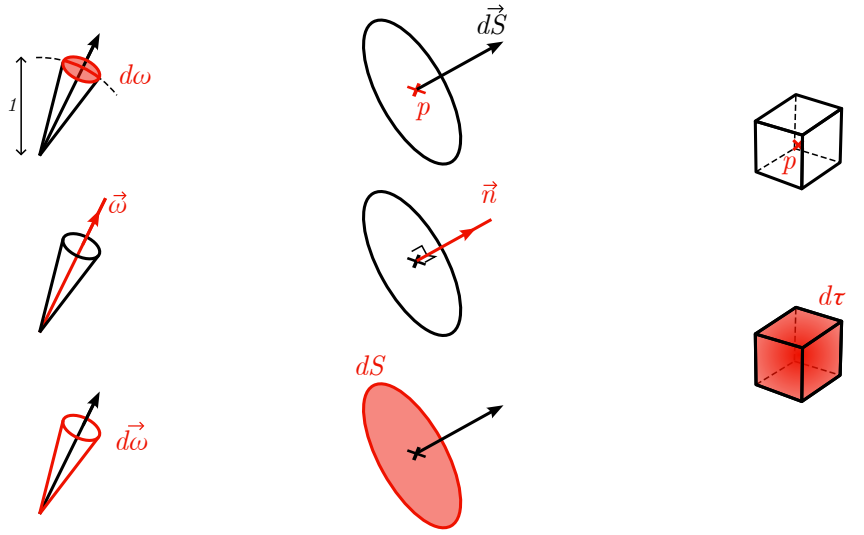
2.1.1 Fondements physiques

Notations

Afin d'alléger les définitions suivantes, commençons par définir brièvement quelques notations courantes, résumées en figure 2.1.

Direction Les directions sont des vecteurs notées $\vec{\omega}$ dont la longueur est supposée unitaire. Puisque l'on raisonne sur des distributions angulaires, on rencontre beaucoup des *cônes élémentaires* centrés autour de $\vec{\omega}$. On note alors le cône lui-même $d\vec{\omega}$, qui est porté par ω et dont la norme est l'angle solide $d\omega$ intercepté par ce cône (voir figure 2.1a).

Surface Un élément de surface est caractérisé par sa surface scalaire dS, sa position p, et son vecteur normal unitaire $\vec{n}$. Le tout constitue d'un élément $d\vec{S} = dS\vec{n}$ de surface *géométrique*, c'est-à-dire un ensemble de point de l'espace, et non uniquement une surface scalaire. Cet élément



(a) Direction (b) Surface (c) Volume

FIGURE 2.1 – Conventions de notations. Lorsque la flèche n'est pas en bout de vecteur, c'est que seule la direction importe.

est de plus orienté : $d\vec{S}$ et $-d\vec{S}$ correspondent au même ensemble de points, mais sont orientés dans des sens opposés (voir figure 2.1b).

Volume Pour les éléments de volume, on confond la notation du volume géométrique et du volume scalaire : on note $d\tau$ l'élément de volume $d\tau$ centré en p (voir figure 2.1c). Aucune direction n'est requise pour caractériser un élément de volume.

En cas d'ambiguïté, on distinguera $p_{d\vec{S}}$ de $p_{d\tau}$. Notons que l'on ne coiffe d'une flèche que les vecteurs traduisant de directions ou de déplacement, pas ceux correspondant à des points de l'espace.

Définitions

Ces notations en mains, nous définissons ensuite les objets physiques qui fondent tout le raisonnement qui suivra.

Photon On modélise le champ d'énergie lumineuse par des *photons*, corpuscules ponctuels porteurs d'une énergie lumineuse. Les transferts d'énergie se traduisent ainsi en déplacement de photons.

L'énergie est une grandeur extensive, et ne se définit donc pas pour un point, mais pour un volume géométrique. C'est pourquoi la position exacte d'un photon n'a pas de sens et nous nous intéresserons donc uniquement à la distribution de cette position. Par exemple, $N(p)d\tau$ est le nombre de photons contenus dans un élément de volume $d\tau$ centré en p.

Un photon est caractérisé par sa fréquence ν, qui est directement liée à son énergie $\epsilon_{\text{photon}} = h\nu$, où h est la constante de Plank [1]. Cette fréquence est constante pour un photon donné. Mais, de même que pour la position, on ne s'intéresse à cette fréquence qu'en distribution. On définit $N_\nu(p)$ comme la densité de photons en espace et en fréquence : $N_\nu(p)d\tau d\nu$ est le nombre de photons contenus dans l'élément de volume $d\tau$, et dont la fréquence est contenue dans un intervalle $d\nu$ centré en ν.

L'énergie lumineuse E d'un volume géométrique V est donc :

$$E(V) = \int_0^\infty \int_V N_\nu(p) d\tau d\nu \tag{2.1}$$

En général, N et E dépendent du temps.

Rayonnement Le déplacement de ces photons est appelé *rayonnement*. Toute propagation d'énergie lumineuse est modélisée par un rayonnement. De même que le photon est associé à une énergie, son déplacement est associé à une puissance :

$$\Phi(V,t) = \frac{\partial E(V,t)}{\partial t} \tag{2.2}$$

Puissance rayonnée La *puissance rayonnée* Φ ainsi définie mesure donc l'énergie *algébrique* transportée par l'intégralité du rayonnement au travers de sa frontière vers le volume V lors d'un intervalle de temps dt autour de t.

Remarque. *Une puissance rayonnée nulle ne signifie pas qu'il n'y a pas de lumière, mais simplement que le volume a pas émis ni plus, ni moins d'énergie lumineuse qu'il n'en a reçue. C'est le cas pour tout volume ne contenant ni source de lumière, ni objet absorbant. Par exemple, le vide, ou un objet parfaitement réfléchissant.*

On s'intéresse le plus souvent à la puissance rayonnée par *une partie seulement du rayonnement.* Par exemple, la partie du rayonnement ayant une certaine direction, ou celle traversant une certaine surface.

Intensité L'intensité lumineuse I est une densité angulaire de puissance rayonnée telle que $I(\vec{\omega}, V, t)\ d\omega$ est la puissance rayonnée à un instant t vers un volume V par la partie du rayonnement dont la direction est comprise dans le cône élémentaire $d\vec{\omega}$.

Éclairement $M(p, \vec{n}, t)\ dS$ est la puissance rayonnée depuis le demi-espace $d\vec{S}^-$, situé « derrière » l'élément de surface $d\vec{S}$, vers le demi-espace $d\vec{S}^+$, situé « devant », par la partie du rayonnement traversant $d\vec{S}$. $M(p, \vec{n}, t)$ est donc une densité surfacique de puissance rayonnée, que l'on appelle *éclairement* (ou *irradiance*).

Remarque. *$M(p, \vec{n}, t)$ dépend de la normale $\vec{n}$ à $d\vec{S}$, et de p, mais pas de la surface dS elle-même, puisque c'est une densité surfacique.*

Remarque. *De même que pour la puissance rayonnée, un éclairement nul ne signifie pas qu'il n'y a pas de lumière traversant $d\vec{S}$. Cela signifie seulement que cet élément de surface n'absorbe ni n'émet de lumière.*

1. $h = 6,62607004 \cdot 10^{-34}\ \text{m}^2\ \text{kg}\ \text{s}^{-1}$

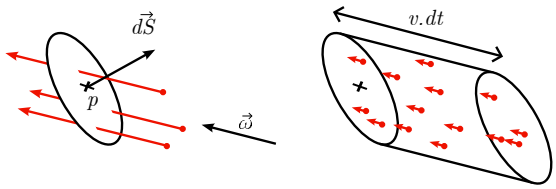


FIGURE 2.2 – Illustration de la radiance, densité de puissance rayonnée au travers d'un élément de surface $d\vec{S}$, dans une direction comprise dans un cône $d\vec{\omega}$.

Luminance La luminance (ou *radiance*) $L(p, \vec{n}, \vec{\omega}, t)$ est à l'éclairement ce que l'intensité est à la puissance rayonnée, c'est-à-dire sa version angulaire. $L(p, \vec{n}, \vec{\omega}, t)\ dS\ d\omega$ est la puissance rayonnée depuis le demi-espace $d\vec{S}^-$ vers $d\vec{S}^+$ par la partie du rayonnement traversant $d\vec{S}$ **et** dont la direction est comprise dans le cône $d\vec{\omega}$.

La figure 2.2 illustre le fait que cette définition correspond à considérer l'énergie des photons dont la direction est dans $d\vec{\omega}$ et situés dans un cylindre élémentaire $d^2\mathcal{C}$ de section $d\vec{S}$, d'axe $\vec{\omega}$, et dont la longueur est la distance parcourue par les photons en un temps dt.

$$\begin{aligned} L(p, \vec{n}, \vec{\omega}, t)\ dS\ d\omega &= N_{d\vec{\omega}}(p)\ d\omega\ d^2\mathcal{C} \\ &= N_{d\vec{\omega}}(p)\ d\omega\ dS\ v\ dt\ (\vec{n} \cdot \vec{\omega}) \end{aligned}$$

où v est la vitesse (supposée uniforme) des photons, et $N_{d\vec{\omega}}(p)\ d\omega$ est la densité de photons en p ayant une direction comprise dans le cône $d\vec{\omega}$. Et donc :

$$L(p, \vec{n}, \vec{\omega}, t) = N_{d\vec{\omega}}(p)\ v\ dt\ (\vec{n} \cdot \vec{\omega}) \tag{2.3}$$

On note à ce titre que :

$$L(p, \vec{n}, \vec{\omega}, t) = L(p, \vec{\omega}, \vec{\omega}, t)\ (\vec{n} \cdot \vec{\omega}) \tag{2.4}$$

Modèles de caméra

Ces dernières définitions nous permettent de modéliser le premier objet indispensable à la capture d'image virtuelle : le *capteur*. Qu'il soit argentique ou numérique, un capteur photographique est un dispositif de mesure de l'éclairement moyen en chacun de ses points ou pixels. Il mesure une moyenne sur un intervalle de temps d'ouverture Δt. En photographie, Δt est typiquement compris entre $\frac{1}{60}$ et $\frac{1}{4000}$. Des temps plus long introduisent du flou de mouvement (ou *flou cinétique*), et des temps plus courts laissent passer trop peu de lumière.

La valeur mesurée est liée à cette énergie surfacique par une courbe de réponse dépendant du type de pellicule et du processus de développement, ou, dans le cas du numérique, de la technologie de capteur et des algorithmes de post-traitement appliqués (a minima le *dématriçage*). Dans tous les cas, on assiste à des phénomènes de saturation (sur-exposition) ou de bruitage (sous-exposition).

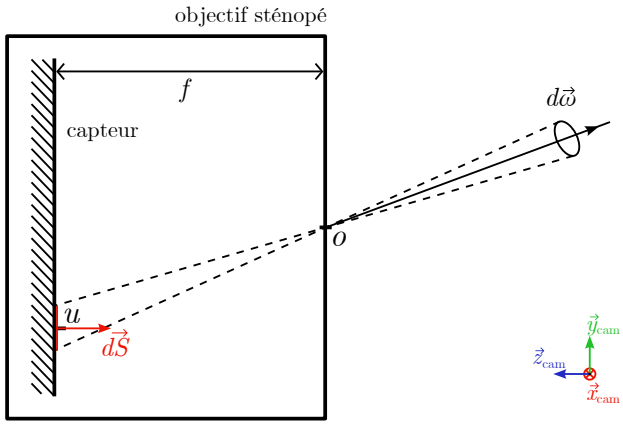


FIGURE 2.3 – Modèle de caméra sténopé. L'objectif est constitué d'un trou de taille infinitésimale dans une paroi infinitésimale, et transforme ainsi les mesures d'éclairement du capteur en mesures de luminance.

Un appareil photo, ou une caméra, est un dispositif de mesure de la luminance. La transformation du capteur, qui mesure un éclairement à plusieurs positions différentes, en un appareil photo, qui mesure la luminance en un seul point mais dans des directions différentes, est assurée par un *objectif*.

L'objectif le plus simple est le *sténopé* idéal, qui consiste en une ouverture infinitésimale située à une distance f du capteur (figure 2.3). Il convertit strictement une direction incidente $\vec{\omega}$ en une position u sur le capteur, selon la projection orthogonale suivante :

$$u = f \left(\frac{\vec{\omega}}{\vec{\omega} \cdot \vec{z}_{\text{cam}}} - \vec{z}_{\text{cam}} \right) \tag{2.5}$$

où $\vec{z}_{\text{cam}}$ est l'axe z du repère de la caméra, et u la position relative à la projection de l'ouverture o du sténopé.

Avec cet objectif, un élément de surface $d\vec{S}$ centré en u du capteur mesure comme éclairement la puissance rayonnée vers la caméra par la partie du rayonnement passant *exactement* par o et dont la direction est comprise dans le cône de projection de $d\vec{S}$ sur o. L'angle solide de ce cône est $d\omega = d\vec{S} \cdot \vec{\omega}$.

Remarque. *La décroissance de l'angle solide de direction lumineuse avec la distance au centre de la pellicule peut être à l'origine d'un phénomène de vignettage, c'est-à-dire un assombrissement de l'image dans les coins. Ce phénomène n'est généralement pas souhaité, et donc l'angle solide des pixels simplement supposé constant lors du rendu.*

Ce modèle est idéal, dans le sens où il n'est pas réalisable en pratique, mais son usage en imagerie de synthèse est très répandu et les autres modèles en sont une variante.

La figure 2.4 montre un exemple de photographie présentant des déformations géométriques et chromatiques telles que peuvent le causer un véritable objectif. L'*aberration chromatique* peut

FIGURE 2.4 – Exemple de déformations causées par un objectif. Les bords montrent la présence d'aberration chromatique, et la planche n'apparaît pas rectiligne, mais légèrement courbe.

être reproduite en employant une longueur focale f légèrement différente pour chaque couleur, et la *distorsion radiale* en ajoutant un terme quadratique à la relation entre u et $\vec{\omega}$. Mais ces effets sont en fait souvent ajoutés en post-traitement, après un rendu avec le modèle idéal.

Un modèle bien plus proche de l'objectif réel, mais aussi plus coûteux à simuler, est le modèle à *lentille mince* (voir figure 2.5). Il modélise le dispositif optique par une lentille convergente unique, et prend en compte l'étendue et la forme de l'ouverture du diaphragme. Ce modèle permet de reproduire le flou de profondeur, tel que présent dans l'image de la figure 2.4.

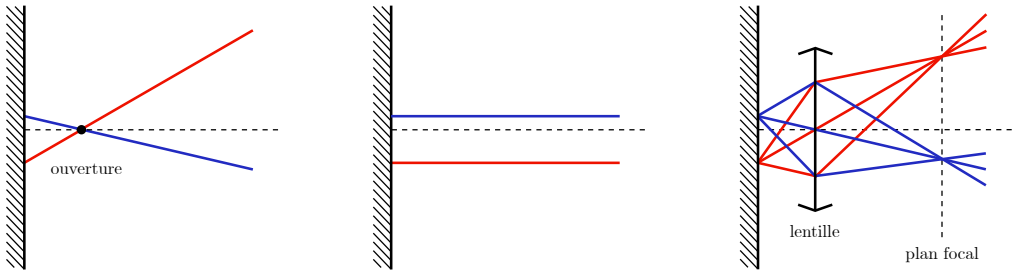
Un autre modèle est également très employé, bien que ne correspondant à aucun dispositif réel : la perspective cavalière. Comme montré en figure 2.5b, chaque point du capteur ne reçoit de lumière que depuis la direction orthogonale, mesurant ainsi $L(u, -\vec{z}_{\text{cam}}, \vec{z}_{\text{cam}})$. [2] Ce modèle est particulièrement apprécié en CAO, et d'autre part nous l'utiliserons par la suite pour pré-enregistrer des informations de luminance dans certains modèles.

Lois de l'optique géométrique

Nous savons maintenant que nous cherchons à calculer une luminance. Mais la seule information de luminance que l'on ait initialement est au voisinage de ce qui a été défini comme des sources lumineuses.

Nous avons nos sujets, les photons et le rayonnement. Nous avons nos grandeurs, puissance rayonnée, éclairement, luminance. Voyons maintenant les lois qui lient ces objets et régissent leurs évolutions. Ce sont ces lois qui vont nous permettre de propager l'information de luminance depuis les sources jusqu'au capteur.

2. ou plus exactement la moyenne de cette grandeur sur un certain temps d'ouverture.



(a) Sténopé (b) Cavalier (c) Lentille fine

FIGURE 2.5 – Schéma de comparaison de modèles sténopé, en perspective cavalière et à lentille fine. Chaque couleur représente le rayonnement contribuant à l'éclairement d'un point du capteur. L'étendue des points par lesquels les rayons peuvent passer dans le modèle à lentille fine est son *ouverture* et sa forme peut varier. Il s'agit souvent d'un disque ou d'un hexagone.

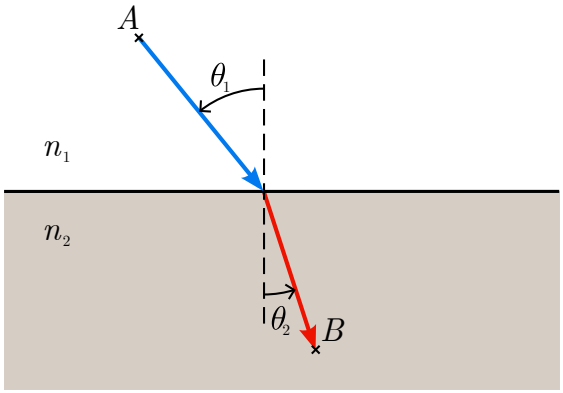


FIGURE 2.6 – Illustration de la loi de Snell-Descartes, qui détermine le chemin le plus rapide du point A au point B étant donné la différence de vitesse entre les milieux 1 et 2.

L'immense majorité des effets visuels quotidiens de la lumière s'explique par l'*optique géométrique*. Nous limiterons donc dans un premier temps nos études à ses lois. Elles déterminent d'une part des chemins suivis par la lumière, puis d'autre part de l'énergie de ces différents chemins. L'ensemble est synthétisé dans l'*équation du rendu*.

Lois géométriques

Les lois déterminant la trajectoire de la lumière découlent du Principe de Fermat :

Principe. *La lumière se propage d'un point A à un point B en empruntant le chemin le plus rapide.*

La vitesse de la lumière dépend de la nature du milieu qu'elle traverse. On note n_M la fraction de la vitesse c de la lumière dans le vide à laquelle la lumière se propage dans le milieu M. La lumière a donc dans ce milieu une vitesse $v_M = n_M c$. Le nombre n_M est appelé *indice de réfraction* du milieu M, en raison de son rôle dans les lois de la réfraction.

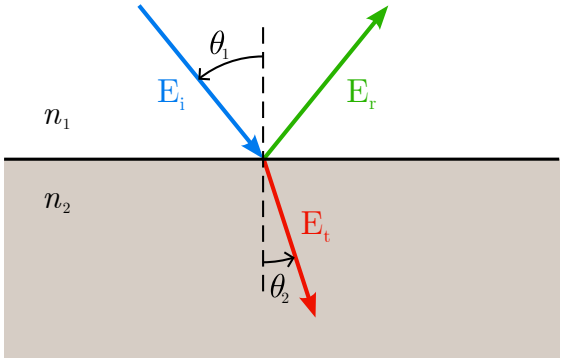


FIGURE 2.7 – Notations pour la loi de Fresnel : E_i est l'énergie incident, E_r l'énergie réfléchie et E_t l'énergie transmise.

Loi de Snell-Descartes La première conséquence du principe de Fermat est la loi de Snell-Descartes, qui caractérise la trajectoire de la lumière à l'interface entre deux milieux homogènes d'indice de réfraction respectifs n_1 et n_2. Avec les notations de la figure 2.6, cette loi donne la relation

$$n_1 \sin\theta_1 = n_2 \sin\theta_2 \tag{2.6}$$

Retour inverse de la lumière Le principe du retour inverse de la lumière, ou principe de réciprocité de Helmoltz, exprime que la lumière suit le même chemin pour aller de A à B que pour aller de B à A. Les algorithmes de rendu exploitent tous ce phénomène, en « remontant » les rayons lumineux depuis les capteurs vers les sources lumineuses. Le chemin direct de la lumière est toutefois utilisé dans ce que l'on appelle le tracer de chemin bi-directionnel (*bidirectional path tracing* [LW93]), qui trace à la fois les rayons depuis le capteur et depuis les sources.

Lois énergétiques

Loi de Fresnel La loi de Fresnel exprime les rapports d'énergies réfléchie $r = \frac{E_r}{E_i}$ et transmise (réfractée) $t = \frac{E_t}{E_i}$, pour une surface ni absorbante ni émettrice d'énergie.

$$r = \frac{n_1 \cos\theta_1 - n_2 \cos\theta_2}{n_1 \cos\theta_1 + n_2 \cos\theta_2} \tag{2.7}$$

$$t = \frac{2n_1 \cos\theta_1}{n_1 \cos\theta_1 + n_2 \cos\theta_2} \tag{2.8}$$

On note que ces coefficients ne dépendent pas de l'énergie incidente.

Conservation de l'énergie La loi de Fresnel vérifie bien la conservation de l'énergie, puisque l'on peut vérifier que $r + t = 1$. Cette conservation doit être prise en compte en tout point de l'espace, mais l'équation doit également intégrer un terme de source ou au contraire d'absorption, exprimant des conversions d'énergie depuis, et vers, d'autre formes d'énergie, non lumineuses. Il en résulte pour tout volume V la relation suivante sur la puissance rayonnée :

$$\Phi(V) = \Phi_{\text{absorbé}}(V) - \Phi_{\text{émis}}(V) \tag{2.9}$$

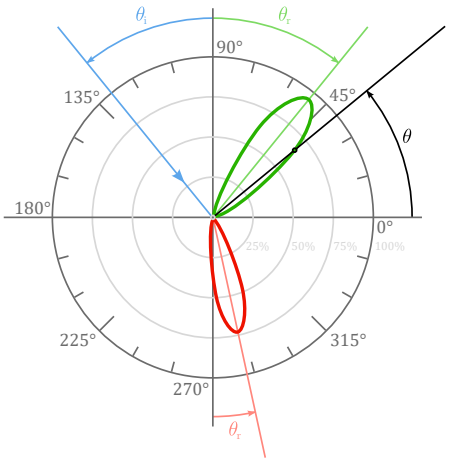


FIGURE 2.8 – Exemple de BRDF sur une interface rugueuse entre deux milieux. La courbe large est le graphe polaire d'une coupe de la BRDF à ω_i fixe (repéré par θ_i).

où $\Phi_{\text{émis}}(V)$ est la puissance de toutes les sources de lumières comprises dans V et $\Phi_{\text{absorbé}}(V)$ est la puissance de conversion de la lumière en chaleur dans ce même volume. En particulier, dans un milieu non absorbant sans source, $\Phi(V) = 0$, et ce même si de la lumière traverse V, et éventuellement y change de direction.

Équation du rendu

L'équation du rendu est une généralisation de la loi de Fresnel au cas où les trajets possibles de la lumière ne sont pas uniquement la réflection et la réfraction parfaites. Elle repose, à la place, sur une distribution de la puissance rayonnée sur les différentes directions possibles, c'est-à-dire l'intensité $I_o(\vec{\omega}_o)$ d'un petit volume $d\tau$ autour du point d'intersection. Donc, pour un rayonnement incident de puissance Φ_i concentré dans la direction ω_i, la puissance rayonnée en retour dans $d\vec{\omega}_o$ est :

$$I_o(\vec{\omega}_o) = f(\omega_o)\Phi_i \tag{2.10}$$

ce qui s'intègre selon $\vec{\omega}_o$ en :

$$\Phi_o = \Phi_i \int f(\omega_o) d\omega_o \tag{2.11}$$

La conservation de l'énergie stipule que $\Phi = \Phi_i - \Phi_o = 0$, et donc que $\int f(\omega_o)d\omega_o = 1$. Cependant, on peut intégrer à f un rôle d'*absorption*, et alors permettre $\int f(\omega_o)d\omega_o < 1$. Cela nécessite de supposer le phénomène d'absorption linéaire en l'énergie reçue, à l'instar des phénomènes de réflexion et de transmission :

$$E_{\text{absorbée}} \propto E_{\text{reçue}} \tag{2.12}$$

Remarque. *Si les phénomènes d'absorption sont toujours supposés linéaires, les phénomènes d'émission, eux, ne le sont jamais. L'émission de lumière est due à des facteurs complètement extérieurs, et est une donnée du problème. Nous prendrons donc soin d'isoler le terme d'émission.*

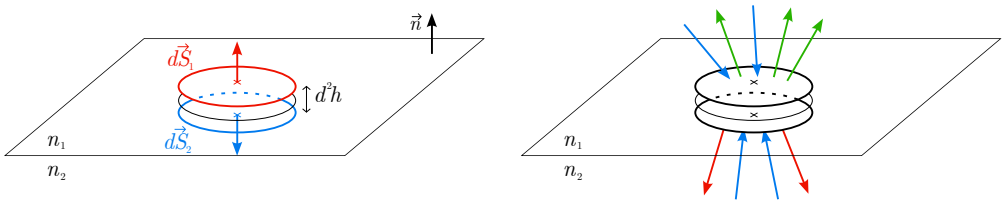


FIGURE 2.9 – Petit élément de volume cylindrique utilisé pour définir les notions de luminance incidente, réfléchie et transmise. Les faces $\mathrm{d}\vec{S}_1$ et $\mathrm{d}\vec{S}_2$ possèdent la même surface scalaire $\mathrm{d}S$. Dans la partie de droite, les flèches rouges, vertes et bleues représentent le rayonnement respectivement transmis, réfléchi et incident.

La fonction coefficient f est a priori différente pour chaque direction incidente $\vec{\omega}_i$, et donc notée par la suite $f(\vec{\omega}_i, \vec{\omega}_o)$. On appelle cette fonction la BSDF, pour *Bidirectional scattering distribution function*. C'est la généralisation des coefficients r et t de Fresnel. C'est une grandeur en « par stéradians ». La figure 2.8 donne un exemple de BSDF pour une interface rugueuse.

Nous faisons ensuite une hypothèse supplémentaire : en plus d'être linéaire en la puissance incidente, l'intensité sortante est linéaire en la distribution angulaire de cette puissance, c'est-à-dire en l'intensité incidente. Si $I_i = I_i^{(1)} + I_i^{(2)}$, alors $I_o = I_o^{(1)} + I_o^{(2)}$. Étant donné que dans tous les cas l'intensité incidente peut être écrite :

$$I_i(\vec{\omega}) = \int_{\vec{\omega}_i} I_i(\vec{\omega}_i)\ \delta_{\vec{\omega}_i}(\vec{\omega})\ d\omega_i \tag{2.13}$$

et que $I_i(\vec{\omega}_i)\ d\omega_i$ est une puissance, on peut appliquer la linéarité et l'équation 2.10 pour obtenir :

$$I_o(\vec{\omega}_o) = \int_{\vec{\omega}_i} f(\vec{\omega}_i, \vec{\omega}_o)\ I_i(\vec{\omega}_i)\ d\omega_i \tag{2.14}$$

En ajoutant un éventuel terme de source $I_e(\vec{\omega}_o)$, l'intensité émise dans la direction $\vec{\omega}_o$, on trouve l'*équation du rendu en intensité* :

$$I_o(\vec{\omega}_o) = I_e(\vec{\omega}_o) + \int_{\vec{\omega}_i} f(\vec{\omega}_i, \vec{\omega}_o)\ I_i(\vec{\omega}_i)\ d\omega_i \tag{2.15}$$

Équation du rendu en luminance

Il reste cependant une zone d'ombre sur cette définition : nous n'avons pas donné de sens aux expressions *intensité émise* et *intensité reçue*. Les lois de Snell-Descartes et de Fresnel suggèrent une approche surfacique de la simulation de la lumière, et pour cela, considérons pour volume $\mathrm{d}\tau$ un petit cylindre dont l'axe est la normale $\vec{n}$ à l'interface entre les deux milieux, de section $\mathrm{d}S$ et de hauteur deux fois infinitésimale d^2h (figure 2.9). L'intensité de ce volume dans une direction $\vec{\omega}$ est :

$$I(\vec{\omega}, \mathrm{d}\tau) = L(p_1, \vec{n}, \vec{\omega})\ \mathrm{d}S + L(p_2, -\vec{n}, \vec{\omega})\ \mathrm{d}S + \mathcal{O}(\mathrm{d}^2h) \tag{2.16}$$

On désigne alors par *luminance incidente* (ou luminance reçue) L_i en l'élément de surface $\vec{dS}$ de l'interface entre les milieux 1 et 2, dans la direction $\vec{\omega}$, la luminance en $\vec{dS_1}$ si $\vec{n}\cdot\vec{\omega}$ est *négatif*, et la luminance en $\vec{dS_2}$ si $\vec{n}\cdot\vec{\omega}$ est *positif*. À l'inverse, la *luminance émise* L_o est la luminance en $\vec{dS_1}$ si $\vec{n}\cdot\vec{\omega}$ est *positif*, et celle en $\vec{dS_2}$ sinon. On peut la scinder en sa composante réfléchie L_r, nulle dans les directions $\vec{\omega}$ telles que $\vec{n}\cdot\vec{\omega}$ est *négatif*, et sa composante transmise L_t, nulle dans les autres directions (voir figure 2.9).

On peut ainsi définir les intensités reçues et émises comme $I_i = L_i\ dS$ et $I_o = L_o\ dS$. On obtient notamment :

$$\begin{aligned} I(\vec{\omega}, d\tau) &= I_i(\vec{\omega}, d\tau) + I_o(\vec{\omega}, d\tau) \\ &= L_i(p, \vec{n}, \vec{\omega})\ dS + L_o(p, -\vec{n}, \vec{\omega})\ dS + \mathcal{O}(d^2h) \end{aligned} \tag{2.17}$$

L'équation 2.15 devient alors :

$$L_o(p, \vec{n}, \vec{\omega}_o) = L_e(p, \vec{n}, \vec{\omega}_o) + \int_{\vec{\omega}_i} f(\vec{\omega}_i, \vec{\omega}_o) L_i(p, \vec{n}, \vec{\omega}_i) d\omega_i \tag{2.18}$$

Afin de s'affranchir de la normale $\vec{n}$ dans l'expression des luminances, on exprime souvent cette équation en fonction des $L(p, \vec{\omega}, \vec{\omega}) = L(p, \vec{n}, \vec{\omega})/(\vec{n}\cdot\vec{\omega})$, notés ici plus simplement $L(\vec{\omega})$:

$$L_o(\vec{\omega}_o) = L_e(\vec{\omega}_o) + \int_{\vec{\omega}_i} f(\vec{\omega}_i, \vec{\omega}_o) L_i(\vec{\omega}_i) \frac{\vec{\omega}_i \cdot \vec{n}}{\vec{\omega}_o \cdot \vec{n}}\, \mathrm{d}\omega_i \tag{2.19}$$

C'est cette équation que l'on désigne par la suite *équation du rendu* et qui fonde les algorithmes de synthèse d'image.

Remarque. *On incorpore parfois $\frac{1}{\vec{\omega}_o\cdot\vec{n}}$, voire même $\frac{\vec{\omega}_i\cdot\vec{n}}{\vec{\omega}_o\cdot\vec{n}}$, à la définition de f. Je préfère ne pas le faire, afin de rester au plus proche de l'origine énergétique de cette équation.*

Propriétés

La loi de Fresnel correspond au cas particulier où :

$$I_o(\vec{\omega}_o)\ d\omega_o\ dt = E_r \delta_{\vec{\omega}_r}(\vec{\omega}_o) + E_t \delta_{\vec{\omega}_t}(\vec{\omega}_o) \tag{2.20}$$

où $\vec{\omega}_r$ et $\vec{\omega}_t$ sont les directions respectivement de réflexion et de transmission, et δ_ω désigne la distribution de Dirac angulaire centrée en ω. Il faudrait plutôt noter dE_r et dE_t, puisqu'il s'agit d'énergies contenues dans le volume élémentaire $d\tau$.

Les autres cas, ceux où la loi de Fresnel ne suffit pas, surviennent lorsque l'on traite d'une réflexion moyenne sur une interface entre les milieux n_1 et n_2 qui n'est pas parfaitement plane. Ces cas découlent en fait d'une loi de Fresnel appliquée à plus petite échelle, pour chaque élément de surface d'ordre deux.

Retour inverse Le principe du retour inverse de la lumière a pour conséquence sur la BSDF que $f(\vec{\omega}_i, \vec{\omega}_o) = f(\vec{\omega}_o, \vec{\omega}_i)$.

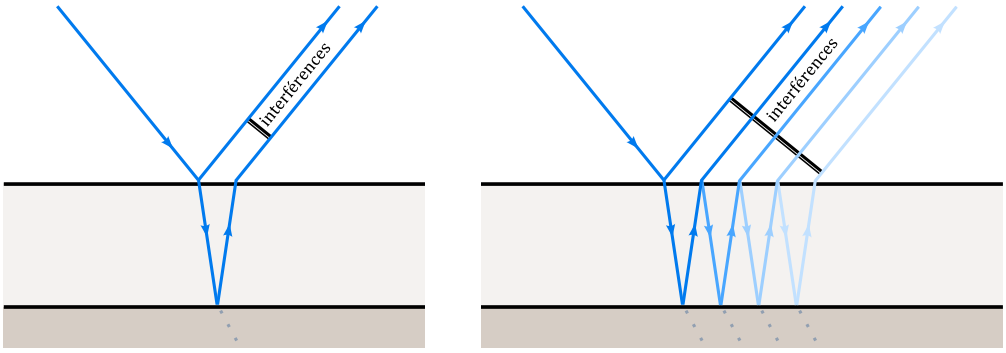


FIGURE 2.10 – Trajectoires lumineuses causant les interférences à l'origine du phénomène d'iridescence. À gauche, un unique rebond sur l'interface entre la couche mince et l'objet est pris en compte, à l'instar de mon implémentation. À droite, de multiples rebonds sont modélisés, ce que fait [BB17]

Limitations

Bien que très utile, l'équation du rendu ne permet pas de modéliser un certain nombre de phénomènes. Par exemple, pour alléger les notations, je n'ai pas inclus la dépendance en temps de toutes les grandeurs, mais elle ne doit pas être oubliée. Nous avons supposé la réflection linéaire en l'énergie incidente *à tout instant*. Mais la ré-émission d'énergie reçue peut avoir lieu un peu plus tard dans le temps. C'est le cas des matériaux dits *phosphorescents*.

D'autre part, nous avons supposé l'absence totale de transferts d'énergie tangents à la surface, au sein d'elle-même, ou d'une petite couche critique, lorsque nous avons considéré comme négligeable la luminance au travers des parois verticales du petit cylindre de la figure 2.9. Mais l'énergie reçue en un point peut en fait être ré-émise depuis un point voisin, légèrement différent. Ce phénomène est appelé *diffusion sous-surfacique*, ou SSS (*subsurface scattering*), et joue un rôle crucial dans l'aspect visuel de matériaux comme la peau, la cire ou les liquides opaques comme le lait. On arrive désormais à reproduire cet effet même en temps réel [JZJ+15].

L'équation du rendu ne traduit pas non plus de certains phénomènes dus à la nature ondulatoire de la lumière. La lumière émise peut être émise dans une longueur d'onde différente de celle incidente (fluorescence). Elle peut subir, dans certains cas extrêmes, l'effet Doppler, notamment lorsque la surface vibre. La réflexion peut également dépendre de la *polarisation* de la lumière, propriété que nous n'avons jusqu'ici pas évoquée. C'est ce qui est utilisé dans les afficheurs à cristaux liquides, ou dans certaines lunettes de cinéma dit 3D.

Iridescence

Enfin, certains phénomènes d'interférence ont des conséquences visibles, comme l'*iridescence*. La figure 2.10 illustre le phénomène à l'origine de cet effet : lorsque deux surfaces sont très proches, par exemple parce qu'un objet est couvert d'un vernis très fin, ou d'une flaque d'huile très étalée, les réflexions sur ces deux surfaces interfèrent. Selon l'angle de vue et la fréquence, l'interférence est constructive ou destructive, donnant une teinte différente.

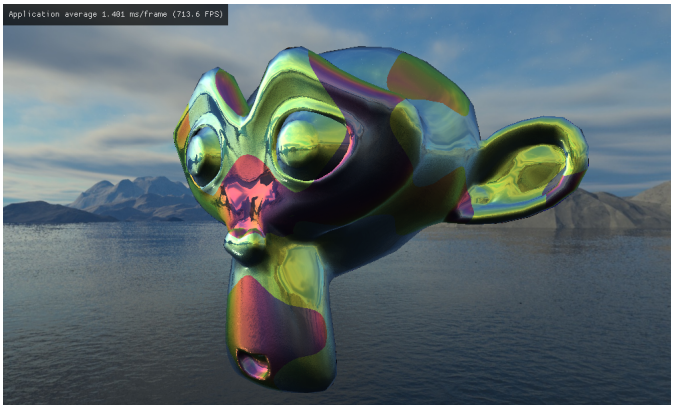


FIGURE 2.11 – Réflexion iridescente sur un matériau métallique couvert d'une fine couche d'épaisseur variable. Les zones plus bleutées sont couvertes d'une couche de 200 nm tandis que les zones plus rouges sont couvertes d'une couche de 500 nm. La couche a un indice de réfraction de 1,45.

Par « très proches », j'entends qu'elle sont espacée d'une distance inférieure à la longueur d'un *train d'onde* de la lumière incidente, aussi appelée *longueur de cohérence temporelle*. C'est, à nouveau, une propriété de la lumière que nous n'avions pas encore abordée. Elle est d'environ 750 nm pour la lumière blanche naturelle, mais peut passer à plusieurs centimètres pour certaines lampes, par exemple à vapeurs de cadmium, et même plusieurs dizaines de kilomètres pour les lasers les plus stables.

Lors de mon étude des modèles de matériaux, j'ai implémenté une approximation d'iridescence (rendue en temps réel), visible en figure 2.11, proche, mais indépendante, de celle présentée par [SM92]. Ce phénomène nous donne l'occasion d'une transition vers une notion très importante en rendu, dont nous n'avons pas encore vraiment parlé : la couleur.

Couleur

J'ai jusqu'ici utilisé le termine « lumineux » de façon abusive. Toutes les définitions précédentes devraient en fait voir ce terme remplacé par « électromagnétique », puisqu'il s'agit de grandeur dites *radiométriques*. Ces grandeurs s'opposent aux grandeurs *photométriques*. Les premières caractérisent un phénomène physique, tandis que les secondes se rapporte à la *perception* que l'on en a. Elles sont intrinsèquement liées à l'œil humain.

La conversion des grandeurs radiométriques aux grandeurs photométriques nécessite d'intégrer les premières par rapport à la longueur d'onde λ (ou la fréquence ν associée), pondérées par la sensibilité de l'œil à ces longueurs d'onde.

$$L_{\text{photo}}(R) = \int_0^\infty c_R(\lambda) L_{\text{rad}}(\lambda) d\lambda \tag{2.21}$$

La sensibilité de l'œil possède trois dimensions : le rouge, le vert et le bleu. Dans cette dernière équation, $c_R(\lambda)$ est la sensibilité des capteurs de rouge de l'œil à la longueur d'onde λ. On l'exprime en lumen par watt (lm.W^{-1}), définissant ainsi le lumen comme une sorte de « puissance perçue ».

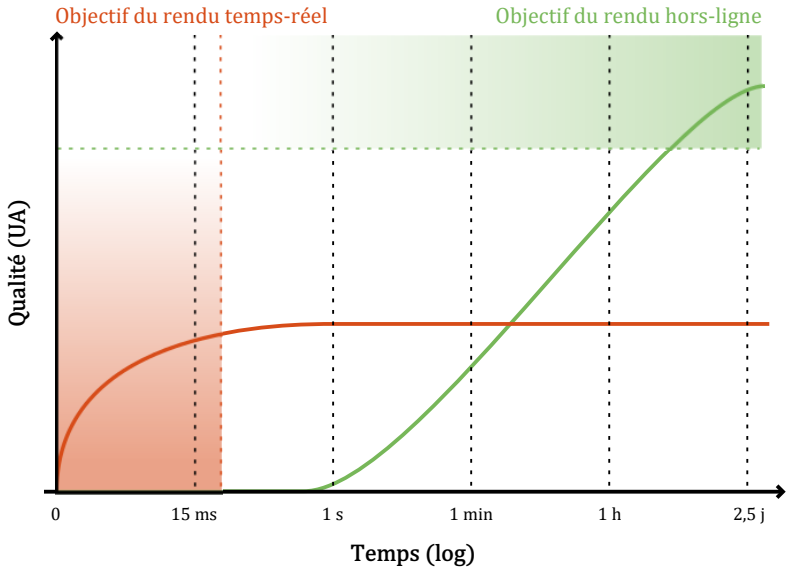


FIGURE 2.12 – Différence de réponse temporelle entre les algorithmes de rendu hors-ligne et temps-réel. Le rendu en temps réel doit atteindre un maximum de précision en un temps fixé de 15 ms à 40 ms, tandis que le rendu hors-ligne doit atteindre une précision fixée en un temps minimal, mais pouvant largement dépasser l'heure.

La valeurs des coefficients $c_R(\lambda)$, $c_V(\lambda)$ et $c_B(\lambda)$ est expérimentale et on peut en trouver des tables, standardisées par le CIE (Commission internationale de l'éclairage), notamment depuis une norme de 1931 faisant encore autorité. Une notion de couleur indépendante de la perception humaine a été également définie par le CIE cette même année : l'espace chromatique XYZ. Ces espaces sont en particulier important pour le bon calibrage à la fois des dispositifs d'acquisition (caméras) et des dispositifs de restitution (écrans, projecteurs).

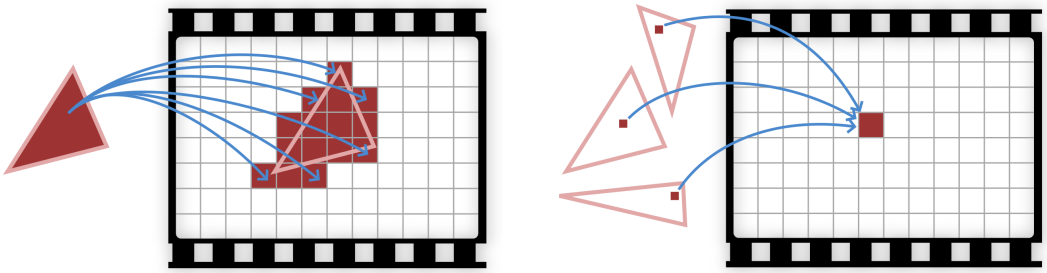
Sauf cas contraire, nous travailleront dans la suite en ayant pré-intégré toutes les grandeurs avec ces coefficients. Par exemple, plutôt que de caractériser les sources lumineuses par leur spectromètre, nous les caractériseront par les composantes photométriques R, V et B. Tous les phénomènes régis par l'équation du rendu, et même certains autres, le permettent. Le principal obstacle survient lorsque l'on veut reproduire des effets comme l'iridescence.

2.1.2 Types de rendu

Selon la contrainte, il existe deux types de problèmes de rendu : le rendu *temps-réel* et le rendu *hors-ligne*. La figure 2.12 illustre cette différence.

Le rendu temps réel doit atteindre la meilleure précision possible en un temps fixe. Tout progrès, aussi important soit-il, n'entrant pas dans cette limite de temps, est inutile. Les coûts fixes des algorithmes ne sont donc souvent pas négligeables.

À l'autre extrémité du spectre, le rendu hors-ligne a pour objectif d'atteindre une précision donnée. Si l'image n'est pas suffisamment précise, quel que soit son temps de rendu, elle ne sera



(a) Approche de la rastérisation (b) Approche du lancer de rayon

FIGURE 2.13 – Deux types de techniques de rendu. La rastérisation consiste à déterminer quels pixels un objet donné influence. Le lancer de rayon, à l'inverse, consiste à déterminer quels objets influencent un pixel donné. Par le jeu des transparences et des réflexions, le nombre de ces objets peut être important. La rastérisation itère sur les objets tandis que le lancer de rayon itère sur les pixels.

pas utilisable dans le film final. J'emploie le terme « précis », et non « réaliste », car la cible peut être un rendu expressif, volontairement non réaliste, tout en ayant des contraintes de précision à respecter.

2.1.3 Techniques de rendu

À ces deux problèmes de rendus sont généralement associées deux techniques permettant de les résoudre (figure 2.13). La première, le lancer de rayon, consiste à tracer un ou plusieurs rayons pour chaque pixel du capteur et à se demander quel objet projette de la lumière sur ce pixel. La seconde, à l'inverse, se demande pour chaque objet sur quels pixels il émet de la lumière. C'est le procédé de *rastérisation*.

Le lancer de rayon est plus utilisé pour le rendu hors ligne, et la rastérisation pour le rendu temps réel, mais rien n'y oblige, et on assiste parfois à des interversions ou mélange de ces techniques.

Lancer de rayon

Le terme de lancer de rayon, *raytracing* en Anglais, regroupe les méthodes basées sur la recherche de point d'intersection entre des rayons et la scène. Le modèle décrivant la scène doit donc être optimisé principalement pour cette opération.

Path tracing La première de ces méthodes est le *path tracing*, qui consiste à lancer itérativement des rayons : un premier rayon est lancé depuis le capteur vers la scène. Au premier point d'intersection donné, on cherche à évaluer numériquement l'équation du rendu. Pour cela, on doit donc échantillonner l'espace des directions, et chaque échantillon nous donne donc un nouveau rayon à lancer, afin d'estimer la luminance incidente. On itère ainsi sur quelques rebonds, même si le coût est exponentiel.

On utilise désormais principalement le path tracing de Monte-Carlo, dans lequel l'intégrale de l'équation du rendu est estimée par une méthode de Monte-Carlo. Cette estimation doit être

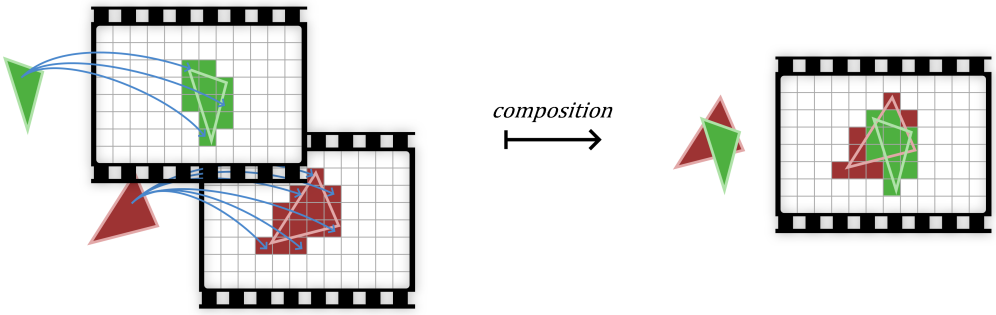


FIGURE 2.14 – La projection des différentes primitives d'une scène sur le capteur donne éventuellement plusieurs *fragments* sur le même pixel. L'opération de *composition* permet de déterminer la valeur finale du pixel.

assistée d'un *échantillonnage préférentiel* (ou *importance sampling*) afin de ne pas « rater » de chemin dans lequel la lumière est particulièrement intense. En particulier, on favorise énormément les échantillons dans la direction exacte des sources primaires de lumière, en particulier si elles sont supposées ponctuelles. Le reste de la préférence des échantillon est plutôt gouvernée par la BSDF : on favorise les échantillons dans la direction où elle est plus importante.

Une variante est le path tracing bidirectionnel [LW93], qui trace également des rayons depuis les sources de lumière, afin de pouvoir préférer, dans l'échantillonnage des chemins venant du capteur, non seulement les sources primaires, mais aussi les premiers rebonds de lumière les plus énergétiques. Cette variante apporte de grande améliorations pour les scènes dans lesquelles l'éclairage est indirect, ou les rayons de lumière primaires restent concentrés.

Comme en témoigne le cours à SIGGRAPH cette année sur le path tracing [FHF+17], cette technique s'est véritablement imposée en production ces dernières années pour le rendu hors-ligne, que ce soit le rendu réaliste pour les effets spéciaux ou le rendu de films d'animation. L'approche de Monte Carlo est en effet très adaptée à la problématique du rendu hors-ligne : on obtient toujours mieux en attendant plus. Un problème est qu'il faut parfois attendre beaucoup, mais différents progrès, en particulier la version bidirectionnelle, permettent de réduire ce temps.

Le path tracing est le sujet de recherche pour le rendu en temps réel également [BvS13]. On parle aussi dans certains cas de *rendu interactif* pour désigner les cas où le temps de rendu est trop long pour le jeu vidéo mais suffisamment court (de l'ordre de la seconde ou moins) pour être permettre aux artistes une visualisation rapide de leur travail et donc une boucle de travail efficace.

Pour plus d'information, une introduction complète au path tracing se trouve dans [PJH16], qui par le jeu du *literate programming* écrit en même temps un livre et le code d'un path tracer.

Rastérisation

La rastérisation adopte une approche toute autre. Elle consiste à projeter chaque primitive de la scène sur le capteur, pour donner une couleur à chaque pixel. Plusieurs objets peuvent influencer le même pixel, aussi on distingue la notion de *fragment*. Un fragment est l'intersection

d'un pixel avec la projection d'une seule des primitives. La couleur du pixel est obtenu par *composition* de tous les fragments qui lui sont associés (figure 2.14).

La scène doit donc être optimisée pour cette opération de projection. En particulier, l'objectif de la caméra suit le modèle sténopé ou cavalier, car la projection sur ces modèles est une projection simple à exprimer. D'autre part, la scène utilise principalement comme primitive des triangles, qui sont simples à projeter.

Ces choix de modélisation ont été entérinés par le fait que le GPU vient accélérer matériellement les opérations de projections orthogonales de triangles et de transformation en fragments.

La composition des fragments peut être complexe lorsque la scène contient des éléments semi-transparents, mais lorsque toutes les surfaces sont opaques (l'énergie lumineuse y est uniquement réfléchie, non transmise), elle se fait grâce à l'algorithme de *Z-buffer*. Lors de la création des fragments, on conserve l'information de profondeur à laquelle ils se trouvent. Une fois tous les fragments prêts, il suffit alors de ne conserver que le plus proche. Le Z-buffer consiste à conserver en mémoire la profondeur (le Z) du fragment actuellement sélectionné. À chaque nouveau fragment, on commence par tester s'il est plus profond ou non que ce Z avant de déterminer sa couleur. S'il est plus profond, on peut l'ignorer. Sinon, il prend la place du précédent et modifie le Z.

En plus de projeter les triangles, le GPU reproduit l'algorithme de Z-buffer. Il est ainsi capable d'ignorer les fragments dont il sait qu'il n'interviendront pas dans le rendu d'un pixel.

Cet algorithme ne spécifie pas comment déterminer la couleur d'un fragment. De nombreuses techniques existent, selon les phénomènes à reproduire. Le problème est de déterminer la luminance incidente. Là où le path tracing itérait le lancer de rayon, la rastérisation ne peut pas se le permettre lorsqu'elle est employée en temps réel. L'approximation la plus simple consiste à ne prendre en compte que les source primaires de lumière, ponctuelles. Mais on perd par exemple les ombres portées.

La rastérisation demande un travail d'extension spécifique à chaque effet recherché (ombres portées, reflets, etc.), mais permet d'obtenir en peu de temps des approximations de rendu, ce qui en fait un bon choix pour le rendu en temps réel. Le lancer de rayon, à l'inverse, ne donne rien d'achevé avant un certain temps, mais est plus général et peut reproduire tous les effets surfaciques tels quels.

Méthodes mixtes

Il y a des méthodes de rendu hybrides, que ce soit pour accélérer le lancer de rayon ou pour rendre plus précise la rastérisation. Il est par exemple possible d'utiliser le lancer de rayon pour pré-calculer certains effets à intégrer dans une scène ensuite rendue par rastérisation. C'est la façon dont fonctionnent les *imposteurs* que nous verrons par la suite.

À l'inverse, il est possible d'effectuer du lancer de rayon *après* la rastérisation, dans l'espace écran. On utilise alors comme information géométrique le Z-buffer. C'est le cas par exemple de la technique de SSAO, pour *screen-space ambiant occlusion*, qui calcule une estimation de l'auto-ombrage des objets, les rendant plus sombre au fond des plis. On trouve plus généralement le SSRT, *screen-space ray tracing* [MM14].

FIGURE 2.15 – Exemple d'image rendue en temps réel par ray marching. Les formes organiques sont caractéristiques de ce que la modélisation par champ de distance rend possible. ©*Iñigo Quilez*

Ray marching

J'ai isolé cette dernière méthode, plus ésotérique, qui n'est ni de la rastérisation, ni du lancer de rayon. Le *ray marching* consiste à tracer les rayons lumineux par petits pas, un peu comme on procéderait pour une descente de gradient. [3]

Il diffère franchement du lancer de rayon, car il ne cherche pas de point d'intersection avec la scène. À la place, il repose sur une description volumique de la scène. Elle est décrite comme un champ de matière, et on n'en accède aux propriétés qu'à un point particulier. Les surfaces y sont implicites.

Le path tracing fonctionne bien pour une scène où les objets sont surfaciques. La prise en compte de milieux participants, dans lesquels la lumière est diffusée en tous points du volume, comme par exemple une fumée, nécessite par contre des adaptations. À l'inverse, le *ray marching* prend naturellement en compte ces effets.

Mais comme toute méthode d'approximation par petits pas, le choix de la taille des pas est important. Prendre de trop petits pas serait une perte de temps. Mais prendre de trop grand pas cacherait certains éléments très localisée du champ de matière. Le problème est que les surfaces sont des éléments infiniment localisés. La solution est alors d'adapter la taille des pas. On ajoute pour cela au champ une information de distance à la surface la plus proche : le *distance field*. Il peut être approximatif lorsque la distance est grande mais doit être exact au voisinage de la surface, pour laquelle il est naturellement nul.

La modélisation par champ de distance est très différente de la modélisation par surface. [Qui08] donne des exemples de modélisation de quelques primitives et d'opérations que l'on peut leur appliquer, et les utilise pour dessiner l'image en figure 2.15. Le principe du ray marching dans les champs de distance est expliquée en figure 2.16, extraite d'une illustration interactive du ray-marching [4] disponible sur Shadertoy. Cette plateforme de partage de *shaders* (programmes pour le GPU) voit beaucoup d'usages de cette méthode.

3. On pourrait essayer de l'appeler « descente de rayons », bien que je n'ai jamais croisé ce terme.
4. https://www.shadertoy.com/view/4dSfRc

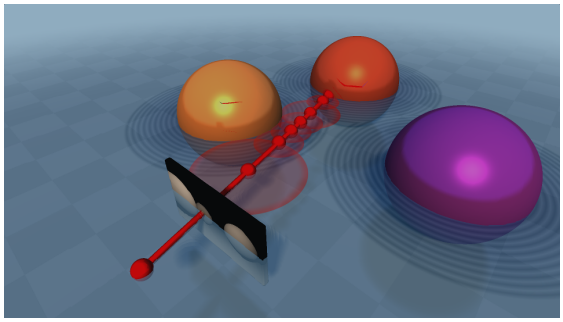


FIGURE 2.16 – Explication du ray marching dans un champ de distance. L'image elle-même est générée par ray-marching. ©*Reinder Nijhoff*

L'hybridation les autres méthode est possible. On peut l'utiliser pour le rendu des effets volumétriques au sein d'un algorithme de path tracing, ou alors en complément à la rastérisation pour certains effets comme le *bump mapping*. Cette technique consiste à reconstituer un relief sur les triangles rastérisés à partir d'une carte de hauteur en effectuant un ray marching de quelques pas depuis chaque fragment.

2.2 Modélisation

Cette section décrit la modélisation de la scène, c'est-à-dire la façon dont on spécifie à l'algorithme de rendu le contenu qu'il doit traiter. Nous avons brièvement vu que la façon de spécifier cette scène dépend de l'algorithme de rendu employé. Et la scène peut servir à d'autres tâches que le seul rendu. La modélisation regroupe ces différentes spécifications, et la relation entre elles.

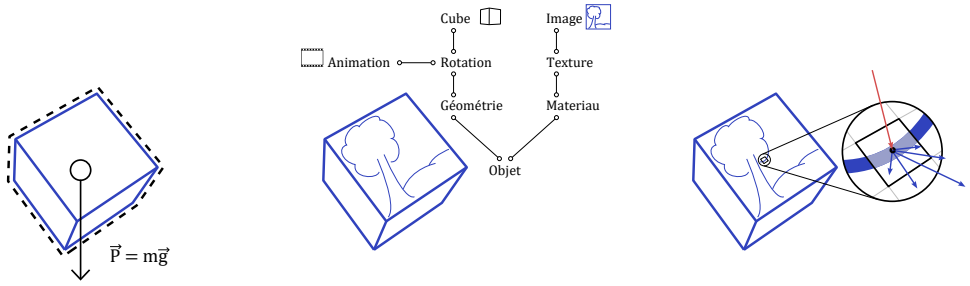
2.2.1 Modèles pour le rendu

Le but du modèle pour la tâche de rendu est de fournir un moyen de déterminer l'intersection de l'objet modélisé, quel qu'il soit, avec un rayon, puis la façon ce rayon peut être réfléchi et/ou transmis. Ou bien, dans le cas de la rastérisation, il fournit un moyen de déterminer la projection de l'objet sur la pellicule virtuelle. Les modèles le plus efficaces sont différents selon la technique de rendu employée.

On veut parfois pouvoir rendre le même objet avec plusieurs techniques. Par exemple, une rastérisation en temps réel est utilisée par les outils interactifs de modélisation, afin de placer les objets pour ensuite en effectuer un rendu par lancer de rayon, plus long mais plus réaliste.

Et pour pour une même technique de rendu, les conditions de vue d'un objet peuvent favoriser une représentation plutôt qu'une autre. Certains modèles sont plus efficaces mais ne permettent que d'être vus de loin, certains modèles imposent des restrictions sur les angles de vue possible, etc. Les différentes représentations d'un même objet doivent donc pouvoir communiquer, être converties de l'une à l'autre.

De plus, les modèles servent au-delà du rendu. La figure 2.17 donne quelques exemples : on veut pouvoir modéliser des phénomènes physiques (collision, déformation), dans le but de générer



(a) Simulation Physique (b) Structure logique (c) Rendu

FIGURE 2.17 – Différents modèles pour différents usages. Les propriétés physiques de l'objet servent à générer des animations par simulation, la structure logique sert à créer et modifier l'objet, et les information de rendu servent pour les calculs de diffusion de la lumière.

certaines animation par simulation, ou conserver les étapes logiques de construction d'un objet afin de pouvoir le modifier de façon cohérente.

Le modèle peut également fournir des aides au calcul, comme des indications d'échantillonnage préférentiel (*importance sampling*) pour le rendu de Monte-Carlo, ou le résultat de calculs préalables comme il est courant d'utiliser pour le rendu en temps réel.

Les besoins sont divers. Pour la rapidité de rendu, on peut vouloir une représentation redondante, évitant de refaire certains calculs. Pour l'édition par l'artiste, il ne faut au contraire aucune redondance, pour éviter le travail fastidieux de modification de tous les points de redondance. L'artiste modifie une *source* qui contient l'ensemble de l'information permettant de mettre à jour les composantes redondantes du modèle. Certains calculs de modèle sont même repoussés aux phases de chargement dans le cas du rendu interactif, car distribuer avec le programme tout ce qu'il est possible de précalculer le rendrait trop lourd.

L'information servant à rendre un objet est généralement répartie entre deux composantes : la géométrie, et le matériau. La géométrie est la forme macroscopique de l'objet, servant à déterminer par exemple sa silhouette, sa capacité à dissimuler d'autres objets et à projeter une ombre. Le matériau est une grandeur synthétisant des propriétés plus fine de l'objet, qui n'influent pas sur sa forme, mais sur la façon dont la lumière est réfléchie dessus. Nous verrons cependant qu'il est possible pour une même information, d'être contenue ou bien dans la géométrie, ou bien dans le matériau.

2.2.2 Géométrie

La représentation de la géométrie est variable. Nous l'avons évoqué lors de la présentation des techniques de rendu : elle peut être volumique ou surfacique, implicite ou explicite. La représentation la plus utilisée est cependant la « soupe de triangle », en particulier parce qu'il est possible de la visualiser en temps réel, grâce aux circuits des cartes graphiques. Il est à noter que la conversion d'une représentation géométrique à une autre peut-être périlleuse, car elles supposent souvent les objets de nature différente. [ED08] donne un exemple intéressant de transformation de triangles en voxels.

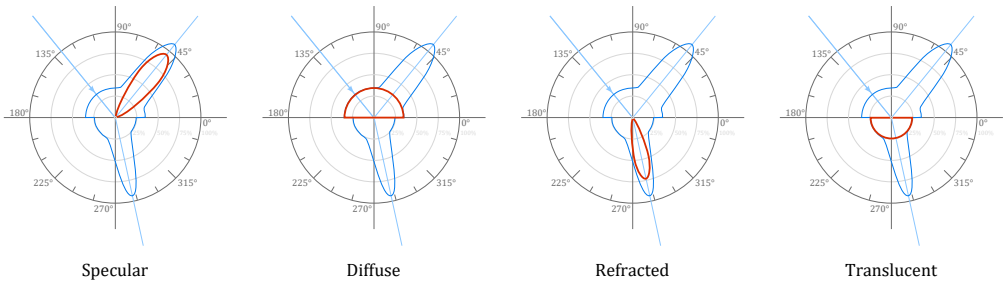


FIGURE 2.18 – Composantes d'une BSDF, fonction de diffusion de la lumière a une surface. Ce sont quelques exemples de phénomènes de diffusion que l'on peut vouloir modéliser.

Le seul traitement de cette donnée géométrique est déjà en soi un domaine de recherche, qui possède ses conférences spécialisées, comme l'*Eurographics Symposium on Geometry Processing*. D'ailleurs, certaines applications, comme l'impression 3D, n'ont quasiment besoin que de la géométrie.

On distingue l'information topologique (connexité, relations entre points, arrêtes et faces), de l'information géométrique (coordonnées dans l'espace euclidien). De nombreux opérateurs peuvent être appliqués à la géométrie, que ce soit pour la créer ou la retoucher : déformation par armature, par cage, opérations de filtrage, morphologie, etc.

2.2.3 Matériaux

Un matériau définit les propriétés de diffusion de la lumière au sein, ou à la surface, de l'objet défini par la géométrie. C'est lui qui détermine la couleur d'un objet, sa brillance, ses aspérités, ses éléments de relief trop petit devant la taille de l'objet pour être pris en compte par la géométrie.

Dans le cas de géométrie surfacique, c'est généralement par la constitution de la BSDF que le matériau est caractérisé. Cette BSDF peut être la même sur toute une surface, ou être paramétrée par des textures, c'est-à-dire des images représentant la variation des différents paramètres du modèle.

Microfacets Models

Le fondement couramment utilisé pour la modélisation des BSDFs est les modèles à micro-facettes. L'idée est de supposer que l'apparence macroscopique d'une surface est en fait due à une constitution de petites facettes, dont les vecteurs normaux suivent une certaine distribution. Le premier est celui de Torrance-Sparrow [TS67], qui formalise par cette approche le modèle, antérieurs, de Phong, et en particulier sa composante rugueuse. Cette approche est reprise par la suite pour donner le modèle de Cook-Torrance [CT82].

Dans les années 90, [Sch94] simplifie énormément les modèles du point de vu du coût calculatoire, tout en tentant de rester aussi proche que possible des modèles théoriquement développés par Cook, Torrance et Sparrow (et autres). Ceci permet l'ouverture des modèles à micro-facettes au rendu temps réel.

L'affinage des modèles à micro-facettes a continué et, plus récemment, [WMLT07] a proposé le modèle GGX, légèrement plus coûteux mais qui s'est très largement répandu car l'amélioration visuelle qu'il apporte est conséquente et permet une meilleure cohérence physique.

Acquisition de matériau La confection de matériau est souvent un mélange de définition de paramètres et de constructions à partir de photographies ou de données relevées. Il existe des appoches plus formelle d'acquisition de matériau d'après réel, comme [DWMG15], qui utilise des profilomètres et reconstruit de très beaux métaux anisotropiques, ou [DHI+15], qui, à partir de données tabulées, donc potentiellement mesurées, fait correspondre des modèles analytiques simples à calculer.

Micro-flakes

Les micro-facettes sont un modèle surfacique, et ont aussi leur pendant volumique : les micro-flocons, plus communément désignés par leur nom en Anglais, *micro-flakes*. Ce modèle a été introduit par [JAM+10], qui y définit la *fonction de phase* comme l'équivalent de la BSDF, dont la sémantique est plus détaillée par [GXZ+13]. Certains modèles sont similaires aux BSDFs, au point que l'on trouve par exemple une généralisation de GGX : SGGX [HDCD15], qui est une distribution de micro-flakes orientée pratique et efficacité.

2.3 Modèles hybrides

L'opposition géométrie/matériau, bien que très pratique, atteint parfois ses limites. Selon le contexte de rendu (occlusion, rendu, collision) et le niveau de détail, la représentation optimale change. Des effets surfaciques peuvent devenir volumétriques, et inversement (cheveux, fourrures, fumées, forêt, sable).

Nous allons voir cette question plus en détail dans le chapitre suivant, puisque le terme *hybride* est justement dans mon sujet de stage, que nous sommes désormais prêts à pleinement décoder.

Chapitre 3

Déroulement du stage

3.1 Introduction

Le stage a duré quatre mois et demi. Le but annoncé dès les premiers jours était d'y intégrer une boucle du cycle de recherche, c'est-à-dire commencer par une phase bibliographique, puis identifier une méthode prometteuse à reproduire, dont tester les limites et explorer les améliorations possibles. Ce rapport conclut la boucle en présentant un retour écrit de ce travail.

Les dix premiers jours ont été consacrés à une remise à niveau en informatique graphique. Mon maître de stage m'a fait écrire un moteur de rendu type *path tracer*. C'était l'occasion de replonger dans l'équation du rendu, les BRDFs, et, sur le plan technique, le C++, OpenGL pour la prévu interactive. Et également un moyen de mettre en place les outils de développement pour la suite, prendre ses marques.

J'ai ensuite abordé pleinement mon sujet, que je rappelle ici, bien qu'il soit le titre de ce document :

Modèles hybrides géométrie-matériau-éclairage pour le rendu temps réel multi-échelles

Le premier terme à considérer dans le sujet est *temps réel*. Bien que les modèles pour le rendu temps-réel et ceux pour le rendu hors-ligne puissent avoir des éléments communs, je cherchais une méthode se comparant, et s'intégrant, aux autres méthodes temps-réel.

Le deuxième terme d'importance est *multi-échelles*. Les modèles multi-échelles intéressent depuis longtemps, car ils sont la clef du rendu d'objets à la fois grands et détaillés, comme un paysage par exemple. On ne peut pas utiliser un modèle trop simple, autrement les vues approchées seraient de trop faible qualité. On ne peut pas non plus utiliser un modèle trop détaillé, car il alourdirait excessivement la scène, alors qu'un grande partie de ces détails, vus de loin, seraient perdus au sein d'un unique pixel. Ou, pire, ils provoqueraient des effets de *moirage* ou de *crénelage* (*aliasing*, en Anglais) dûs au fait que seul l'un des détails primera pour chaque pixel, car en temps réel la transparence, au sens de la contribution de plusieurs primitives au même pixel, est un problème très difficile.

FIGURE 3.1 – Photographie d'une scène illustrant la diversité d'apparence d'un même objet, un arbre, en fonction des conditions de vue (de loin, de près, flou, net).

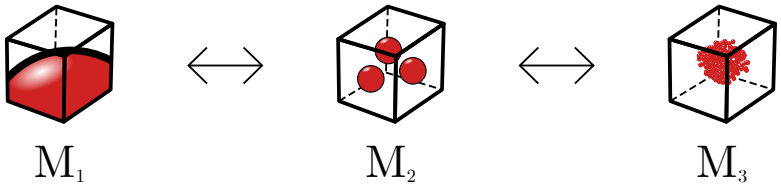


FIGURE 3.2 – Un modèle multi-échelles est un modèle par échelle et un cohérence les liant.

La photographie en figure 3.1 illustre ce problème : le même objet – un arbre – est visible à la fois de loin, dans le flou de profondeur, et de près, dans le plan focal, au point que l'on distingue les nervures des feuilles. On comprend bien que, alors qu'il s'agit du même objet, l'information à fournir à l'algorithme de rendu est complètement différente.

Un modèle multi-échelles est donc un modèle qui, en fonction de la distance à l'objet, fourni une information différente. C'est, en quelque sorte, une séquence de modèles, un pour chaque échelle, et une *cohérence* les liant, permettant qu'aux distances transitoires les modèles de deux échelles consécutives permettent le même rendu (figure 3.2).

3.2 Modèles multi-échelles

3.2.1 Approximation géométrique

Ce problème a bien entendu déjà été étudié. Les premières solutions consistaient simplement à construire plusieurs maillages géométriques, de précision différente, et à remplacer les objets, ou des éléments d'objets, par l'un ou l'autre de ces maillages en fonction de la distance au point de vue (figure 3.3). La construction de ces différentes représentations peut être manuelle, c'est-à-dire effectuée par un artiste, ou automatisée [HDD+93].

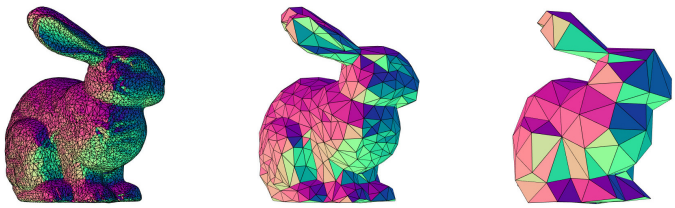
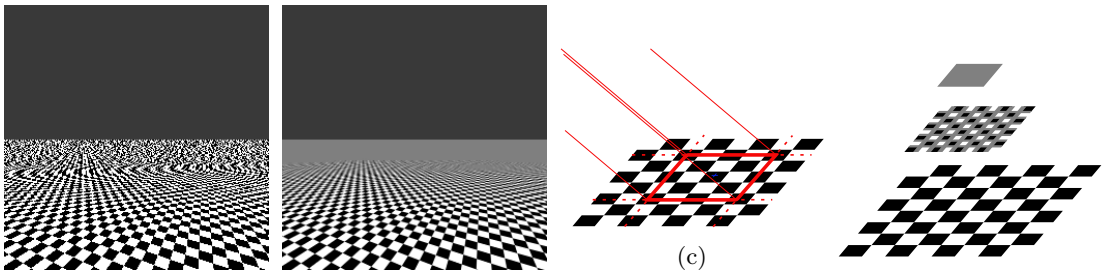


FIGURE 3.3 – Exemple de simplification d'un maillage géométrique



(a) Sans (b) Avec (c)

FIGURE 3.4 – Principe du préfiltrage de *mip-map*. Le crénelage visible en (a) est dû au fait que, lors d'un rendu en temps réel, on ne lit qu'un seul point de la texture pour le rendu d'un pixel donné. Si, comme illustré en (c), l'empreinte du pixel (carré rouge) couvre plusieurs *texels*, seul celui du milieu (croix bleue) est pris en compte alors qu'il faudrait calculer la moyenne de tous ces texels. Les mip-maps sont la pyramide (d) de ces moyennes, précalculées pour des noyaux de taille différentes.

La méthode automatique proposée par Hoppe consiste à fusionner les faces d'un modèle de proche en proche tout en minimisant la divergence au modèle d'origine. Il essaye de conserver la même silhouette tout en diminuant le nombre de faces à traiter lors du rendu à l'aide d'une méthode d'optimisation de forme classique, visant à minimise une énergie constituée d'un terme d'attache aux données et d'un terme d'*a priori* sur le modèle vers lequel converger (le moins de triangles possible).

Malheureusement, ces différents maillages ne sont pas parfaitement équivalents aux distances de transition, aussi cette technique crée des effets d'apparition et de disparition subites de géométrie particulièrement remarquables par un observateur humain lorsque utilisé dans une animation ou un rendu interactif. Hoppe ajouta à sa méthode de simplification automatique un processus de transition progressive [Hop96]. Ce type de méthode a été utilisé avec succès, y compris pour permettre l'affichage de modèles trop détaillés pour tenir dans la seule mémoire vive de la machine [Lin00]. Mais ces techniques sont uniquement fondées sur la géométrie, et échouent alors sur des modèles dont la géométrie n'est pas simplifiable car constituée uniquement d'éléments trop fins, risquant de complètement disparaître. Des objets comme un grillage ou de l'herbe s'accommodent donc très mal de ce type de simplification.

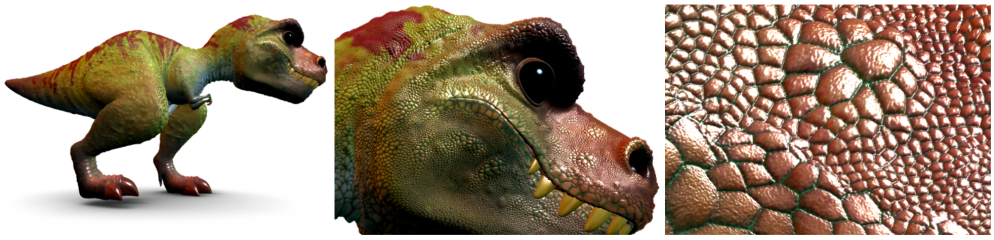


FIGURE 3.5 – Image extraite de [DHI+13] présentant un modèle multi-échelles de texture.

3.2.2 Matériaux multi-échelles

Par ailleurs, la question des modèles multi-échelles a aussi été abordée pour la représentation des matériaux. Le problème le plus immédiat est celui de crénelage qui survient lorsque l'on utilise des textures. C'est l'origine de la technique dite de *mip-map*, qui consiste à *préfiltrer* les textures pour les préparer au rendu à différentes échelles, et que celui-ci ne nécessite qu'une seule lecture de la texture par fragment rendu (voir figure 3.4). Cette idée de préfiltrage est très présente en rendu temps-réel, utilisée partout où les modèles nécessitent en théorie le calcul d'intégrales, que l'on ne peut pas se permettre en pratique.

La technique de mip-map a donc ensuite été étendue au préfiltrage d'autres informations que la couleur, comme le *LEAN mapping* [OB10], qui filtre les cartes de normales, puis le *LEADR mapping* [DHI+13], qui ajoute la prise en compte d'une information de relief, et ses conséquences sur le terme d'ombrage et de masquage, en proposant une approche raisonnant sur un espace des pentes des micro-facettes d'un matériau (figure 3.5). De récents travaux vont jusqu'à chercher à appliquer le principe des mip-maps directement aux BSDF [XWZB17].

3.2.3 Hybridation

C'est alors qu'intervient un troisième terme d'importance du sujet de stage : *hybride*. Nous venons de voir que certains modèles multi-échelles traitent de la géométrie, et d'autres des matériaux, mais le changement de nature de certains objets avec l'échelle est tel que l'on veut parfois *transférer une partie de l'information entre la géométrie et le matériau*, ou entre plusieurs représentations très différentes de ces composantes, en changeant d'échelle. Ces modèles hybrides doivent concevoir ces deux composantes dans un même mouvement, afin de respecter la cohérence du modèle lors de ces transferts.

Par exemple, [BN12] propose un modèle de forêt dans lequel les arbres les plus proches sont complètement modélisés, les arbres un peu plus lointains sont approchés par des imposteurs (voir ci-après), et les vues les plus distantes se contentent de colorer le sol (figure 3.6). Selon la distance, l'information de rendu des arbres est ou bien principalement géométrique, ou bien principalement matérielle.

Un autre exemple récent et très représentatif de cette approche hybride est le travail de [LN17]. En fonction de l'échelle et de la forme des objets, l'algorithme choisi automatiquement un mode de représentation ou un autre. Les éléments de géométrie suffisamment surfaciques *à une échelle donnée* sont représentés par un maillage triangulaire, tandis que les détails fins, comme des tiges

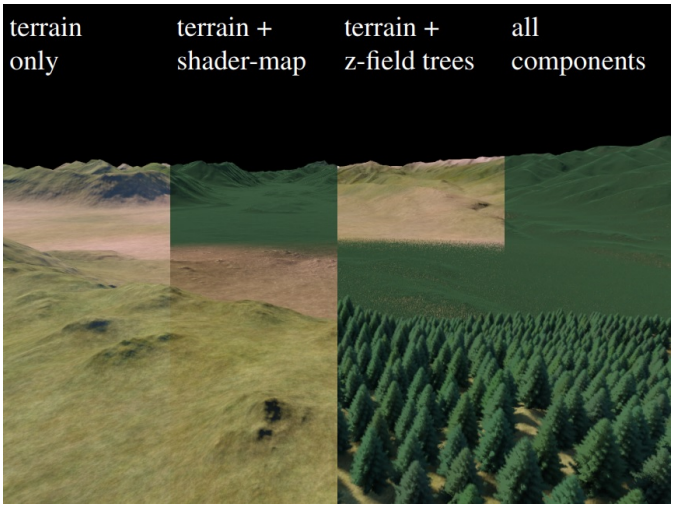


FIGURE 3.6 – Image extraite de [BN12] présentant un modèle multi-échelles de dans lequel les arbres sont transférés de la géométrie, au premier plan, au matériau, à l'arrière plan. La cohérence de la transition est vérifiée et assurée par une conception jointe des modèles rapprochés et éloignés.

FIGURE 3.7 – Images issues du court métrage *Piper* (2016), qui illustre l'état de l'art des techniques de rendu *hors-ligne* de sable ©Pixar

ou des brins d'herbe, sont représentés par des voxels. Selon l'échelle un même élément peut changer cette répartition.

3.3 Objet d'étude : le sable

Afin de diriger mon étude de ces modèles, je me suis intéressé à un objet pour lequel un modèle multi-échelle hybride est particulièrement : le sable. La figure 3.7 donne un exemple de deux rendus d'une même plage à des niveaux de détail différents. Dans l'image de gauche, il est possible de distinguer les grains individuels. Dans celle de droite, l'apparence d'un seul pixel résulte de la participation d'un très grand nombre de grains. L'exemple du sable est intéressant non seulement parce que son apparence varie beaucoup avec l'échelle, mais aussi parce que cette variation d'apparence met en jeu un mélange de la géométrie des grains et de leur matériau. On observe des effets de scintillement ou de translucidité à échelle distante difficiles à modéliser, et il est complètement inenvisageable de rendre la géométrie intégrale de chaque grain (figure 3.8).

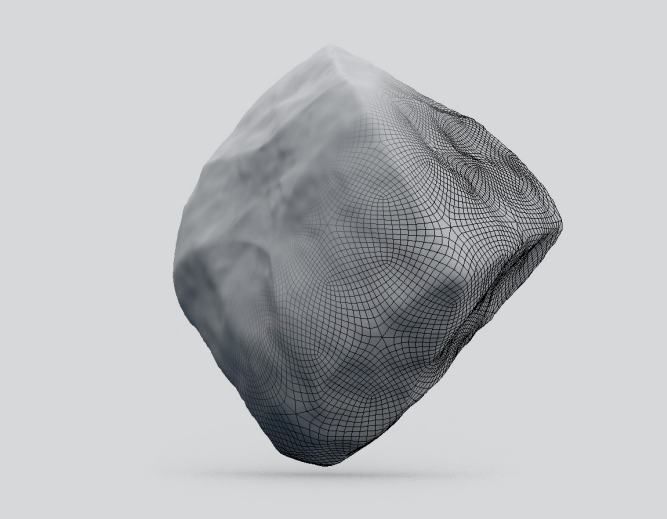


FIGURE 3.8 – Aperçu de la géométrie détaillée d'un seul grain de sable

La modélisation de sable, et les milieux granulaires en général, pour le rendu hors-ligne fait l'objet de recherches spécifiques, et de modèles spécialisés. [MWM07] propose une méthode dite de *Shell Tracing*, traçant des chemins de lumière dont la précision dépend de la profondeur dans le volume de sable. Disney Research s'est dernièrement intéressé à cette question, construisant des méthodes à la fois complexes et intégrées aux outils de production, dont [MPH+15], puis l'année suivante [MPG+16] qui étend ces travaux au rendu de scènes dynamiques, dans lesquelles le volume de sable évolue.

Mais ces méthodes ne sont pas applicables au rendu en temps réel car les modèles y sont très liés à la nature du rendu par path tracing. Peu de travaux se concentrent spécifiquement sur problème du rendu en temps réel de sable, notoirement difficile.

3.4 Modèles existant

3.4.1 Imposteurs

Plans imposteurs

Les premières semaines de mon stage ont consisté à explorer les différents modèles déjà disponibles pour le rendu multi-échelles. J'ai commencé mon étude en m'intéressant à un cas un peu extrême, venant directement de la pratique, des *plans imposteurs*, plus communément appelés *billboards*. Ils sont un cas extrême car la quasi intégralité de l'information qu'ils contiennent sur l'objet qu'ils représentent est contenu dans leur matériau. Leur géométrie est réduite à ce qui, pour le rendu temps réel, est le strict minimum : un triangle (ou deux, pour former un rectangle). La silhouette de l'objet est alors reproduite en jouant sur la transparence du matériau (figure 3.9).

Les plans imposteurs sont particulièrement utilisés dans le jeu vidéo, parfois de façon assez visible. Les arbres par exemple sont (ou étaient) souvent modélisés par une photographie d'arbre

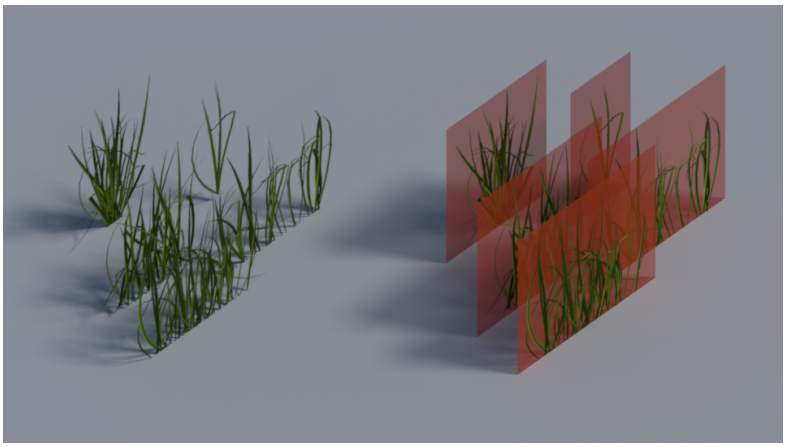
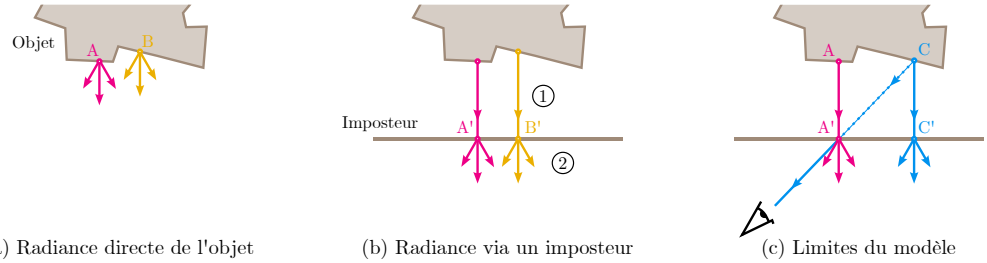
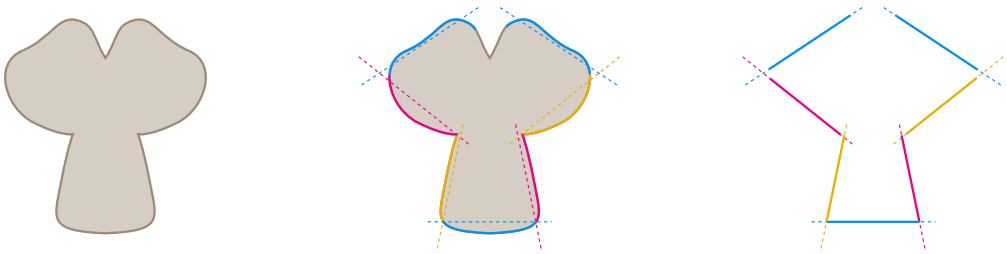


FIGURE 3.9 – Rendu d'herbe par plan imposteur. La partie de droite révèle en semi-transparence leur support géométrique. L'information de géométrie est en fait contenue dans le canal de transparence de la texture d'herbe.



(a) Radiance directe de l'objet (b) Radiance via un imposteur (c) Limites du modèle

FIGURE 3.10 – Interprétation du rendu par plan imposteur. Dans la partie (a), les points A et B émettent une radiance dans chaque direction depuis leur véritable position. Dans la partie (b), une étape préalable ① projette A et B sur leurs images respectives A' et B') dans un imposteur. Lors du rendu ② , la radiance de A et B est émise depuis leur image. La partie (c) illustre l'erreur causée par l'imposteur : la radiance reçue par l'oeil devrait être celle émise par C, et non A.



(a) Objet d'origine (b) Recherche de plans approximants (c) Nuage de plans

FIGURE 3.11 – Illustration de la construction d'un *billboard cloud*.

plaquée sur un plan faisant toujours face à la caméra, ou sur deux plans croisés. Mais le pragmatisme de ce modèle ne le rend pas moins intéressant ; il possède une interprétation rigoureuse.

La partie (c) de la figure 3.10 montre que pour que l'imposteur soit valable, l'angle de vue doit être suffisamment proche de la normale au plan imposteur. Avec les notations de cette figure, l'imposteur est valable si A et C sont à une distance faible devant la distance caractéristique de variation de la radiance. De plus, la résolution à laquelle est discrétisée l'imposteur est telle que la distance $A'C'$ y soit sous-pixellique.

L'usage de plans imposteurs impose ainsi des restriction sur l'angle de vue des objet. Or, dans une vue en perspective optique, la direction de vue varie avec la position sur la capteur. Pour que l'approximation de faible angle avec la normale à l'imposteur soit respectée, il convient donc d'être placé à une distance suffisante de l'objet. Aussi, les plans imposteurs sont généralement utilisés pour des vues lointaines.

Le modèle du plan imposteur pose les bases d'autres modèles, et de la démarche de transfert d'information vers le matériau. Étant donné l'extrême simplicité de sa géométrie, il autorise à investir plus de ressources dans le rendu du matériau, et donc de produire des effets inédits.

Nuages de plans

Une première extension de cette idée, et aussi la première implémentation d'article que j'ai menée, est celle des *billboard clouds* [DDSD02]. L'idée de ce modèle est de conserver le faible coût des plans imposteurs tout en se défaisant de la restriction qu'ils imposent sur l'angle de vue. Pour se faire, la géométrie de l'objet est analysée à l'aide d'une transformée de Hough pour en extraire un jeu de plans approximants. Chacun de ces plans est responsable de rendre une partie de l'objet, qui y est alors projetée pour être remplacée par un imposteur (voir figure 3.11).

La figure 3.12 donne un exemple d'application de cet algorithme, à plusieurs échelles. Cette technique évite la restriction d'angle de vue de deux façons. La première est qu'il y a des plans dirigés dans toutes les directions. Le seconde est que, dans un plan donné, la géométrie d'origine en est très proche, donc les défauts de parallaxe n'apparaissent que pour des angles assez forts. Cependant, cette approche perd la connexité du modèle, ce qui peut parfois poser problème, et n'est en aucun cas satisfaisant pour des vues rapprochées.

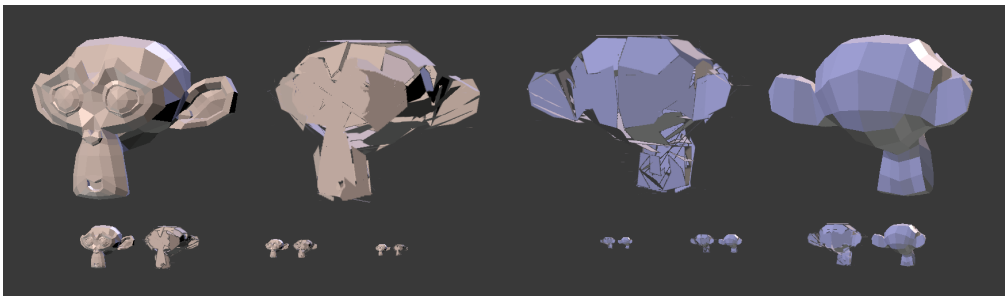


FIGURE 3.12 – Billboard cloud approchant un modèle de 967 triangles par 41 plans

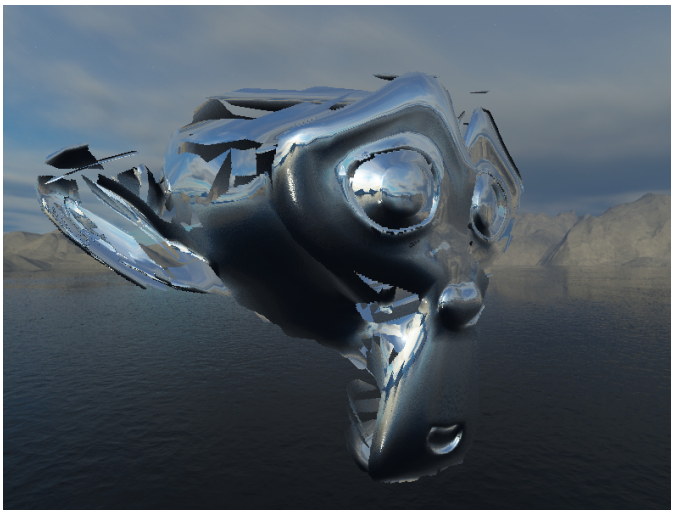


FIGURE 3.13 – Billboard cloud de la figure 3.12 dans lequel on utilise l'information de normale de la surface d'origine plutôt que sa luminance, pour pouvoir changer dynamiquement l'éclairage de l'objet.

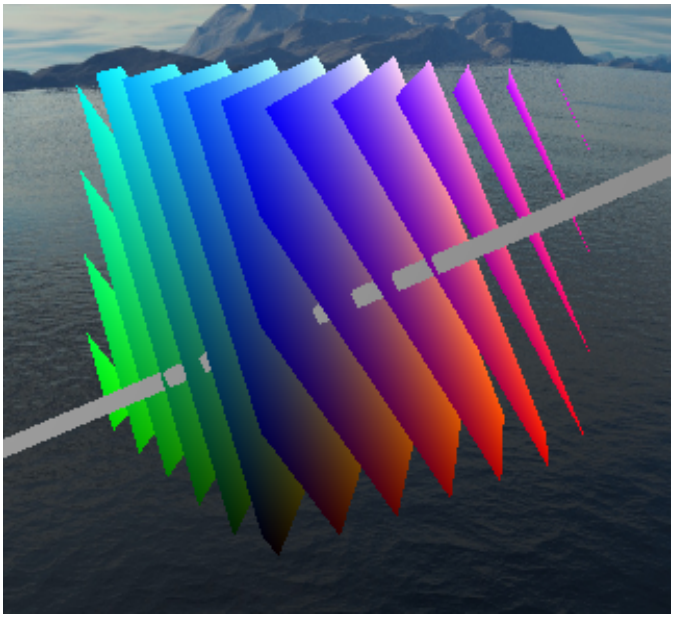
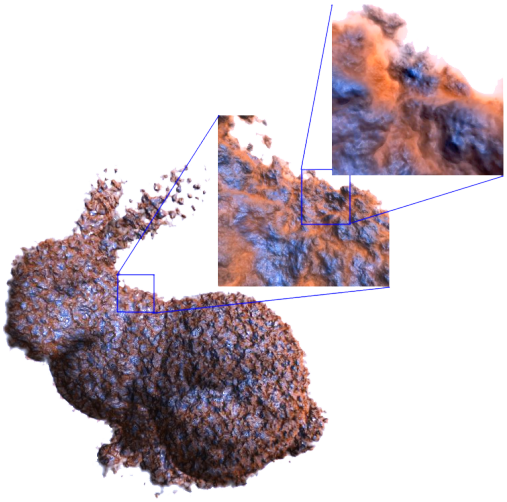


FIGURE 3.14 – Tranchage dynamique d'un volume en fonction de la direction de vue matérialisée par le rayon gris. Cette technique permet de rendre des effets volumétriques et des données voxelliques.

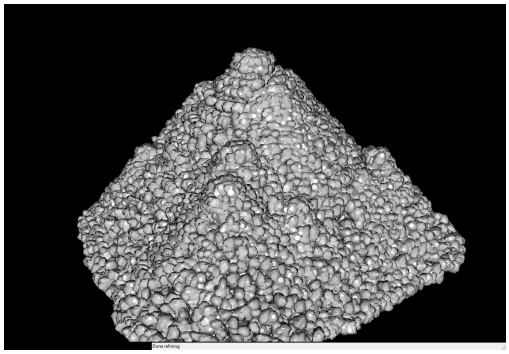
J'ai à cette occasion exploré la possibilité d'enregistrer autre chose que la luminance finale de l'objet dans l'imposteur. À la place, on peut y enregistrer des informations sur la géométrie d'origine comme sa normale en chaque pixel, ou sa distance au plan imposteur. La figure 3.13 montre le nuage de plans de la figure 3.12 pour lequel la luminance est calculée dynamiquement en fonction de l'angle de vue (puisque c'est un matériau très réfléchissant). La lumière est réfléchie en fonction de la géométrie d'origine et non de celle des plans, alors que seuls les plans sont réellement rendus.

Imposteurs volumétriques

Une autre méthode liée aux plans imposteurs que j'ai passagèrement explorée est celle des imposteurs volumétriques (*volumetric billboards*) [DN09]. C'est une technique utilisée pour rendre des effets fondamentalement volumétrique tout en respectant la contrainte imposée par le GPU qui force à ne rendre que des triangles. L'idée est de trancher *à la volée* un volume donné, comme illustré en figure 3.14, afin de le rendre par plans. Un des avantages de construire la géométrie à la volée est de pouvoir l'ordonner. On peut choisir de commencer par tracer le plan le plus profond, puis revenir vers le point de vue. Cette possibilité rend possible l'utilisation de la transparence, ce que l'algorithme du Z-buffer, utilisé en temps réel, ne permet pas pour une géométrie non triée. Cette technique est en particulier utilisée pour rendre des données médicales, données sur une grille de voxels.



(a) Illustration extraite de la présentation de GigaVoxel [CNLE09]



(b) Tas constitué de 12 millions de triangles rendu avec QSplat [RL00]

FIGURE 3.15 – Modèles arborescents/hiérarchique qui permettent le rendu à toute échelle sans besoin de transition.

3.4.2 Méthodes arborescentes/hiérarchiques

Voxels

La raison pour laquelle je me suis intéressé aux imposteurs volumétrique est que la modélisation par voxels (*vo*lume *el*ements), le x est là par analogie avec *pixel*), permet une approche fondamentalement multi-échelle. Il est possible, en les organisant en arbre, de construire un modèle qui soit utilisable à toutes les échelles. Plutôt, comme vu précédemment, d'assurer la transition entre plusieurs modèles différents, cette approche permet d'avoir un unique modèle. Le potentiel de ces arbres, en particulier des *Sparse Voxel Octrees* (SVO), est démontré par GigaVoxel [CNLE09] (figure 3.15a), et a depuis vu des améliorations significatives. Par exemple, on peut étendre l'arbre à un graphe orienté acyclique [KSA13] afin de factoriser les motifs communs, dans un arbre de voxels binaire. Et, pour ajouter des informations non binaires comme la couleur, on peut utiliser les récents résultats de [DSKA17].

L'utilisation de représentation arborescente de voxels peut cohabiter avec de la géométrie à base de triangles. Nous l'avons vu avec [LN17], qui construit un modèle multi-échelles mélangeant surfaces et voxels, mais on peut aussi utiliser les voxels comme structure d'organisation des triangles, ce que fait [**?**], également dans le but d'optimisation multi-échelle.

QSplat

Une autre représentation arborescente de géométrie est QSplat [RL00]. Il ne se fonde pas sur une grille discrète de voxels, mais sur une transformation de la géométrie en nuage d'éléments de surface placés dans l'espace continue, et dont on précalcule les propriétés de visibilité et de normale à différente échelles, un peu comme on précalcule des mip-maps. QSplat est bien antérieur à GigaVoxel, et a permis au moment de sa parution l'affichage d'objets constitués de plusieurs centaines de millions de points (affichage progressif, mais interactif).

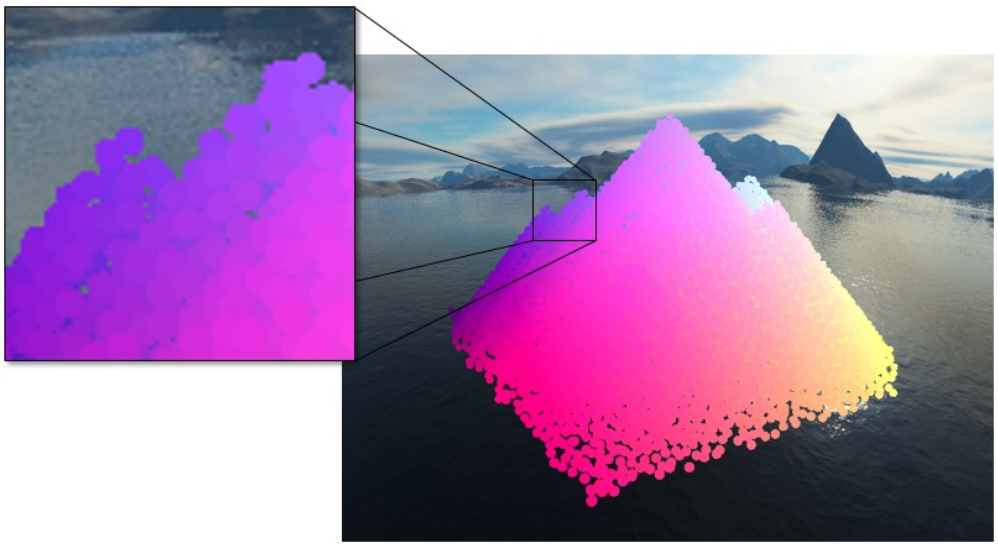


FIGURE 3.16 – Base du nuage d'imposteurs utilisé pour rendre un tas de sable. Chaque tache de couleur différente est un petit imposteur.

3.5 Développements

3.5.1 Direction de recherche

Revenons à mon fil rouge qu'est le tas de sable. Comme le montre la figure 3.15b, j'ai appliqué QSplat à son cas. J'ai à cette occasion étudié le code de QSplat et l'ai intégré au programme développé dans le cadre de mes divers tests. Cependant, QSplat a été prévu pour visualiser uniquement de la géométrie, dans le cadre d'un projet de sauvegarde du patrimoine qui générait des scans très précis de sculptures, mais n'a pas du tout chercher à modéliser d'information de matériau.

J'ai cherché à fusionner une approche par hiérarchie de points comme peut l'être QSplat avec le modèle beaucoup plus orienté matériau qu'est l'imposteur, créant un *nuage d'imposteurs* (figure 3.16). Mon idée était d'utiliser un arbre sur le nuage de positions de grains pour déterminer quels imposteurs rendre au moment de la rastérisation de la scène, puis, au moment de déterminer la couleur des fragments, d'utiliser le résultat de calculs d'imposteurs faits en amont. C'est surtout sur cette seconde partie que je me suis concentré.

La question que je me posais se résumais donc à savoir quoi tracer dans les imposteurs. Leur rendu se fait après la phase de rastérisation, pixel par pixel. L'approche a adopter est alors plus celle du lancer de rayon : mes premiers tests ont donc consisté à rendre ce qu'il y a de plus simple à dessiner par lancer de rayon : des sphères. La figure **??** montre une deuxième étape, complexifiant vaguement le modèle de « grain ». J'aurais pu complexifier cette génération procédurale au point de reproduire la méthode, présentée en figure 3.18, avec laquelle j'ai généré mes géométries de grains.

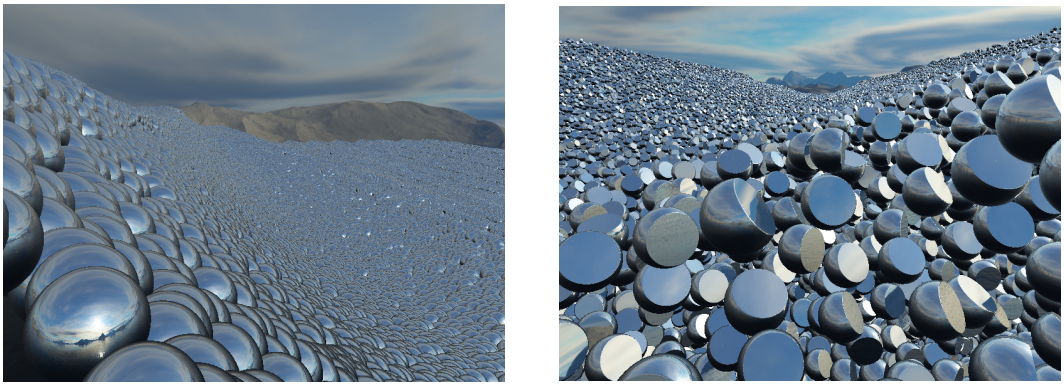


FIGURE 3.17 – Imposteurs rendus par lancer de rayon sur des formes géométriques simples. Le rayon est réfléchi sur la surface paramétrique (sphère ou sphère coupée dans ces exemples) vers la carte d'environnement, donnant un aspect métallique, bien qu'il n'y ait pas de réflexions inter-grains.

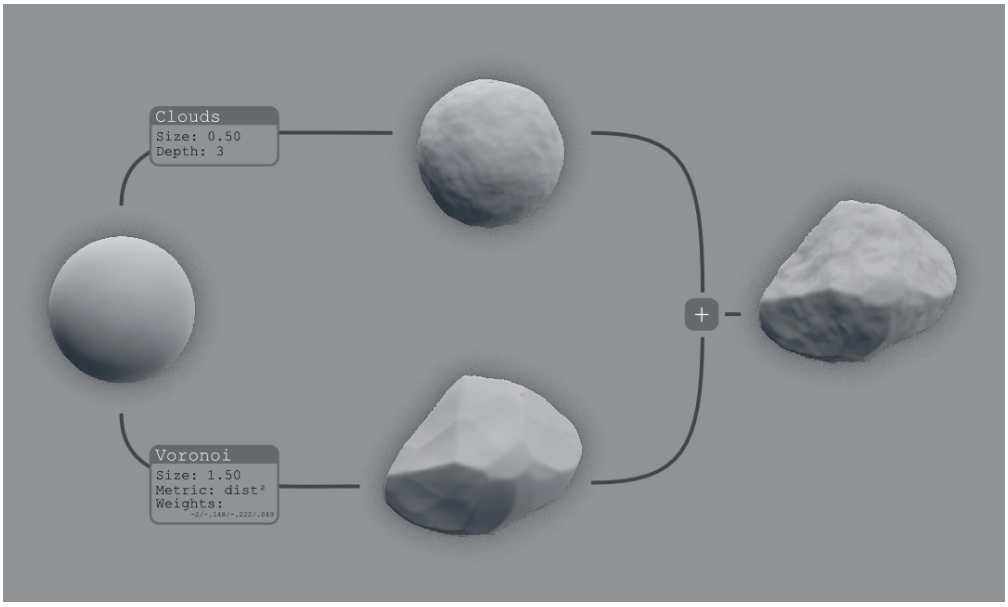


FIGURE 3.18 – Processus de génération procédurale employé pour construire la géométrie des grains de sable. Le grain est le graphe d'une fonction sphérique constituée de la somme d'un bruit de Perlin et de cartes de distance de Voronoi aléatoires à plusieurs échelles (pour l'effet de facettes).

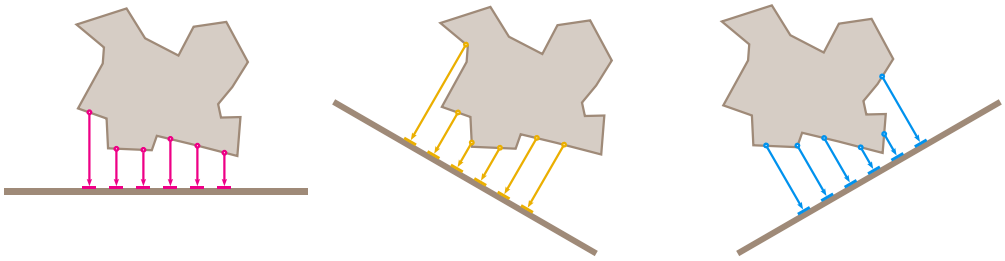


FIGURE 3.19 – Rendu préalable d'imposteur plan dans chaque direction. Lors du rendu temps réel, l'imposteur le plus proche de l'orthogonale à la direction de vue est utilisé.

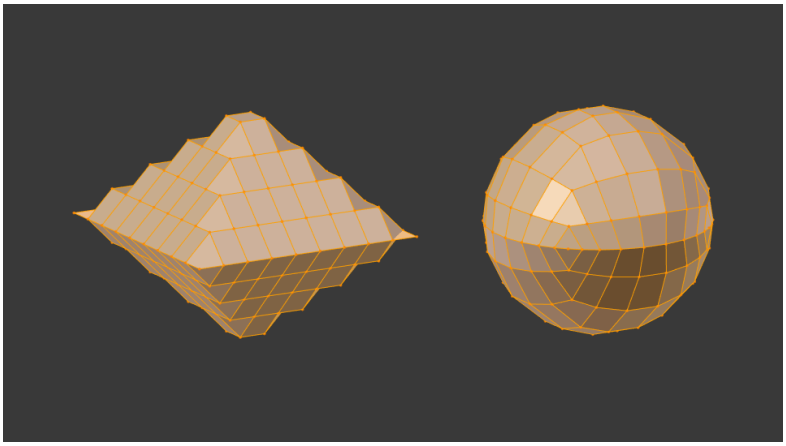


FIGURE 3.20 – Octaèdre servant de support aux rendus pré-calculés. Chaque point correspond à une position de la caméra. La grille est un découpage simple et relativement régulier de l'octahèdre de gauche, que l'on projète sur $\mathcal{S}_2$ pour la diviser. Le résultat de cette projection, à droite, met en évidence certaines irrégularités de la grille, mais elles ne sont pas critiques.

Mais, pour rendre la méthode plus générale que le rendu d'un type de grains en particulier, j'ai préféré utiliser des rendus précalculés à partir de la géométrie triangulée des grains. De cette façon, n'importe quel modèle peut être utilisé comme grain. J'ai donc construit des *imposteurs multi-vues*.

3.5.2 Imposteurs multi-vues

Imposteurs de luminance

Comme pour les billboard clouds, j'ai cherché à m'affranchir de la restriction d'angle de vue des plans imposteurs, mais cette fois sans approcher la géométrie par des plans, mais plus simplement en construisant un imposteur par cône de directions. J'ai en pratique divisé la sphère en 128 secteurs, pour lesquels j'ai calculé un imposteur (figure 3.19). Cet imposteur peut être d'assez basse définition (128 × 128 pixels) puisque les grains sont vus à une distance telle qu'ils apparaissent plus petits que cette taille (s'ils sont plus près, on utilise leur véritable géométrie).

FIGURE 3.21 – Extrait d'une scène de 20 millions de grains rendue en temps réel à l'aide d'imposteurs multi-vues. Trois précalculs différents ont été effectués pour obtenir trois couleurs de grains différentes.

Lors du rendu en temps réel des imposteurs, il est important de pouvoir efficacement déterminer laquelle des 128 vues pré-calculées est la plus adaptée à l'angle de vue. J'ai donc envisagé plusieurs divisions de la sphère. Une division statistiquement très régulière de la sphère se fait grâce aux coordonnées de Fibonacci sphériques, dont il a été trouvé une technique d'inversion [KISS15]. Mais cette inversion est tout de même trop coûteuse pour mon cas, nécessitant son application pour chaque grain. J'ai donc accepté de réduire un peu la régularité de la division au profit d'un moindre coût d'inversion. Les deux divisions qui se sont alors imposées étaient basées sur une projection cubique ou une projection octaédrale, correspondant respectivement à diviser la boule de la norme infinie ou de la norme L_1. J'ai choisi la seconde, moins régulière mais aussi encore moins coûteuse, illustrée en figure 3.20.

Afin d'assurer une transition continue entre les différents imposteurs, on considère en fait les 4 directions pré-calculées les plus proches de la direction de vue, puis on interpole l'information de ces 4 approximations [1].

Cette technique donne des résultats intéressants, comme le montre la figure 3.21. Mais cette figure met en évidence un problème conséquent : les grains ont tous la même apparence. Il y a trois couleurs de grains, car il y a eu trois précalculs. Si on voulait ne serait-ce que mille grains différents, cela nécessiterait déjà 1000^2 bit = 8 Go, ce qui dépasse les capacités de mémoire de la grande majorité des GPU grand public.

Imposteurs intermédiaires

J'ai donc repris l'idée des imposteurs enrichis que j'avais commencé à explorer lors de mes tests sur les nuages de plans. Plutôt de précalculer une luminance finale, incluant déjà tous les effets d'éclairage, je précalcule des informations intermédiaires, qui viennent s'insérer dans le processus de rendu à une étape intermédiaire. La figure 3.22 compare ce qui est rendu pour les imposteurs multi-vues simples avec ce qui est calculé dans le cas des imposteurs enrichis, ou plutôt *imposteurs intermédiaires*.

Cette approche permet une très nette amélioration : il devient possible de tourner les grains, sans voir les reflets tourner avec (voir image de couverture). Cette possibilité particularise l'aspect de chaque grain, même s'ils ont tous le même modèle (ou l'un de 3-4 modèles) à une rotation

1. au risque de perdre certaines informations de haute fréquence

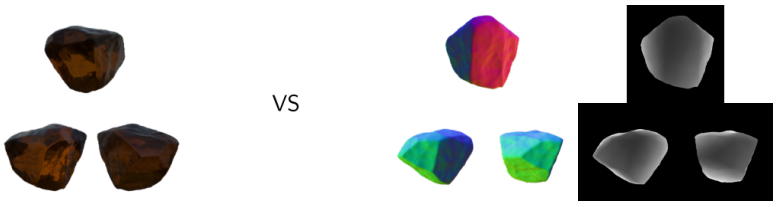


FIGURE 3.22 – À gauche, extrait des 128 vues précalculées d'un grain de sable. On remarque en particulier que le bleu du ciel de l'environnement dans lequel elles ont été calculées est intégré à ces images. À droite, cartes rendues pour les imposteurs intermédiaires, comportant un peu plus d'information (normale et profondeur) mais permettant a posteriori un changement des conditions d'éclairage.

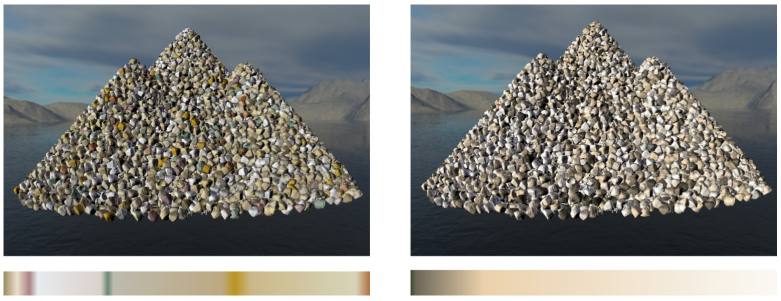


FIGURE 3.23 – Couleur paramétrique des grains rendu possible par les imposteurs intermédiaires. Le programme prend en entrée une image spécifiant la distribution de couleur attendu et une couleur est attribuée lors du rendu à chaque grain en échantillonnant dans cette texture.

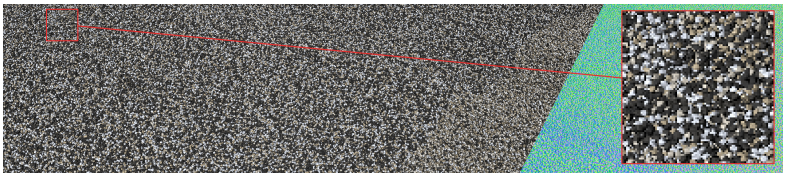


FIGURE 3.24 – Application du nuage d'imposteurs intermédiaires à la scène de 20 millions de grains.

près. Il est aussi possible de faire varier les matériaux de grains, en y appliquant n'importe quel modèle d'éclairage qui prend en paramètre la carte de normale. Par exemple, la figure 3.23 montre l'utilisation d'image pour spécifier la distribution de couleur des grains que mon programme permet.

Ce nuage d'imposteurs donne donc des résultats satisfaisants, et passe bien à l'échelle, comme en atteste la figure 3.24, mais il n'est pas en soin un modèle multi-échelle. Il est un modèle adapté à une échelle intermédiaire, et construit automatiquement à partir de la géométrie, c'est-à-dire le modèle en vue rapprochée, ce qui en assure une bonne transition. Je me suis également intéressé aux autres échelles.

3.5.3 Autres échelles

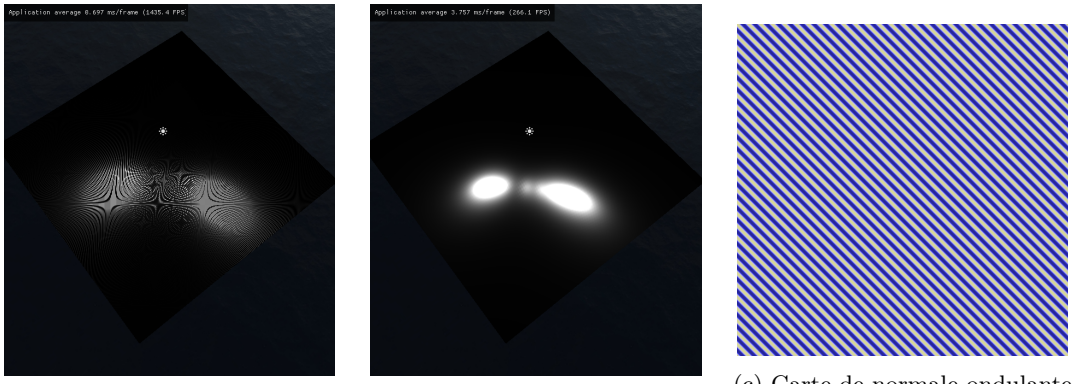
Comme je viens d'évoquer, les grains vraiment proches de la caméra sont rendus en utilisant leur véritable géométrie, ou une approximation obtenue par simplification géométrique. Étant donné qu'à une rotation près j'utilise peu de géométries de grains différentes, j'ai utilisé les primitives d'*instanciation* fournies par le GPU précisément pour ce genre de cas. C'est un point qui relève plus de la technique que de la recherche, mais c'était une occasion de plus d'affiner ma connaissance des primitives GPU.

Ce qui est plus intéressant concerne les vues plus distantes. Le nuage d'imposteur est satisfaisant tant que les grains sont plus gros que les pixels. Dès que ce n'est plus le cas, des phénomène de crénelage, similaires à ceux traités par les mip-maps, surviennent. Si plusieurs grains se projettent sur le même pixel, seul l'un d'eux sera réellement rendu. En déplaçant très légèrement le point de vue, même d'un mouvement sous-pixellique, ce peut être un autre pixel qui prend le dessus, créant ainsi des artefacts visuels très remarquables.

Approche surfacique

J'ai d'abord réfléchi à la possibilité de remplacer les zones lointaine par des surfaces enveloppant le sable, et attribuer un matériau particulier, à l'instar de ce que fait [BN12] (vu précédemment en figure 3.6). Mais contrairement à ce dernier, qui part d'une surface et y ajoute des arbres, je dois construire cette enveloppe, éventuellement dynamiquement si le sable venait à bouger et le tas se déformer.

J'ai tout de même étudié à ce titre les modèles multi-échelles de matériaux présentées précédemment. En particulier, j'ai entrepris d'implémenter le LEADR mapping et de le tester sur des exemples simples. La figure 3.25 montre un « cas jouet » sur lequel j'ai testé l'algorithme. Le fort crénelage présent avec l'utilisation de normales classiques est du même ordre que celui



(a) Normal mapping (b) LEADR mapping (c) Carte de normale ondulante

FIGURE 3.25 – Capture de mes tests d'implémentation du LEADR mapping [DHI+13]. La carte de normale utilisée est celle de l'image de droite, construite par mes soins dans le but de faire apparaître deux lobes de réflexion.

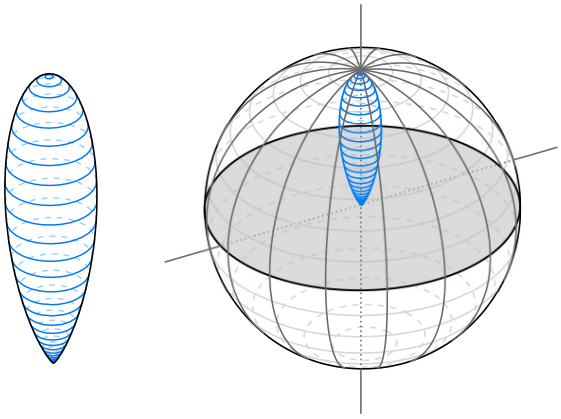


FIGURE 3.26 – Distribution de von Mises-Fisher de paramètre $\kappa = 20.0$

pour lequel les mip maps ont été développés. Mais si l'on appliquait la technique de mip map aux normales comme on le fait pour la couleur, le crénelage serait totalement perdu à grande distance et on ne verrait apparaître qu'un seul lobe de réflexion, central.

À la place, on construit des mip-maps des paramètres de la distribution de pente de la carte de normales, distribution supposée gaussienne. J'ai également généré des cartes de normales ne respectant pas cette hypothèse de distribution gaussienne des normales afin de mettre en échec le LEADR mapping à son tour. L'idée était de comparer avec [XWZB17], plus récent, qui modélise la BSDF par une somme de lobes dont le nombre est automatiquement adapté à la complexité de la distribution, un peu comme le faisait déjà [TLQ+08]. Il utilise pour lobes des distributions de von Mises-Fisher (figure 3.26), dont il est possible d'intégrer les paramètres à des mip-maps.

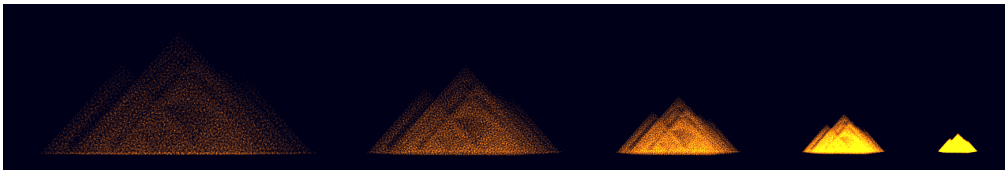


FIGURE 3.27 – Rendu par distribution de fragments. Ici, on affiche simplement le nombre de fragments présent dans chaque pixel, via la luminosité du pixel. C'est un résultat qu'il est possible d'obtenir avec le process de rendu normal, mais cette image me permettait de tester mon début d'algorithme et est une porte ouverte vers des traitements plus subtils des distributions de fragments.

Accumulation de fragments

Mais l'approche surfacique ne me convenait pas complètement. Je ne voulais pas reposer sur une enveloppe des grains, et je voulais pouvoir dynamiquement déformer le tas. J'ai donc cherché un moyen de reproduire l'idée de filtrage des différentes techniques dérivées des mip-maps mais à des ensembles désordonnés de points, et non plus aux texels régulièrement répartis d'une texture.

Pour cela, j'avais besoin d'accéder, au moment du rendu d'un pixel, non plus au grain le plus proche, avec les conséquences de crénelage que l'on a vu, mais à l'ensemble des grains se projetant sur le pixel. Je savais ce résultat possible parce que utilisé pour l'*Order Independent Transparency* (OIT) [Eve01], conséquence de l'idée d'*accumulation buffer* introduite par [Car84] et des *virtual pixel maps* [Mam89]. J'ai donc commencé à implémenter ce mécanisme, dont un aperçu du résultat est visible en figure 3.27.

C'est un mécanisme un peu plus complexe que d'autres à mettre en place car il remet en cause une partie du pipeline de rendu habituel : les fragments ne sont plus sélectionnés en ne gardant pour chaque pixel que le plus proche de la caméra. Au lieu de cela, il faut enregistrer une liste chaînée de tous les fragments présents dans chaque pixel, puis gérer soi-même la phase de composition. La construction des listes chaînées dans un environnement très parallélisé comme le GPU demande un peu de réflexion [YHGT10].

Cette direction de recherche a été entamée à la toute fin de mon stage, et est donc encore largement en chantier.

3.6 Intégration à un processus de rendu moderne

Tout au long de mes tests de modélisation, j'ai conservé à l'esprit une volonté de pouvoir interfacer correctement mes modèles avec les méthodes de rendu actuellement en usage.

3.6.1 Éclairage différé

La première de ces méthode est l'éclairage différé, ou *deferred shading*, qui est à l'origine de ce que j'ai appelé ci-avant imposteur intermédiaire. Le principe de l'éclairage différé est de faire un rendu en deux temps : un premier temps concerne la projection de la géométrie sur l'espace écran, et un second temps applique l'éclairage à cette géométrie. L'information transmise entre ces deux étapes est appelé *g-buffer* ('g' comme géométrie), et consiste en la normale,

la profondeur, et d'éventuels paramètres de matériau. L'avantage est de ne faire les calculs d'éclairage, potentiellement lourd, que pour les pixels que l'on est sûr de voir dans l'image finale. L'inconvénient est que le transfert de beaucoup de données entre deux passes de rendu pour être un frein conséquent, surtout à haute résolution, aussi le g-buffer est-il parfois compressé, ce qui peut dégrader la qualité du rendu.

Avec ce mécanisme en place, les imposteurs intermédiaires prennent tout leur sens : ce sont des imposteurs géométriques, qui sont utilisés pour rendre le g-buffer, mais pas pour rendre l'éclairage (qui n'a pas besoin de connaître la géométrie d'origine).

3.6.2 Cartes d'ombre

Un autre élément important et technique d'un moteur de rendu temps réel est la gestion des ombres dynamiques. Contrairement au rendu par path tracing, on ne peut pas en temps réel tracer un rayon par source lumineuse et par fragment et tester son intersection avec la scène entière. On fait donc l'inverse : on trace les ombres depuis les sources lumineuses. Avant chaque trame rendue depuis la caméra, on rend une carte de profondeur de la scène vue depuis les différentes lampes, et on utilise ensuite cette carte pour déterminer si un fragment donné est à l'ombre ou non. Ces cartes sont appelées des *shadow maps*.

Les shadow maps font l'objet de beaucoup d'efforts de recherche, car c'est une méthode fondamentalement imparfaite. Le fait de discrétiser la scène par rapport à la source lumineuse et non par rapport au point de vue peut faire apparaître des effets de crénelage très grossiers.

Les *shadow maps* étaient également utilisées en rendu hors-ligne avant que le path tracing ne s'immpose, et les premières extensions sont venues de là. Pixar a ainsi proposé les *deep shadow map* [LV00], enrichissant les cartes d'ombres pour y stocker plus qu'une seule distance, tranchant la scène telle que vue depuis la source lumineuse et enregistrant la succession d'opacités correspondant à l'intervalle entre les tranches. Cela permettait le rendu d'ombres volumétriques, par exemple de raies de lumières dans un bâtiment poussiéreux.

Cette approche est portée sur GPU par [KN01], puis amélioré par [YK08] pour donner les *opacity shadow maps*, pour lesquelles le tranchage est rendu adaptatif, changeant de forme selon la scène.

On trouve encore beaucoup de travaux concernant les shadow maps, que je n'ai pas eu le temps d'étudier : Percentage-closer soft shadows [Fer05], Convolution shadow maps [AMB+07] et Exponential shadow maps [AMS+08], Variance soft shadow mapping [DY10], etc.

Je me suis contenté de reproduire des ombres raisonnables traitant des problèmes les plus gênants comme la *shadow acne*, qui est encore un problème de crénelage/moirage. Le résultat obtenu (figure 3.28) était suffisant étant donné que ce n'était pas a priori le coeur de mon sujet, mais j'ai étudié les techniques comme les opacity shadow maps dans l'idée d'éventuellement l'utiliser pour prendre en compte la transparence partielle des grains.

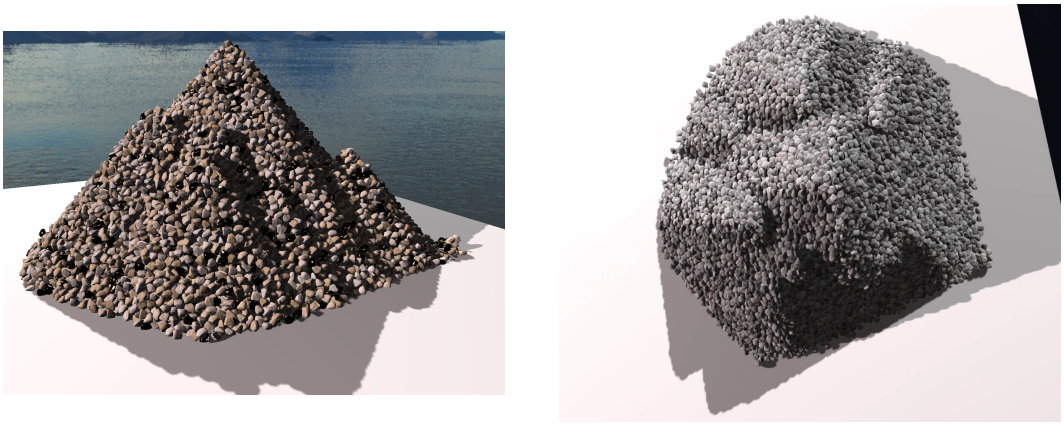


FIGURE 3.28 – Rendu d'ombres portées dynamiques (les sources peuvent tourner autour de la scène) mettant en évidence le relief du tas de sable.

FIGURE 3.29 – Tas dans lequel les grains ont été remplacé par un objet plus complexe (200k triangles) et dont le matériau est caractérisé par diverses cartes.

FIGURE 3.30 – Quelques unes des cartes caractérisant le matériau PBR utilisé pour les imposteurs de la scène de la figure 3.29. De gauche à droite : albedo, émission, métallicité, normale

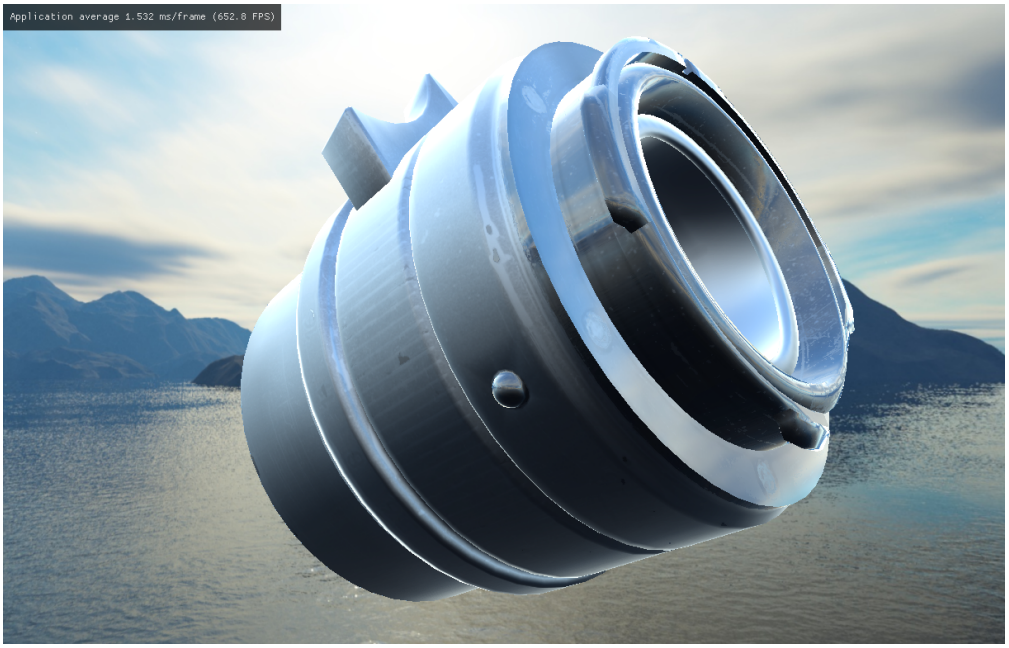


FIGURE 3.31 – Modèle d'objectif d'appareil photo de rugosité variable. La monture, à droite, est moins rugueuse que les bagues de réglage, à gauche. L'IBL permet de rendre compte de cet effet tout en utilisant la carte d'environnement comme source lumineuse.

3.6.3 Matériaux PBR

J'ai dit que les imposteurs intermédiaires pouvaient s'adapter à tout modèle de rendu fonctionnant pour de la géométrie triangulaire, et j'ai donc testé avec les matériaux dit PBR (*Physically Based Rendering*) pour rendre un tas d'autre chose que des grains (figure 3.29). Ce qui importe n'est pas tant que le modèle soit physiquement plausible que le fait que ce soit un modèle sur plusieurs textures à la fois. Il suffit dans mon cas de générer à partir des textures du modèle de robot d'origine les reprojections de ces textures par la géométrie de l'objet, dans les différentes directions, comme montré en exemple en figure 3.30.

Cette expérience était également l'occasion de constater que l'on peut rendre, dans le rôle des grains, des géométries arbitrairement complexes, puisque quel que soit le nombre de triangles dans le modèle d'origine, on n'en conserve finalement qu'un jeu de textures de 128 pixels.

3.6.4 Image-Based Lighting

Le modèle de matériau est lié à la façon dont on modélise les sources de lumière. On a vu jusqu'ici des sources de lumière ponctuelles, ou à la limite directionnelle. Ce sont celles qui permettent l'usage de shadow maps. Mais nous avons aussi vu des cas, comme en figure 3.17, où, pour émuler une réflexion, on utilise la carte d'environnement comme source.

L'*Image-Based Lighting* est une généralisation de cette pratique à tous les matériaux, en permettant l'utilisation de la carte d'environnement également pour des surfaces non parfaitement réfléchissantes. A priori, il faudrait considérer chaque point de cette carte comme une source

lumineuse, et sommer toutes les contributions. Mais ce serait beaucoup trop lourd. Au lieu de ça, lors du rendu des surfaces plus rugueuses, on accède à une version de la carte d'environnement dans laquelle la réflexion est déjà intégrée sur le cône de direction dans lequel la rugosité mène les rayons. C'est, comme pour les mip-maps, un exemple de préfiltrage pour minimiser les calculs requis au moment du rendu. Ici encore, les imposteurs intermédiaires ne posent pas de problème.

3.7 Perspectives futures

Accumulation J'ai évoqué le fait que l'utilisation d'accumulation de fragments était encore au stade de développement. C'est donc une direction naturelle vers laquelle il est possible de continuer à travailler.

AOF Une autre direction d'exploration qui a été envisagée est de s'inspirer des *Ambiant Occlusion Fields* [KL05] pour traiter des d'éclairage de chacun des grains sur ses voisins, en particulier dans le cas de réflexions fortes ou de transparence.

Hiérarchie D'autre part, j'ai dit en introduisant l'idée de nuage d'imposteur que je cherchais à réunir certains avantages des imposteurs avec d'autres des modèles plus hiérarchiques comme QSplat. Mais pour le moment, il n'y a aucun hiérarchie d'utilisée pour traiter le nuage de positions de grains. On peut réfléchir à un moyen de construire des imposteurs prenant la place non plus d'un grain donné, mais d'un groupe de grains, et ainsi éviter d'avoir des imposteurs de taille sous-pixellique.

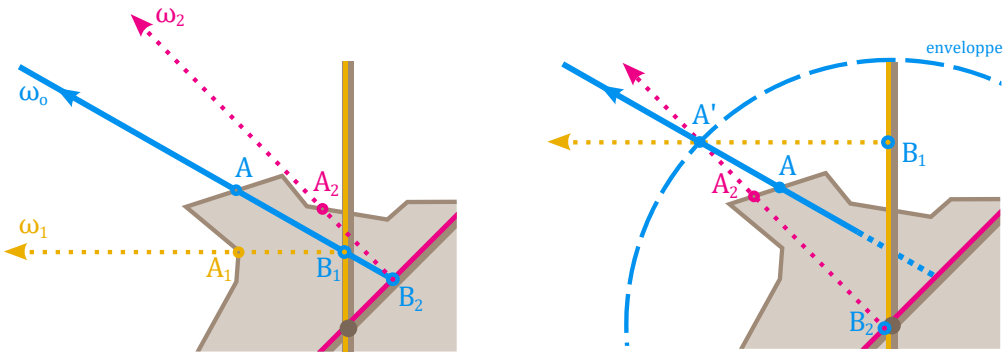
Voxels La question des voxels a été survolée, mais peu ou pas explorée en pratique. Je serais curieux d'en voir les capacités appliqués au cas du sable, mais c'est finalement techniquement une direction assez orthogonale aux autres : je n'ai aucun élément à ce stade d'un moteur de rendu de voxel.

3.7.1 Imposteurs sphériques

Enfin, la réflexion théorique sur les imposteurs multi-vues que j'ai été amené à réaliser pour la rédaction de ce rapport a éveillé quelques questions sur ce modèle. La figure 3.32 illustre en (a) le modèle tel que je l'ai mis en place, et en (b) un modèle basé sur une interprétation différente : mon ensemble d'imposteurs plans serait en fait un moyen de représenter un imposteur sphérique. Ce modèle nécessiterait par contre de spécifier un rayon à l'objet représenté.

Cette interprétation serait un moyen de se rapprocher des travaux ayant cours sur les *light fields*, traitant de champs de luminance en tout point et toute direction dans son ensemble. La donnée de mes imposteurs précalculés est en fait le light field mesuré sur la sphère englobante de l'objet à représenter. Des travaux spécifiques aux light field et imposteurs sphériques existent. [BN12] cite par exemple le travail sur ce sujet de Todt [TRSKK07], qui a fait sa thèse précisément sur les imposteurs sphériques [Tod09]. Ces travaux, découverts pendant la rédaction du rapport, me restent à étudier.

Parmis les autres affinages possibles de ces imposteurs, on peut imaginer une méthode d'affinage légèrement itérative, utilisant l'information de profondeur des imposteurs comme pour du *parallax occlusion mapping*, sur chaque plan indépendemment, pour avoir une meilleure estimation de des A_i, comme illustré en figure 3.33.



(a) Multi-plan (b) Imposteur sphérique

FIGURE 3.32 – Deux alternatives d'estimation de la radiance en A. En (a), l'estimation est faite par intersection du rayon avec les plans imposteurs les plus orthogonaux. En (b), l'intersection est estimée à celle avec une sphère. Le résultat est alors dépendant du rayon de cette sphère, de la même façon que le résultat d'un imposteur plan est dépendant de la distance de l'objet au plan.

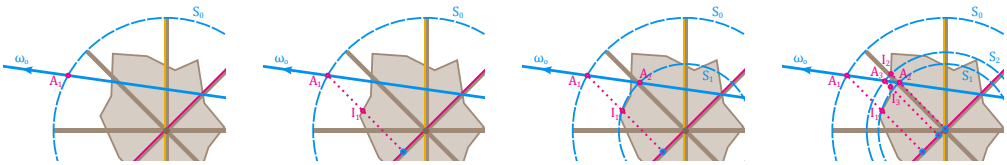


FIGURE 3.33 – Affinage de l'intersection entre le rayon ω_o et l'objet représenté par l'imposteur. En première approximation, le point d'intersection est estimé à l'intersection A_1 avec l'enveloppe sphérique de l'imposteur. Un accès à la carte de profondeur permet de trouver I_1, et ainsi une estimation du rayon de l'objet au voisinage du réel point d'intersection. On affine donc A_1 en le point A_2, intersection du rayon avec la sphère passant par I_1. On réitère ainsi. En pratique, peu d'itérations sont faites, d'autant plus que des accès aux textures dépendant eux-mêmes d'accès aux textures sont coûteux (voire interdits dans certaines versions d'OpenGL ES).

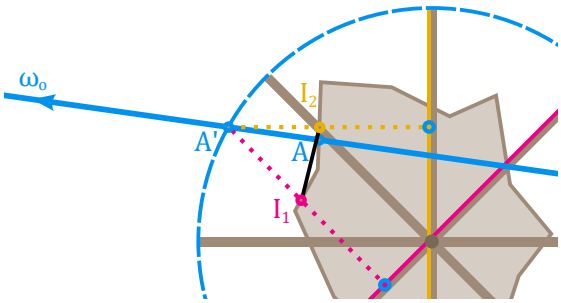


FIGURE 3.34 – Idée de méthode d'affinage du point d'intersection entre le rayon ω_o et l'objet. Les points I_1 et I_2 sont obtenu par accès aux textures des imposteurs 1 et 2, puis le point d'intersection est estimé à $\alpha_1 I_1 + \alpha_2 I$ où les α_i sont tels que $\omega_o = \alpha_1\omega_1 + \alpha_2\omega_2$.

Mais on peut également vouloir exploiter le fait que l'on affine la position du point d'intersection rayon/objet pour plusieurs imposteurs à la fois, au moins en réutilisant le point de convergence obtenu pour un plan comme point de départ du suivant. Ou même en couplant l'affinage du point d'intersection comme proposé en figure 3.34.

3.8 Animation des grains de sable

J'isole cette section, car elle ne relevait pas vraiment de mon sujet de stage.

Dans le but d'illustrer le fait que ma technique de rendu fonctionne sur un volume de sable en mouvement, déformable, j'ai mis un place une ébauche de simulation physique de grains. Je me suis fondé pour ce faire sur une méthode très utilisée en simulation de fluides : les SPH, pour *Smoothed Particles Hydrodynamics*.

Les SPH ont été introduits en informatique graphique par [DC+96], qui s'inspire de travaux de simulation en astrodynamique [Mon92]. Ils ont ensuite été adaptés à la simulation en temps réel par [MCG03]. Ce dernier ajoute également la reconstruction de surface, c'est-à-dire une conversion de modèle des particules à un maillage surfacique. Ce mécanisme pourrait servir dans le cas d'une construction d'enveloppe du tas de sable à la volée, mais je n'ai pas implémenté cet aspect.

J'ai reproduit le résultat de [MCG03] et ai commencé à regarder [AO11] qui ajoute un terme de friction, idéal pour un tas de sable. À noter que par la suite [MMCK14] généralise les SPH à toutes les simulations de déformations en temps réel (corps souples, textiles, fluides, milieux granulaires), en faisant un outil très générique, ce qui est une des raisons qui m'ont donné envie de m'y intéresser : j'avais envie d'un aperçu de ce qui me semble un ensemble de techniques actives à l'état de l'art. On trouve même des utilisation des SPH pour des mélanges multi-phase [YJL+16].

Techniquement, cette simulation m'a aussi donné une occasion de mettre en place des *compute shaders*, noyaux de calcul GPU indépendant du pipeline de rendu.

3.9 Validation

Dès les premières semaines, j'ai travaillé à la mise en place d'un moyen de comparaison entre le rendu de mes modèles et une référence rendue hors-ligne, à l'aide de Mitsuba et de Blender, deux outils libres et open source. Un exemple de rendu effectué dans Mitsuba est donné en figure 3.35.

Mon outil de rendu temps réel peut communiquer (moyennant à ce jour une petite intervention manuelle) la position de la caméra à Mitsuba afin de produire des rendus sous le même angle de vue. Les scènes pour Mitsuba et pour mon moteur sont générées, par un jeu de scripts, à partir des mêmes données. Cependant, malgré des efforts, comme présentés précédemment, pour reproduire un maximum de fonctionnalités d'un moteur de rendu, les résultats étaient trop différents pour faire des mesures objectives et chiffrées de similarité.

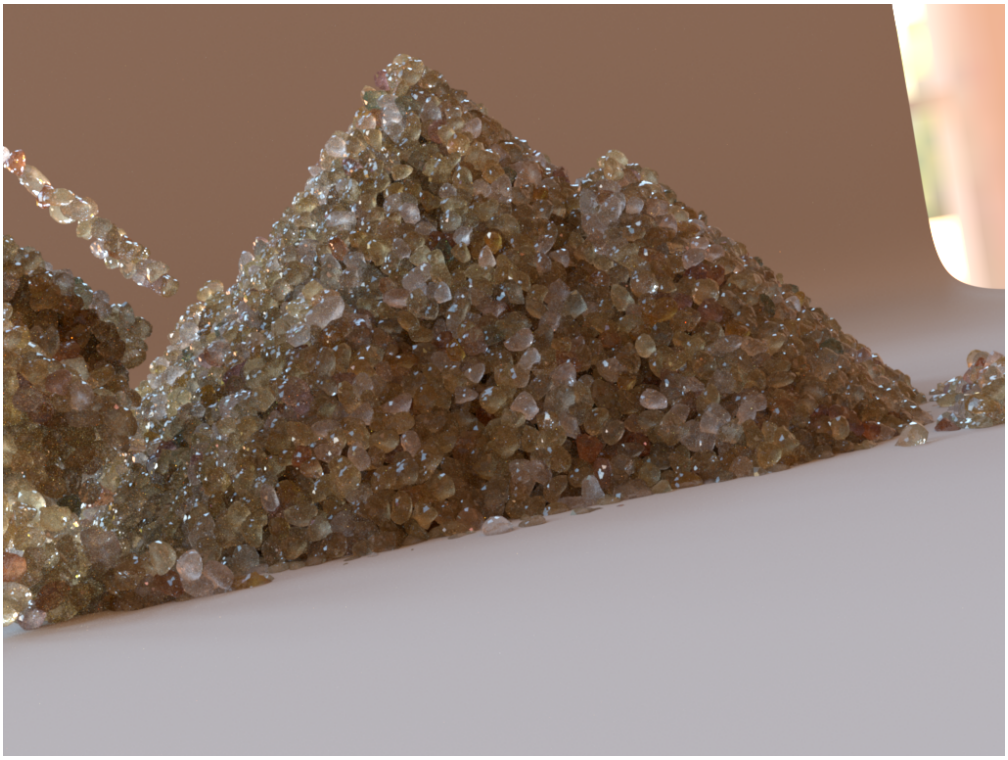


FIGURE 3.35 – Rendu du tas de sable dans un moteur de path tracing

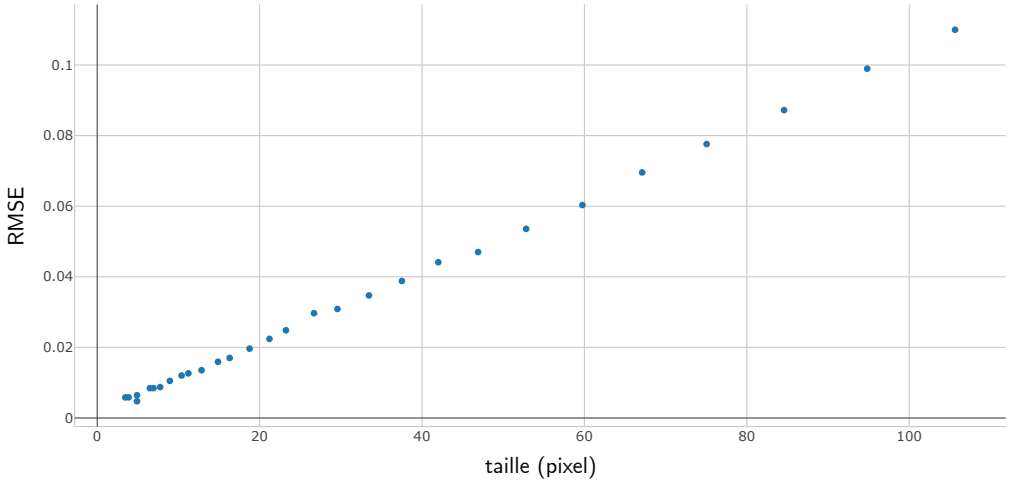


FIGURE 3.36 – Mesures d'erreur entre le rendu par imposteur et le rendu par triangles, au niveau du canal de normale du G-buffer. La taille est la racine carrée de la surface du plus petit carré contenant la projection du grain à l'écran.

Je me suis donc rabattu sur des mesures, présentées en figure 3.36, ayant lieu entre deux types de rendus différents produits tous deux par mon moteur. Cela ne permet donc pas de mesurer les erreurs dues à d'éventuelles lacunes de mon modèle d'éclairage, par exemple, mais évite le parasitage des erreurs par des différence dues à des phénomènes non implémentés dans mon moteur. C'est donc tout de même une mesure raisonnable de comparaison, tant qu'on l'applique uniquement à ce que l'on veut comparer, c'est-à-dire ici l'étape intermédiaire de rendu, au niveau du G-buffer.

La décroissance de la courbe au lorsque l'objet est plus lointaint correspond à la cible du modèle d'approximation. En outre, cette courbe permet de choisir, en fonction du degré de précision recherché, ou placer le seuil à partir duquel utiliser la véritable géométrie plutôt que les imposteurs.

Chapitre 4

Conclusion

L'étude et la recherche de modèles multi-échelles et hybride a été une occasion de s'intéresser un peu à tous les aspects du rendu, à chercher des modèles moins communs.

Cet objectif est un moyen de rapprocher des éléments de modélisation habituellement laissés disjoints, car la spécialité de personnes distinctes. Par exemple, les modèles hybrides couplent géométrie et matériau. Les modèles multi-échelles couplent le modèle de l'objet et celui du point de vue, de la caméra. Ce type de couplage se retrouve aussi par ailleurs dans l'image-based-lighting, qui lie la modélisation des matériaux à celle des sources de lumière.

Sur le plan technique aussi, ce stage a été l'occasion pour moi d'apprendre beaucoup. J'y ai acquis une bien meilleure compréhension de connaissance des composantes d'un moteur de rendu temps-réel moderne, de la modélisation au rendu, ainsi que sur les contraintes imposée par le matériel et avec lesquelles il est possible de jouer. Je n'ai d'ailleurs pas eu le temps de détailler beaucoup ces dernières, mais je recommande à cet effet [SWJH13]. Je ne m'attendais pas à rédiger un rapport si long, mais j'ai été grisé par la quantité de détails qui m'étaient inconnus auparavant et pourtant si déterminants.

Je conclue ce rapport par de chaleureux remerciements envers toute l'équipe qui m'a accueillie, avec laquelle j'ai passé des mois forts agréables et avec laquelle je reste en contact.

Bibliographie

[AMB+07] Thomas Annen, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. Convolution shadow maps. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, pages 51–60. Eurographics Association, 2007.

[AMS+08] Thomas Annen, Tom Mertens, Hans-Peter Seidel, Eddy Flerackers, and Jan Kautz. Exponential shadow maps. In *Proceedings of graphics interface 2008*, pages 155–161. Canadian Information Processing Society, 2008.

[AO11] Iván Alduán and Miguel A Otaduy. Sph granular flow with friction and cohesion. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics symposium on computer animation*, pages 25–32. ACM, 2011.

[BB17] Laurent Belcour and Pascal Barla. A practical extension to microfacet theory for the modeling of varying iridescence. *ACM Transactions on Graphics*, 36(4) :65, 2017.

[BN12] Eric Bruneton and Fabrice Neyret. Real-time realistic rendering and lighting of forests. In *Computer Graphics Forum*, volume 31, pages 373–382. Wiley Online Library, 2012.

[BS12] Brent Burley and Walt Disney Animation Studios. Physically-based shading at disney. In *ACM SIGGRAPH*, pages 1–7, 2012.

[BvS13] Jacco Bikker and Jeroen van Schijndel. The brigade renderer : A path tracer for real-time games. *International Journal of Computer Games Technology*, 2013, 2013.

[Car84] Loren Carpenter. The a-buffer, an antialiased hidden surface method. *ACM Siggraph Computer Graphics*, 18(3) :103–108, 1984.

[CNLE09] Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, and Elmar Eisemann. Gigavoxels : Ray-guided streaming for efficient and detailed voxel rendering. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 15–22. ACM, 2009.

[CT82] Robert L Cook and Kenneth E. Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics (TOG)*, 1(1) :7–24, 1982.

[DC+96] Mathieu Desbrun, Marie-Paule Cani, et al. Smoothed particles : A new paradigm for animating highly deformable bodies. In *Proceedings of the Eurographics workshop on Computer animation and simulation*, volume 96, pages 61–76. Springer, 1996.

[DDSD02] Xavier Décoret, Frédo Durand, Francois X Sillion, and Julie Dorsey. *Billboard clouds*. PhD thesis, INRIA, 2002.

[DHI+13] Jonathan Dupuy, Eric Heitz, Jean-Claude Iehl, Pierre Poulin, Fabrice Neyret, and Victor Ostromoukhov. Linear efficient antialiased displacement and reflectance mapping. *ACM Transactions on Graphics (TOG)*, 32(6) :211, 2013.

[DHI+15] Jonathan Dupuy, Eric Heitz, Jean-Claude Iehl, Pierre Poulin, and Victor Ostromoukhov. Extracting microfacet-based brdf parameters from arbitrary materials with power iterations. In *Computer Graphics Forum*, volume 34, pages 21–30. Wiley Online Library, 2015.

[DN09] Philippe Decaudin and Fabrice Neyret. Volumetric billboards. In *Computer Graphics Forum*, volume 28, pages 2079–2089. Wiley Online Library, 2009.

[DSKA17] Dan Dolonius, Erik Sintorn, Viktor Kämpe, and Ulf Assarsson. Compressing color data for voxelized surface geometry. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, page 13. ACM, 2017.

[DWMG15] Zhao Dong, Bruce Walter, Steve Marschner, and Donald P Greenberg. Predicting appearance from measured microgeometry of metal surfaces. *ACM Transactions on Graphics (TOG)*, 35(1) :9, 2015.

[DY10] Zhao Dong and Baoguang Yang. Variance soft shadow mapping. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, page 18. ACM, 2010.

[ED08] Elmar Eisemann and Xavier Décoret. Single-pass gpu solid voxelization for real-time applications. In *Proceedings of graphics interface 2008*, pages 73–80. Canadian Information Processing Society, 2008.

[Eve01] Cass Everitt. Interactive order-independent transparency. *White paper, nVIDIA*, 2(6) :7, 2001.

[Fer05] Randima Fernando. Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Sketches*, page 35. ACM, 2005.

[FHF+17] Luca Fascione, Johannes Hanika, Marcos Fajardo, Per Christensen, Brent Burley, and Brian Green. Path tracing in production - part 1 : Production renderers. In *ACM SIGGRAPH 2017 Courses*, SIGGRAPH '17, pages 13 :1–13 :39, New York, NY, USA, 2017. ACM.

[GXZ+13] Ioannis Gkioulekas, Bei Xiao, Shuang Zhao, Edward H Adelson, Todd Zickler, and Kavita Bala. Understanding the role of phase function in translucent appearance. *ACM Transactions on Graphics (TOG)*, 32(5) :147, 2013.

[HDCD15] Eric Heitz, Jonathan Dupuy, Cyril Crassin, and Carsten Dachsbacher. The sggx microflake distribution. *ACM Transactions on Graphics (TOG)*, 34(4) :48, 2015.

[HDD+93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 19–26. ACM, 1993.

[Hop96] Hugues Hoppe. Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108. ACM, 1996.

[Hua] Ke-Sen Huang. Ke-sen huang's home page.

[JAM+10] Wenzel Jakob, Adam Arbree, Jonathan T Moon, Kavita Bala, and Steve Marschner. A radiative transfer framework for rendering materials with anisotropic structure. In *ACM Transactions on Graphics (TOG)*, volume 29, page 53. ACM, 2010.

[JZJ+15] Jorge Jimenez, Károly Zsolnai, Adrian Jarabo, Christian Freude, Thomas Auzinger, Xian-Chun Wu, Javier von der Pahlen, Michael Wimmer, and Diego Gutierrez. Separable subsurface scattering. *Computer Graphics Forum*, 2015.

[Kar13] Brian Karis. Real shading in unreal engine 4. *Proc. Physically Based Shading Theory Practice*, 2013.

[KISS15] Benjamin Keinert, Matthias Innmann, Michael Sänger, and Marc Stamminger. Spherical fibonacci mapping. *ACM Transactions on Graphics (TOG)*, 34(6) :193, 2015.

[KL05] Janne Kontkanen and Samuli Laine. Ambient occlusion fields. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 41–48. ACM, 2005.

[KN01] Tae-Yong Kim and Ulrich Neumann. Opacity shadow maps. In *Rendering Techniques 2001*, pages 177–182. Springer, 2001.

[KSA13] Viktor Kämpe, Erik Sintorn, and Ulf Assarsson. High resolution sparse voxel dags. *ACM Transactions on Graphics (TOG)*, 32(4) :101, 2013.

[Lin00] Peter Lindstrom. Out-of-core simplification of large polygonal models. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 259–262. ACM Press/Addison-Wesley Publishing Co., 2000.

[LN17] Guillaume Loubet and Fabrice Neyret. Hybrid mesh-volume lods for all-scale pre-filtering of complex 3d assets. In *Eurographics 2017*, volume 36, 2017.

[Low14] Henry Lowood. Game engines and game history. In *Kinephanos (History of Games International Conference Proceedings). http ://www. kinephanos. ca/2014/game-engines-and-game-history*, 2014.

[LV00] Tom Lokovic and Eric Veach. Deep shadow maps. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 385–392. ACM Press/Addison-Wesley Publishing Co., 2000.

[LW93] Eric P. Lafortune and Yves D. Willems. Bi-directional path tracing. In *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*, pages 145–153, Alvor, Portugal, December 1993.

[Mam89] Abraham Mammen. Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. *IEEE Computer Graphics and Applications*, 9(4) :43–55, 1989.

[MCG03] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 154–159. Eurographics Association, 2003.

[MM14] Morgan McGuire and Michael Mara. Efficient gpu screen-space ray tracing. *Journal of Computer Graphics Techniques (JCGT)*, 3(4) :73–85, 2014.

[MMCK14] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Unified particle physics for real-time applications. *ACM Transactions on Graphics (TOG)*, 33(4) :153, 2014.

[Mon92] Joe J Monaghan. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics*, 30(1) :543–574, 1992.

[MPG+16] Thomas Müller, Marios Papas, Markus Gross, Wojciech Jarosz, and Jan Novák. Efficient rendering of heterogeneous polydisperse granular media. *ACM Transactions on Graphics (TOG)*, 35(6) :168, 2016.

[MPH+15] Johannes Meng, Marios Papas, Ralf Habel, Carsten Dachsbacher, Steve Marschner, Markus H Gross, and Wojciech Jarosz. Multi-scale modeling and rendering of granular materials. *ACM Trans. Graph.*, 34(4) :49, 2015.

[MWM07] Jonathan T Moon, Bruce Walter, and Stephen R Marschner. Rendering discrete random media using precomputed scattering solutions. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, pages 231–242. Eurographics Association, 2007.

[OB10] Marc Olano and Dan Baker. Lean mapping. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pages 181–188. ACM, 2010.

[PJH16] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering : From theory to implementation*. Morgan Kaufmann, 2016.

[Qui08] Iñigo Quilez. Modeling with distance functions, 2008.

[RL00] Szymon Rusinkiewicz and Marc Levoy. Qsplat : A multiresolution point rendering system for large meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 343–352. ACM Press/Addison-Wesley Publishing Co., 2000.

[Sch94] Christophe Schlick. An inexpensive brdf model for physically-based rendering. In *Computer graphics forum*, volume 13, pages 233–246. Wiley Online Library, 1994.

[SM92] Brian E Smits and Gary W Meyer. Newton's colors : simulating interference phenomena in realistic image synthesis. In *Bouatouch K., Bouville C. (eds) Photorealism in Computer Graphics. EurographicSeminars (Tutorials and Perspectives in Computer Graphics)*, pages 185–194. Springer, 1992.

[SWJH13] Graham Sellers, Richard S Wright Jr, and Nicholas Haemel. *OpenGL SuperBible : Comprehensive Tutorial and Reference*. Addison-Wesley, 2013.

[TLQ+08] Ping Tan, Stephen Lin, Long Quan, Baining Guo, and Harry Shum. Filtering and rendering of resolution-dependent reflectance models. *IEEE Transactions on Visualization and Computer Graphics*, 14(2) :412–425, 2008.

[Tod09] Severin Sönke Todt. Real-time rendering and acquisition of spherical light fields. 2009.

[TRSKK07] Severin Todt, Christof Rezk-Salama, Andreas Kolb, and KD Kuhnert. Fast (spherical) light field rendering with per-pixel depth. Technical report, Technical report, University of Siegen, Germany, 2007. 5, 2007.

[TS67] Kenneth E Torrance and Ephraim M Sparrow. Theory for off-specular reflection from roughened surfaces. *Josa*, 57(9) :1105–1114, 1967.

[VHK+15] Ryusuke Villemin, Christophe Hery, Sonoko Konishi, Takahito Tejima, Ryusuke Villemin, and David G Yu. Art and technology at pixar, from toy story to today. In *SIGGRAPH Asia 2015 Courses*, page 5. ACM, 2015.

[WMLT07] Bruce Walter, Stephen R Marschner, Hongsong Li, and Kenneth E Torrance. Microfacet models for refraction through rough surfaces. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, pages 195–206. Eurographics Association, 2007.

[XWZB17] Chao Xu, Rui Wang, Shuang Zhao, and Hujun Bao. Real-time linear brdf mipmapping. In *Eurographics Symposium on Rendering*, 2017.

[YHGT10] Jason C Yang, Justin Hensley, Holger Grün, and Nicolas Thibieroz. Real-time concurrent linked list construction on the gpu. In *Computer Graphics Forum*, volume 29, pages 1297–1304. Wiley Online Library, 2010.

[YJL+16] Xiao Yan, Yun-Tao Jiang, Chen-Feng Li, Ralph R Martin, and Shi-Min Hu. Multiphase sph simulation for interactive fluids and solids. *ACM Transactions on Graphics (TOG)*, 35(4) :79, 2016.

[YK08] Cem Yuksel and John Keyser. Deep opacity maps. In *Computer Graphics Forum*, volume 27, pages 675–680. Wiley Online Library, 2008.

Annexe A

Relation entre intensité et luminance

L'intensité lumineuse selon une direction $\vec{\omega}$ d'un volume V est la somme de la luminance selon cette même direction sur toute la frontière de ce volume :

$$I(\vec{\omega}) = \oiint_{S(V)} L(p, \vec{n}, \vec{\omega}) \mathrm{d}S \tag{A.1}$$

où p et $\vec{n}$ sont respectivement la position et la normale à l'élément de surface d'intégration $\mathrm{d}\vec{S}$.

Or, $L(p, \vec{n}, \vec{\omega})$ peut être exprimé $L(p, \vec{\omega}, \vec{\omega})$ $(\vec{n} \cdot \vec{\omega})$, et $\mathrm{d}\vec{S} = \vec{n}\mathrm{d}S$. On a alors :

$$I(\vec{\omega}) = \oiint_{S(V)} \vec{L}(p, \vec{\omega}) \cdot \mathrm{d}\vec{S} \tag{A.2}$$

où $\vec{L}(p, \vec{\omega})$ est une notation pour $L(p, \vec{\omega}, \vec{\omega})\vec{\omega}$.

Il est alors possible d'appliquer le théorème de Green-Ostrogradski, ou *théorème de flux-divergence*, ce qui donne :

$$\begin{aligned} I(\vec{\omega}) &= \iiint_V \vec{\nabla} \cdot \vec{L}(p, \vec{\omega}) \\ &= \iiint_V \vec{\nabla} L \cdot \vec{\omega} \end{aligned} \tag{A.3}$$

où ∇L est le gradient de $L(p, \vec{\omega}, \vec{\omega})$ selon $\vec{\omega}$.

Cette dernière relation donne un lien intéressant entre l'intensité et la luminance, bien que je n'en connaisse pas d'application.