

IMAC Shading Paradize

Rapport projet openGL avancé

Présenté par : **Lauren AGOPIAN**
Régis DAMON
Quentin FASQUEL
Jonathan LEFEVRE



Université de Marne la Vallée - **Ingénieur IMAC**

INTRODUCTION

Durant notre premier semestre en troisième année d'IMAC, les élèves ayant choisi l'unité d'enseignement « programmation » ont eu à réaliser, par groupe de quatre ou cinq élèves, un projet étalé sur la durée du semestre.

Ce projet regroupait les matières « OpenGL avancé » et « algorithmie avancée », toutes deux enseignées par François de Sorbier ainsi que Vincent Nozick. La matière OpenGL avancé portait principalement sur la programmation sur GPU à l'aide des Shaders, et la matière algorithmie avancée offrait quant à elle une perspectives sur certains algorithmes très utilisés en synthèse d'image.

Le but de ce projet était donc de mettre en commun ces deux matières, ainsi que toutes les connaissances acquises dans ce domaine depuis le début de notre scolarité à l'IMAC.

Ce rapport a pour but de présenter notre projet ainsi que notre démarche. Nous verrons dans un premier temps une présentation générale du projet, puis une description plus détaillée sur la façon dont il a été réalisé.

Sommaire

1 Présentation du projet.....	4
1.1 Contexte.....	4
1.2 Sujet.....	4
1.3 Objectifs.....	5
1.3.1 Objectifs fixés.....	5
1.3.2 Objectifs atteints.....	6
1.4 Contraintes.....	7
1.5 Equipe.....	8
1.6 Organisation.....	8
2 Description du Projet.....	8
2.1 Composition graphique de la scène.....	8
2.1.1 La modélisation	9
2.2 Textures.....	12
2.2.1 Techniques générales.....	12
2.2.1.1 Color Map.....	12
2.2.1.2 Normal Map.....	13
2.2.1.3 Height Map.....	15
2.2.1.4 Gloss map.....	17
2.2.2 Techniques particulières.....	18
2.2.2.1 Le palmier.....	18
2.2.2.2 Le Ponton.....	18
2.2.2.3 Le Terrain.....	19
2.3 Shadow mapping.....	20
2.3.1 Principe.....	20
2.3.2 Implémentation.....	21
2.3.3 Débordement de texture.....	22
2.4 Soft shading.....	22
2.5 Les particules.....	22
2.5.1 Implémentation.....	23
2.6 Le son.....	23
2.6.1 Implémentation.....	23
2.7 La caméra.....	24
2.7.1 Implémentation.....	24
2.8 Création d'une surface d'eau.....	25

2.8.1 Le principe.....	25
2.8.2 Le fonctionnement du shader.....	26

1 Présentation du projet

1.1 Contexte

Ce projet s'inscrit dans le cadre de l'achèvement d'un cycle complet d'étude en matière de synthèse d'image et d'algorithme. En effet, depuis la première année de notre scolarité à l'IMAC, nous avons bénéficié d'un module algorithme/synthèse d'image par année.

Notre apprentissage s'est fait de façon progressive, si bien qu'il nous est désormais possible de réaliser une démonstration relativement complète tant sur le plan technique que visuel. Évidemment le résultat n'est pas parfait, ce que nous expliquerons par la suite, mais nous sommes tout de même satisfaits.

Ce projet n'est donc pas qu'un simple projet de fin de semestre, mais une synthèse de toute les connaissances acquises au cours de cycle de trois ans. Il marque donc une étape pour nous, et nous permet de constater l'avancée de nos connaissance depuis notre première année à l'IMAC.

Ce projet est également important car pour la première fois il nous a été possible d'obtenir un rendu dont nous sommes satisfaits sur le plan esthétique. Ceci est fondamentale, car rappelons que l'IMAC est une école pluridisciplinaire qui allie la maîtrise technique et la sensibilité artistique. Ce projet était donc parfait pour démontrer qu'un tel compromis est possible au terme d'un cursus réalisé à l'IMAC.

1.2 Sujet

Le sujet nous demandais de façon très précise de réaliser une mini-démo présentant notre savoir acquis durant la formation, ceci en temps réel.

Une liste de thèmes nous était proposée. Il s'agissait exclusivement de thèmes permettant une recherche graphique intéressante. Cette démo devait se dérouler selon un scénario préalablement établi, permettant de mettre en oeuvre les différentes étapes nécessaire à la mise en place des éléments souhaités.

1.3 Objectifs

Nous avons choisi le thème « Ile paradisiaque ». En effet, quoi de plus agréable que de se laisser aller à rêver à des vacances sous les tropiques dans un lieux enchanteur, après une succession de plusieurs mois d'études et de stages en entreprise. Ce sujet nous a donc apporté une réelle inspiration, et les idées n'ont pas manqué.

Nous avons par ailleurs été fortement impressionnés par l'une des démos fournies en guise d'exemple, intitulée « Tropics », représentant une île paradisiaque d'une grande qualité graphique.

1.3.1 Objectifs fixés

Notre premier objectif fut donc de réaliser une île vaste et complète, afin de diversifier les techniques de rendu. Nous avons donc imaginé une île découpée en plusieurs zones : plage, forêt, zone habitée.

La zone habitée devait se composer de cabanes ou de comptoirs proposant des boissons rafraichissantes. Un ponton devait également être situé sur une plage afin de débiter l'animation (l'arrivée dans l'île en quelque sorte, suivant un chemin bien précis).

Nous avons également imaginé d'autres éléments, tels que des crabes ou autres animaux marins échoués sur l'île.

Finalement une gestion du temps était prévue, avec un passage du jour à la nuit. La nuit devait se démarquer par l'allumage de lumières au niveau du ponton pointant vers le ciel, formant un chemin à travers la plage vers les habitations. Une nuée de moustique pouvait éventuellement tourner autour de l'une des lampes.

L'animation prévue devait se faire à l'aide d'une caméra progressant selon un chemin établi à l'avance (le long d'une courbe B-Spline), et permettant à l'utilisateur de la démo de visiter l'île.

Enfin, nous avons également prévu d'optimiser l'affichage à l'aide d'un Quadtree, cette technique venant d'être vue en cours. Nous pensions en effet qu'un île de taille importante nécessitait ce type d'optimisation.

1.3.2 Objectifs atteints

Malheureusement notre ambition initiale a dépassé nos réelles capacités humaines en terme d'heures et de vitesse d'apprentissage. En effet nous avons du gérer simultanément un nombre important de projets dans différentes matières, et la diversité des matières enseignées à l'IMAC nous a parfois laissé un sentiment de dispersion dans nos focalisations respectives. Il a donc été difficile de s'organiser de façon efficace, ce qui a grandement ralenti la progression de notre projet.

Nous avons cependant un certain nombre d'objectifs essentiels que nous nous étions imposés.

Ceci sera détaillé dans la suite du rapport, cependant nous pouvons déjà évoquer les différents points importants de notre rapport :

- L'obtention d'un rendu satisfaisant (image, son, déplacements...)
- Une île paradisiaque
- Un palmier, point centrale de l'île, offrant une sensation mystérieuse
- Un rendu d'océan en mouvement (avec reflets et transparence)
- Une gestion des ombres et des lumières
- L'utilisation d'un grand nombre de Shaders vus cette année
- L'utilisation d'algorithmes vus en cours (heightmap, moteur de particules, courbes B-spline en trois dimensions)

Notre île est donc moins vaste et moins riche que prévu, cependant nous avons réussi à la terminer et le résultat nous plaît d'un point de vue graphique malgré sa simplicité. Certains ont même trouvé que la simplicité du lieu renforçait son esprit paradisiaque, et qu'une surcharge d'éléments aurait brisé le charme de l'île.

La réduction de la taille de l'île nous a incité à remettre en question l'utilisation d'un Quadtree. En effet, ne constatant pas de problème de ralentissement, nous avons jugé préférable de ne pas utiliser le Quadtree et de focaliser nos efforts sur d'autres points du projet.

1.4 Contraintes

Les contraintes étaient principalement énoncées dans le sujet.

Il nous était tout d'abord imposé de conserver une cohérence temporelle et spatiale dans notre scénario. C'est la raison pour laquelle nous avons choisi une île, qui symbolise un espace délimité par excellence. La cohérence temporelle va de soi dans notre sujet vu la petite taille de notre scène.

Les contraintes de langage étaient l'utilisation du C++ et de l'API OpenGL, ainsi que l'ajout de bibliothèques libres tant que celles-ci ne nécessitaient pas d'installation.

La démo devaient durer quatre minutes, et tourner sur les machines mises à dispositions par l'école. Notre application devait également pouvoir tourner en boucle, et être arrêté à tout moment par un bouton du clavier.

Cependant certains membres de l'équipe ont du faire face à une contrainte supplémentaire. L'impossibilité d'utiliser les Shaders sur leur machine personnelle les a contraint à travailler certaines parties du projet sur les machines fournies par l'école, les empêchant d'avancer aussi vite qu'ils l'auraient souhaité.

1.5 Equipe

Notre équipe est composée des membres suivants :

Lauren Agopian

Régis Damon

Quentin Fasquel

Jonathan Lefèvre

Les rôles ont été répartis en fonction des affinités de chacun (travaux sur les Shaders, sur la structure globale du projet, sur les algorithmes...)

1.6 Organisation

Notre équipe a opté pour l'utilisation d'un gestionnaire de versions, à savoir Subversion (svn). Ceci nous permettait de mettre à jour régulièrement le projet, à tout moment et de n'importe où. Ainsi le travail « à distance » était possible, et chacun pouvait se tenir informé à tout moment des avancées sur le projet.

Nous avons également opté pour une séance hebdomadaire de travail, les jeudi dans la salle du CRT. Ces séances débutaient par une réunion d'avancement du projet, les objectifs atteints et les objectifs restant, ainsi qu'éventuellement une nouvelle répartition des tâches en fonction des besoins urgents.

Ces séances se poursuivaient par le développement du projet, où la proximité des autres membres de l'équipe rendaient le travail plus efficace.

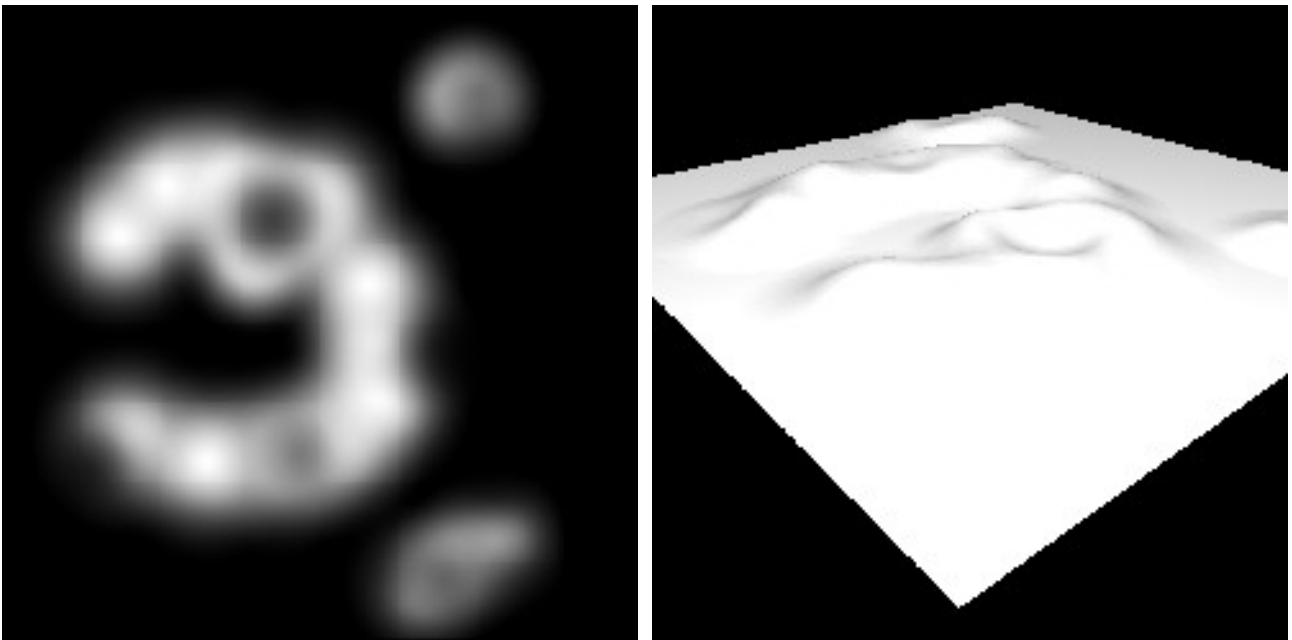
2 Description du Projet

2.1 Composition graphique de la scène

C'est par une première phase de brainstorming autour du thème de l'île paradisiaque qu'a pu commencer la conception de notre scène. Notre imaginaire et les photos de rêves que nous avons pu visionner, nous ont conforté dans l'idée d'évoquer le dépaysement et la beauté de la nature par la découverte d'une île de sable fin, et d'une végétation tropicale au beau milieu d'une large étendue d'eau.

La composition graphique de notre scène est relativement simple, nous voulions privilégier l'esthétique de l'île avant de complexifier la scène. Par conséquent notre scène dispose :

- d'un terrain construit à l'aide d'une carte de hauteur codée en niveau de gris. Une correspondance entre les pixels de l'image et les coordonnées du terrain permet d'en définir la hauteur au vertex correspondant.

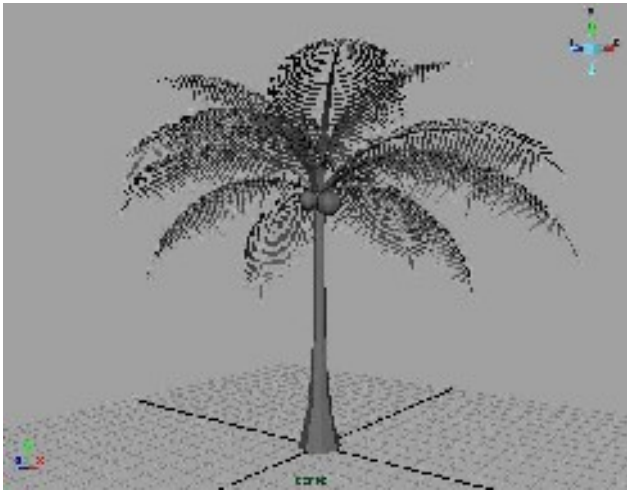


- d'un plan horizontal pour la surface de l'eau
- d'un palmier pour mettre en avant les reflets de l'eau et l'ombrage
- d'un ponton en bois pour y appliquer les effets du mapping et texturing
- d'une cube map représentant le ciel, afin de délimiter le champ de vision de la scène

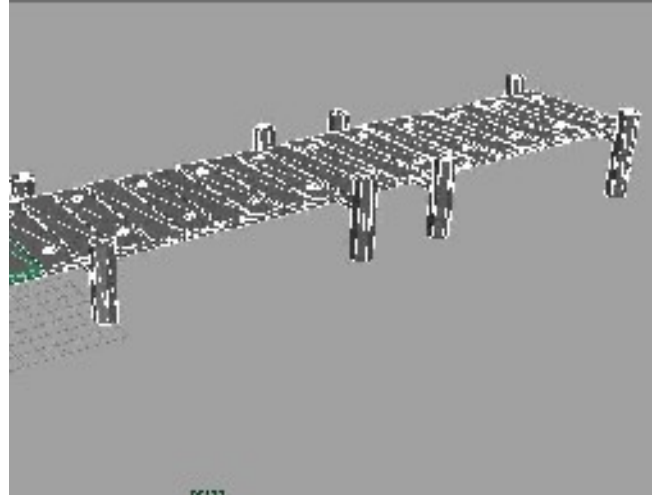
2.1.1 La modélisation

Afin d'optimiser la démonstration pour le temps réel, les objets ont été modélisés avec Maya avec un minimum de polygones pour un bon compromis temps-reel, esthétique. Les détails des objets n'est donc pas donnée par sa géométrie propre, mais par une texture de normale qui lui sera associée.

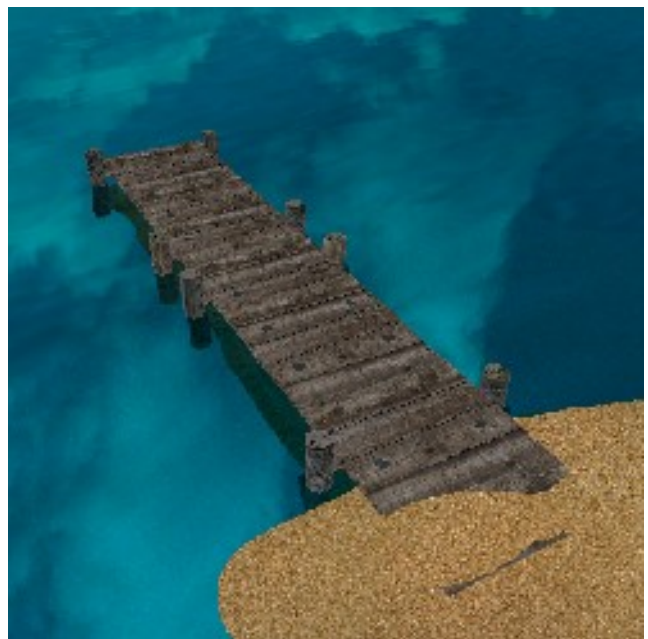
Nous utilisons le format OBJ, un standard dans la modélisation 3D, pour l'intégration des objets dans notre scène. Ce format stocke uniquement les informations sur la géométrie de l'objet ce qui nous à permis de mettre facilement en place un chargeur pour nos modèles.



Palmier

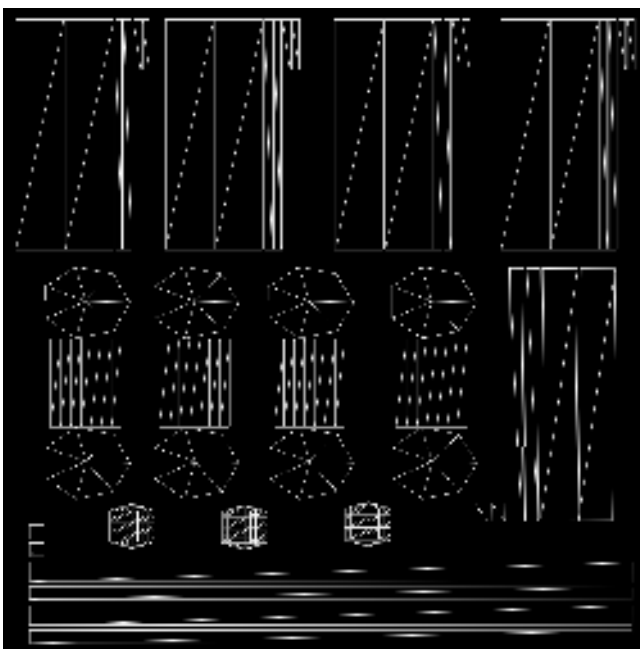


Ponton



Notre chargeur prend en compte les informations sur la position, la normale et les coordonnées de textures de chaque vertex du modèle. Ces coordonnées sont ensuite associées aux vertex du modèle à l'aide d'une carte appelée UV map pour indiquer l'emplacement des pixels d'une texture qui sera projeté sur le modèle 3D. Le plaquage d'un carte UV (mapping) est une méthode de plaquage de texture qui consiste à déplier un modèle en trois dimension en une sorte de carte plane, sur laquelle on placera une image qui servira de texture. Cette image est déformée, suivant les lignes de découpe du dépliage de la carte que nous avons spécifiés dans le logiciel de modélisation (Maya), pour être appliquée sur l'objet.

Les texture peuvent ensuite être réalisée dans un logiciel d'édition d'image comme Photoshop, à l'aide des lignes de repères de cette carte UV.



Carte UV du ponton



Texture associée

2.2 Textures

Les textures sont un point essentiel de notre projet. Alors que nous avons pris le parti de simplifier les modèles 3D, nous avons souhaité améliorer le rendu par des textures travaillées et des effets de Shader. Aussi, nous avons utilisé différentes manières de texturer selon le type d'objet.

2.2.1 Techniques générales

2.2.1.1 Color Map

Lors de l'export des modèles, depuis Maya, dans un fichier OBJ, sont stockées les coordonnées de textures. Un fichier image représentant le gabarit de la texture est aussi généré. Ce gabarit est ensuite rempli sous un logiciel de traitement d'image, ici The Gimp.

Pour exemple, la texture du palmier est un fichier .ppm contenant à la fois la couleur du tronc, des noix de coco, et des feuilles.

Principe

Pour chaque vertex de chaque polygone, la coordonnée de texture associée va pointer sur la zone de couleur correspondant à la partie actuelle du modèle dont le polygone fait parti.

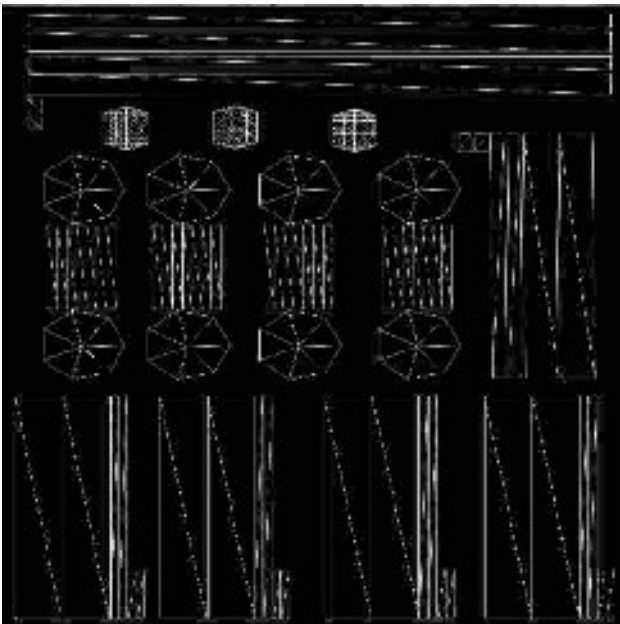


Illustration 1: Gabarit du Ponton



Illustration 2: Color Map du Ponton

2.2.1.2 Normal Map

A partir de cette texture de couleur est générée une Normal Map qui sera utilisée pour l'effet de Bump Mapping. Cet effet renforce l'impression de dénivelé d'une texture en agissant sur la modèle d'illumination. Cette carte est générée grâce au Plugin Normal Map de Gimp, l'équivalent du Plugin Nvidia pour Photoshop¹.

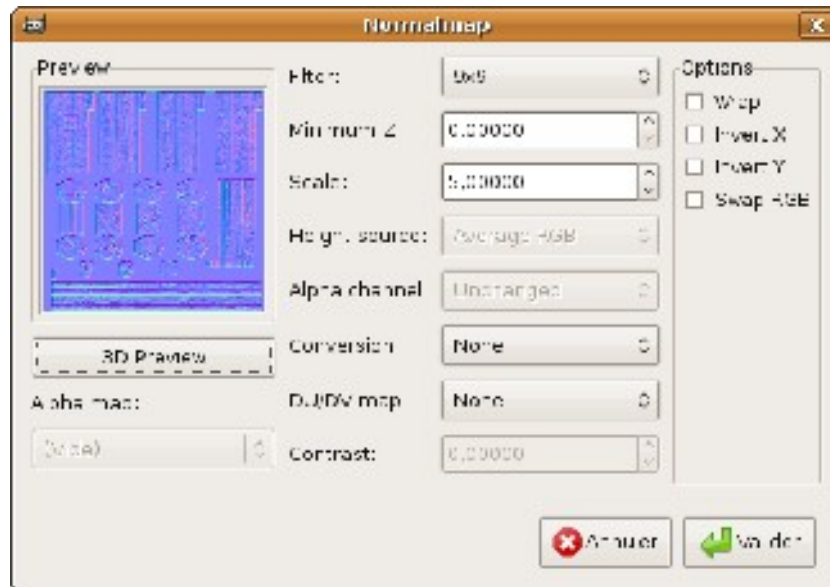


Illustration 3: Plugin: Normal Map - Settings

¹http://developer.nvidia.com/object/nv_texture_tools.html

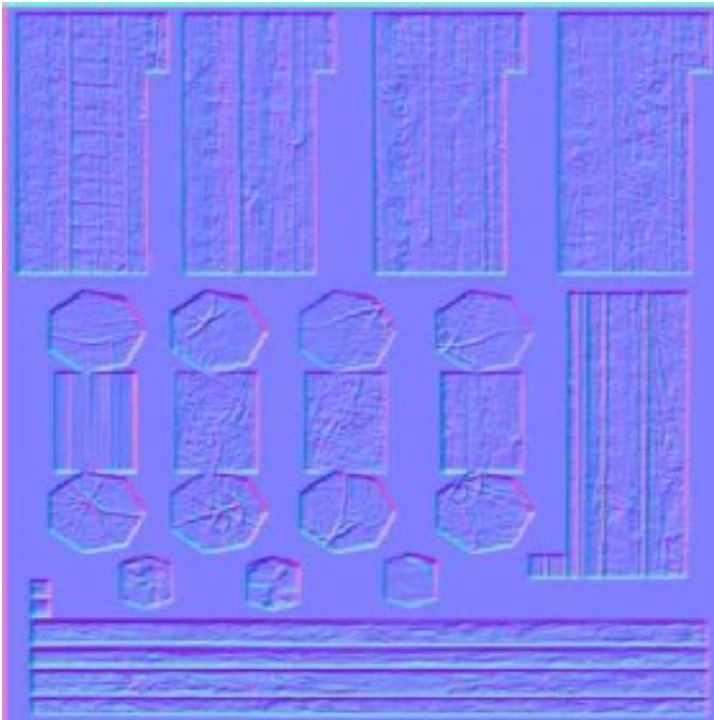


Illustration 4: Normal Map du Ponton

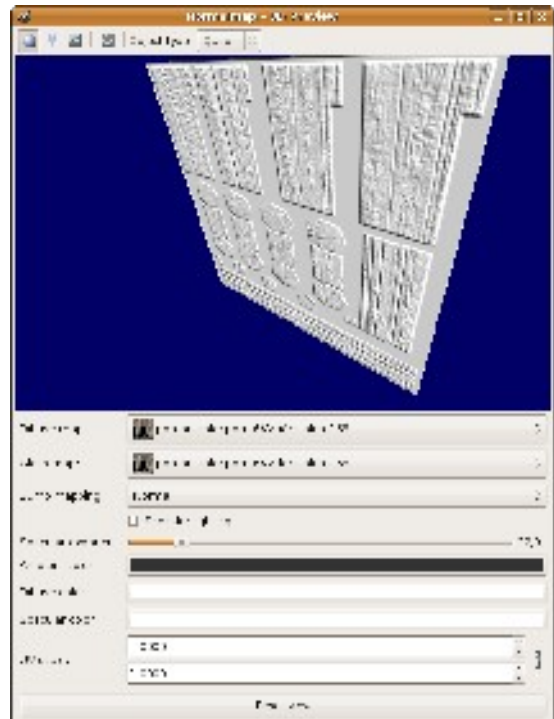


Illustration 5: Plugin : Normal Map - 3D preview

Principe

A chaque pixel de la texture est associé une normale de réflexion. Les 3 composantes de ce vecteur normal correspondent aux trois composantes colorimétrie de la Normal Map.

La Shader d'illumination appliqué à un Texel de la Color Map, va calculer l'illumination en fonction de ce vecteur normal.

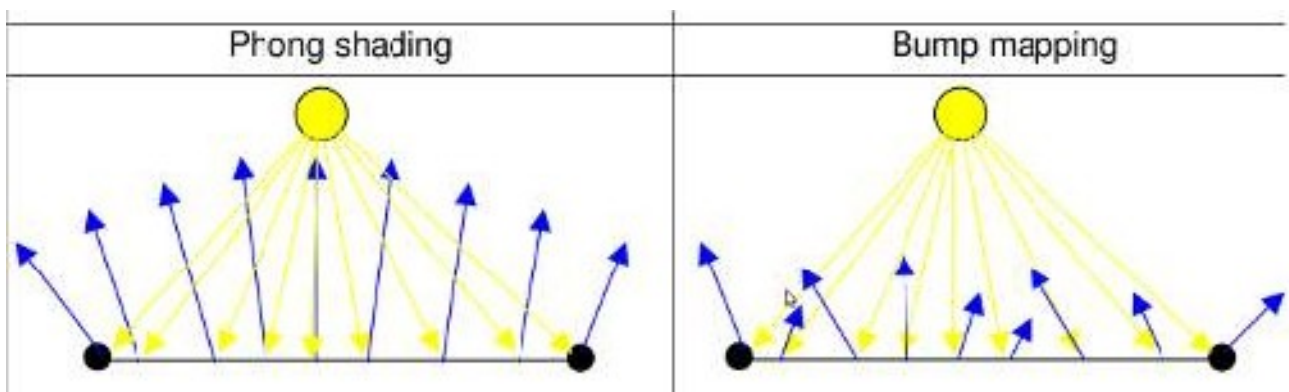


Illustration 6: Bump Mapping : Principe

Problème

Les normales de chaque Texel sont exprimées dans l'espace de texture. Cette espace est décrit par la base orthonormale TBN pour Tangente Binormal et Normal. Afin de calculer l'illumination des pixels d'un triangle, il faut exprimer les différents vecteurs impliqués dans le calcul (vecteurs directeurs de la caméra et de la lumière) dans cette espace vectorielle.

Pour un modèle 3D complexe, chaque triangle a son propre espace TBN.

Il est défini par

- N la normale naturelle du triangle : un vecteur normal au plan défini par les 3 points du triangle.
- T la Tangente au triangle : vecteur colinéaire à une droite passant par l'une des arêtes
- B la Binormale : vecteur normal à N et T : $B = \text{cross}(N, T)$

Lorsque l'on applique un Shader de Bump Map sur un modèle, il faut lui fournir, pour chaque triangle, la matrice de transformation du repère globale des Objets au repère locale, TBN, du triangle considéré.

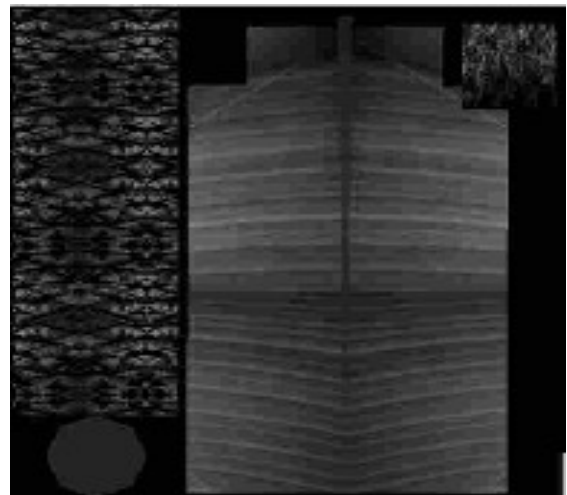
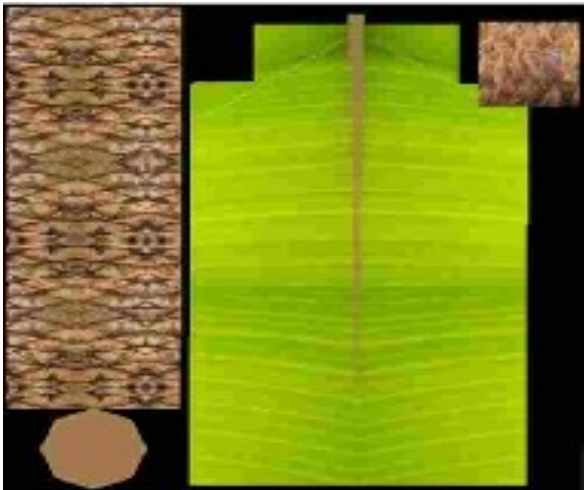
Solution

Etant donné que tous nos modèles 3D sont statiques, nous calculons, lors du chargement de l'OBJ le repère TBN correspondant à chacun de ces triangles. Calculer une fois pour toute l'application, celle-ci gagne en performance.

Dans le cas où l'objet serait mouvant, le repère TBN associé à chacun de ses triangles n'est pas affecté lors d'une translation et d'un opérateur d'échelle. Ce repère devra seulement être recalculé lors d'une rotation.

2.2.1.3 Height Map

Le Parallax Mapping nécessite une carte de hauteur, afin de simuler le dénivelé d'une texture en se basant sur l'angle de vue de la caméra. Cette effet est complémentaire du Bump Mapping : alors que le Bump agit sur la lumière pour émettre une impression de profondeur, le Parallax se base sur l'angle de vue de la caméra. Cette carte est générée en traduisant la Color Map en niveau de gris. Afin de renforcer l'effet, un réglage de la luminosité et du contraste de l'image est nécessaire.



Principe

La Height Map indique, en tous Texel de la texture de couleur, la hauteur supposé de celui-ci.

En fonction de la position de la caméra, cette carte nous indique quel Texel de la carte de couleur le programme est censé afficher.

Soit Eye le vecteur incident de la caméra sur le pixel du polygone courant. Si ce vecteur est normal au plan défini par la polygone, l'effet est nul.

Sinon, le pixel courant adoptera la couleur du plus proche texel de couleur, déterminé par la hauteur de celui-ci. C'est le pixel qui rencontre en premier la demi droite du regard qui sera afficher.

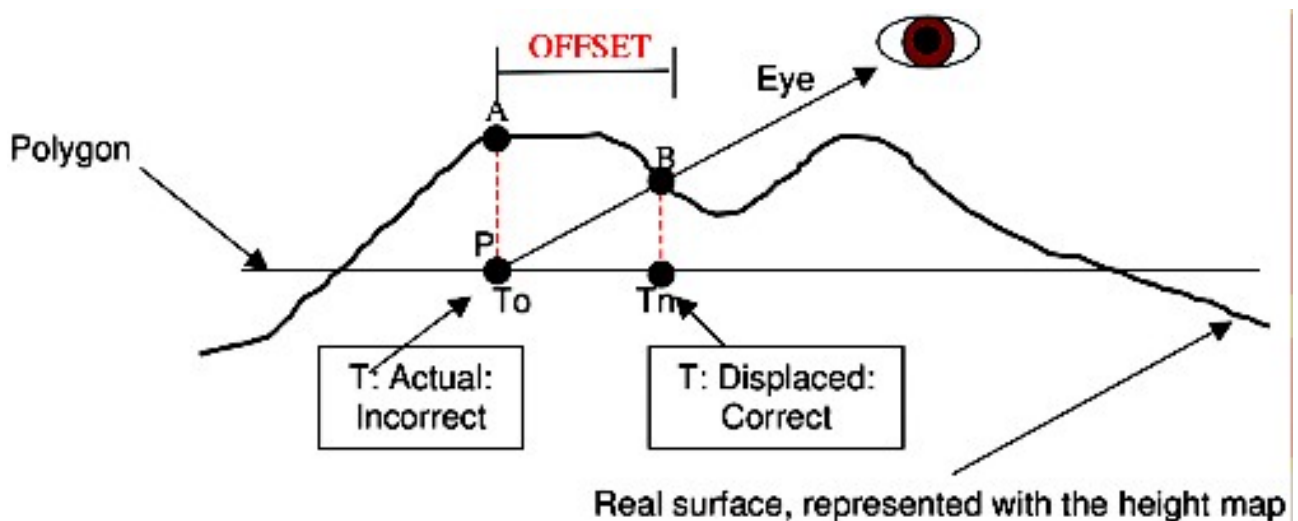


Illustration 9: Principe du Parralax Mapping ou Pixel Displacement

Problème

Le Parralax Mapping possède, dans notre cas, un certain inconvénient. En effet, étant donnée qu'un seul fichier image est utilisé pour texturer l'ensemble des parties d'un objet, il se peut que le décalage des coordonnées de textures sortent de la zone de couleur pour envoyer un pixel noir.

Solution

Pour minimiser ce problème, un réglage précis du Scale & Bias est nécessaire pour chacun des modèle 3D.

2.2.1.4 Gloss map

La gloss map est une texture .ppm en niveau de gris qui indique le taux d'effet spéculaire à appliquer au texel considéré. Cette carte est générée de la même manière que la heightmap.

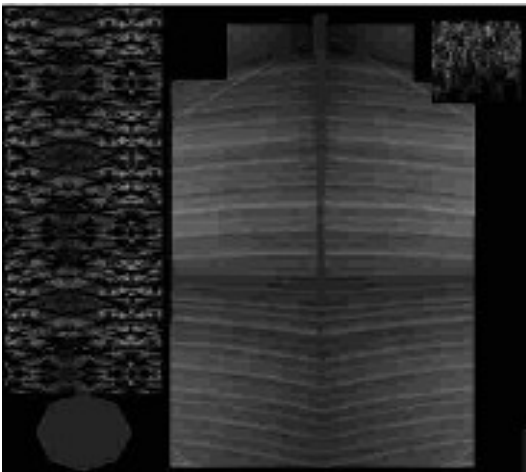


Illustration 10: Height Map du Palmier

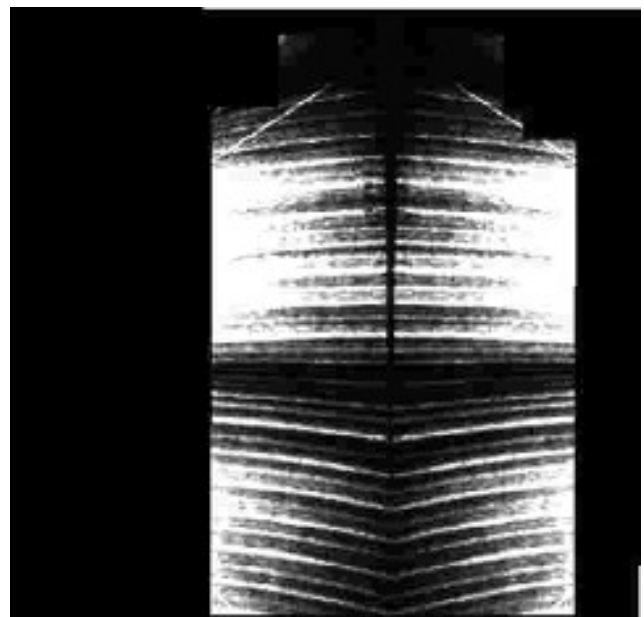


Illustration 11: Carte de Gloss du palmier, générée grâce à la Height Map

Principe

L'effet spéculaire appliqué au pixel courant sera multiplié par le facteur de Gloss (compris entre 0.à et 1.0) contenu dans la Gloss map au texel correspondant.

2.2.2 Techniques particulières

2.2.2.1 Le palmier

Le modèle 3D du palmier a été modélisé avec peu de polygone, notamment pour le tronc, afin de garantir les meilleures performances possibles.

Afin de d'ajouter un confort de vision, la technique de Bump Mapping et Parralax Mapping a été utiliser pour minimiser l'aspect plat du tronc du palmier.

Afin de renforcer l'aspect végétale des feuilles, une carte de Gloss a été appliqué à celle-ci. De cette manière, l'effet spéculaire est uniquement appliqué au feuille du palmier.

2.2.2.2 Le Ponton

Le ponton est composer de X planches et Y poteaux. Une première solution consistait à n'utiliser qu'une seule même texture pour toutes les planches et poteaux. Force et de constaté que très vite, la répétition de la même texture est assez désagréable, a supercherie trop évidente.

Pour ne pas avoir à texturer chaque planches tout en cassant cette répétition, nous avons choisit d'introduire plusieurs gabarit de planches et de poteaux.



Illustration 12: Color map du ponton contenant plusieurs planches et poteaux différents

2.2.2.3 Le Terrain

La texture du terrain devait simuler au mieux notre thème de l'île paradisiaque. Afin de conserver de la modularité dans le texturage du terrain, nous avons adopté un système de multitexturage.

Deux Color Map sont utilisé :

- Un motif de sable qui sera répété sur toutes la surface de la HeightMap
- Une carte d'effet qui sera plaquée sur la totalité de la HeightMap

Coordonnées de textures

La HeightMap possède 2 tableaux de coordonnées de texture, un pour chaque texture. Ce coordonnées sont générées via la fonction `computeTexturesCoordinates(float nb, int tex)` qui prend en paramètre le nombre de répétition `nb` que la texture d'identifiant `tex` effectuera sur toutes l'étendu de la HeightMap.

Motif de sable

Ce motif sera répété sur l'étendu de la HeightMap, le motif ayant été généré de sorte que les jointures ne soient pas apparentes. Le nombre de répétitions permettra d'ajuster l'effet : plus de répétition signifier une meilleure définition mais posera des problèmes de scintillement, alors qu'une trop faible répétition laissera apparaître de gros grains de sables flous.

Une solution pourrait être d'ajuster ce facteur de répétition en fonction de la distance entre caméra et la HeightMap.

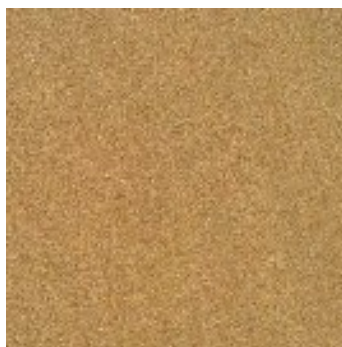


Illustration 13: Motif de sable

Carte d'effet

Cette texture globale à la HeightMap va permettre de placer des éléments pour enrichir le terrain. Il s'agit par exemple des ridules de sables sur les dunes, de coquillage sur la plage, d'étoiles de mer dans le fond marin... Cette texture va habiller le terrain.

Cette texture va aussi permettre de générer la Normal Map du terrain afin d'ajouter l'effet de Bump.



Au niveau du Shader du terrain le mélange des deux textures sera fait en fonction de la hauteur du pixel considéré. Si celui-ci est inférieur au niveau de l'eau, la carte d'effet sera affichée et le motif de sable occulté. Entre le niveau de l'eau ($y=0.0$) et une certaine hauteur ($y=2.0$) le mélange des deux textures sera fait progressivement. Au-dessus de cette hauteur ($y=2.0$), le motif de sable sera affiché entièrement et la carte d'effet occultée.

2.3 Shadow mapping

Le shadow mapping, comme son nom l'indique est une technique de rendu des ombres. Nous l'avons utilisé pour donner des ombres aux objets de la scène, voici son procédé.

2.3.1 Principe

La solution consiste à afficher deux fois la scène : d'abord du point de vue de la source de lumière pour y stocker les profondeurs des fragments (leur distance à la source), puis depuis la caméra. Lors de ce dernier affichage, il suffit alors de recalculer la position de chaque vertex dans le repère de la source de lumière et comparer leur distance à celle stockée dans la première étape. Si la distance est supérieure à la profondeur stockée, le sommet n'est pas le plus proche de la source, il est donc à l'ombre ; dans l'autre cas, le sommet est visible de la source, il est donc éclairé.



2.3.2 Implémentation

D'abord, nous effectuons un rendu des profondeurs dans un fbo (FramebufferObject), depuis le point de vue de la source de lumière. Ainsi l'on récupère une « shadow map ». Nous stockons également la matrice du point de vue de la lumière.

Lors d'un second rendu, nous appliquons un Shader pour le modèle d'illumination. Pour utiliser notre carte des ombres, nous pouvons simplement appeler la fonction texture2D et « taper » dans la texture à la coordonnée correspondante après application de la matrice de lumière. Cette fonction retourne alors la valeur de la texture, assimilable comme une profondeur, et nous n'avons plus qu'à comparer.

2.3.3 Débordement de texture

Selon le point de vue de la lumière, une partie de la scène n'est pas visible, ainsi dans la passe du Shader, nous rencontrons un « débordement de texture », qui se témoigne par une répétition de nos ombres en des endroits incohérents.

Pour pallier à ce problème, il suffit de vérifier que les coordonnées utilisées pour la « shadow map » sont bien comprises entre 0 et 1.

2.4 Soft shading

Le résultat obtenu à partir du shadow mapping n'est pas totalement satisfaisant puisque l'on obtient des ombres plus ou moins pixelisées selon que l'ombre projetée d'un objet est loin de son objet.

Une solution évidente de ce problème est d'appliquer un flou gaussien sur notre ombre.

Cependant il faut avouer que cette solution s'avère très coûteuse et réduit considérablement les performances de rendu. On pourrait imaginer une astuce consistant à n'appliquer ce calcul de flou que lorsque l'on est proche de l'ombre...

2.5 Les particules

Nous avons décidé d'agrémenter notre projet d'un ensemble de particule simple. Il devait tout d'abord servir à modéliser une nuée de moustiques, chaque moustique tournant à l'intérieur d'une sphère, et la nuée se déplaçant dans son ensemble selon une trajectoire aléatoire.

Ce système de particule a été réalisé et fonctionne, cependant nous avons décidé de ne pas l'inclure au projet, car il ne s'accordait pas avec le reste de l'île d'un point de vue graphique. Il aurait été utile dans un mode nocturne de l'île, cependant ce mode n'a pas été réalisé.

Nous avons donc cherché une autre utilisation possible pour les particules, et avons pensé qu'afin de rendre notre île d'avantage paradisiaque, il serait amusant de créer un scintillement factice à notre palmier centrale, afin d'apporter une

certaine féerie rappelant l'univers de Disney.

2.5.1 Implémentation

Le principe du système de particules a été conservé dans son ensemble, cependant un comportement différent a été affecté à chaque particule. Voici brièvement son fonctionnement.

Une classe Etoile contient deux méthodes, une pour l'affichage de l'étoile, et une pour son comportement en fonction du temps.

Chaque particule a une durée de vie qui correspond à sa valeur de transparence. Lorsque la transparence d'une particule passe en dessous d'un seuil limite, celle-ci « meurt », puis renait à une position différente, avec une taille et une orientation différente, calculés de façon aléatoire.

La vitesse de diminution de la vie est également calculée de façon aléatoire, de façon à produire un effet de scintillement.

2.6 Le son

Afin d'apporter une dimension supplémentaire à notre île, nous avons décidé d'ajouter du son à notre application.

La question que nous nous sommes posé était de savoir quel degré d'attention devait être porté sur cette partie. Il était en effet possible assez simplement de créer un son spatialisé, une gestion du volume, et d'autres fonctionnalités.

Cependant, afin de se concentrer sur le reste du projet, il a été décidé de se limiter à la simple lecture d'un mp3 au chargement de la scène. Ce mp3 est lu en boucle.

Afin de réaliser ceci, nous avons utilisé la bibliothèque FMOD, dont l'utilisation est autorisée librement tant que le projet n'a pas un but commercial.

2.6.1 Implémentation

Une classe Sound a été créée afin de lire les mp3, reprenant les fonctionnalités de base de la FMOD.

Cette classe permet de gérer les erreurs en cas de problème. Elle offre plusieurs méthodes, telles que le chargement d'un son, la lecture d'un son en boucle, la mise en pause de ce son, l'arrêt de ce son et la gestion du volume de ce son.

Il aurait été également possible de façon simple de gérer un ensemble de musique afin de permettre à l'utilisateur de changer de musique durant l'application, mais cette fonctionnalité n'a pas été retenue.

Le choix de la musique s'est fait selon un critère logique, la détente apportée par l'île. Nous avons donc choisi de diffuser un morceau du groupe « Flight of the concord », intitulé « foux da fa fa ».

Ce morceau a nécessité quelques retouches à l'aide du logiciel Pro Tools afin d'améliorer la mise en boucle.

2.7 La caméra

La caméra a été gérée de façon à pouvoir parcourir l'île selon une trajectoire déterminée à l'avance. Le mouvement de la caméra débute au lancement de l'application, et suit une courbe de type B-spline.

2.7.1 Implémentation

Une classe caméra a été créée, permettant de gérer la position du point de vue de l'utilisateur. Cette classe permet de contrôler la position de la caméra ainsi que son point de visée.

Trois mouvements de caméra sont gérés, chacun selon un angle : le « yaw », le « pitch » et le « roll », qui correspondent chacun à un axe de rotation de la caméra.

Il existe également un mode « debug » dans notre application, qui permet de contrôler la caméra à l'aide du clavier et de la souris.

Dans son fonctionnement normal, la caméra se déplace le long d'une B-spline en trois dimensions, et le point de visée de la caméra suit également une courbe de ce type. Ainsi la caméra ne regarde pas uniquement dans la direction de son déplacement.

Les B-splines sont gérées à l'aide d'une classe dédiée qui reprend un algorithme de création de B-spline.

Le principal problème rencontré fut de placé les points de contrôles de façon judicieuse afin que la caméra ait un parcours harmonieux. Ceci est rendu difficile par le fait que le type de courbes choisies ne passent pas par leurs points de contrôle. Nous pensons donc avoir commis une erreur dans le choix de nos courbes, car nous aurions du privilégier des courbes passant par leurs points de contrôle.

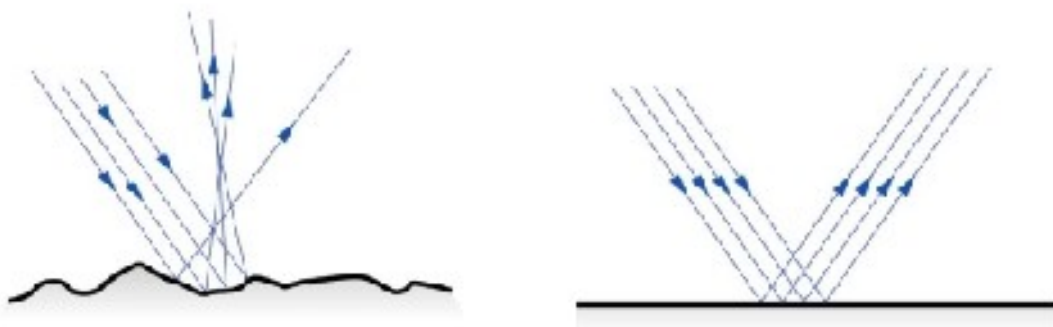
Une autre amélioration possible serait de pouvoir contrôler la vitesse de déplacement de la caméra selon sa trajectoire, afin de pouvoir se focaliser sur un panorama particulier à certains instants.

2.8 Création d'une surface d'eau

2.8.1 Le principe

Dans notre programme, la surface de l'eau est représenté par une surface plane placée à l'horizontale. Afin d'obtenir un rendu d'eau réaliste, il fut nécessaire de s'intéresser aux principaux phénomènes physiques s'appliquant à une telle surface, pour tenter de les reproduire, par l'intermédiaire d'un Shader GLSL :

- La réflexion de l'environnement sur la surface de l'eau résulte du fait qu'un rayon lumineux est réfléchi suivant un angle identique à celui sous lequel il "frappe" la surface d'un objet. Si un ensemble de rayons de même direction arrive sur une surface plane (miroir), ils seront tous réfléchis dans la même direction.



- Les déformation induite par la réfraction de la lumière proviennent du fait qu'un rayon lumineux est déviés lors de son passage d'un milieu transparent à un autre.
- L'animation des vagues permet de donner plus de vie et de réalisme à ces déformations.

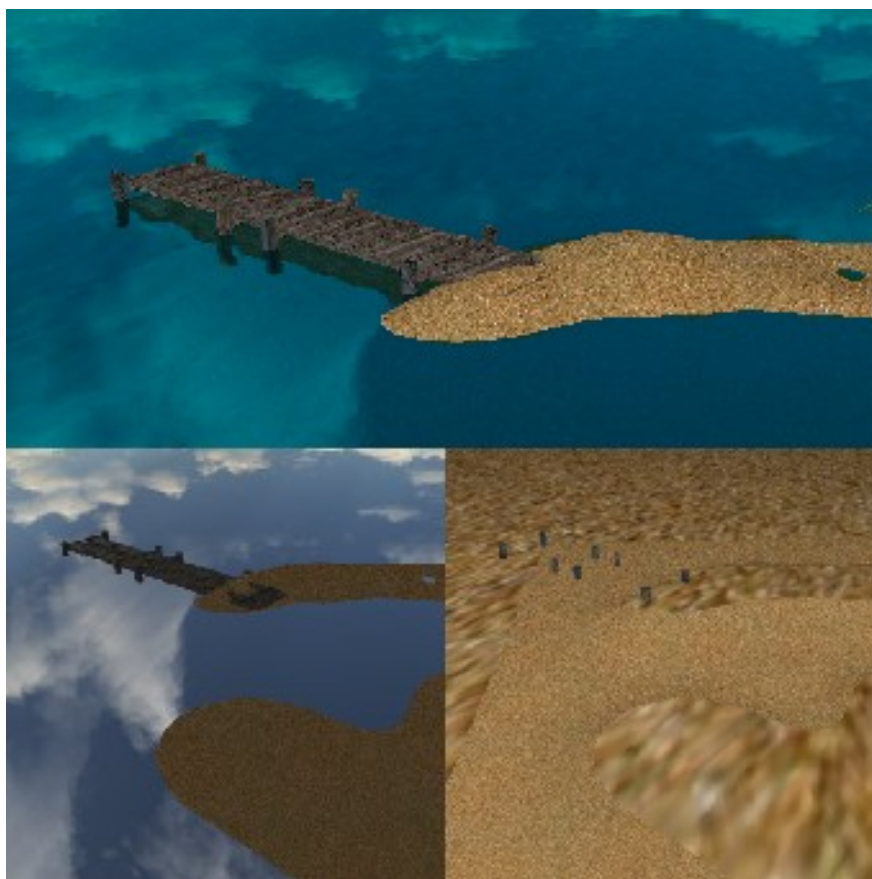
2.8.2 Le fonctionnement du Shader

Le Shader réalise toute les opération nécessaire à la mise en place de modèle simplifié pour simuler de tels phénomènes , mais il est d'abord nécessaire d'envoyer un certain nombre d'information en provenance du code openGL : textures, variables temporelles pour l'animation de l'eau, matrice de projection ... (Les information sur la géométrie de la scène ne sont pas accessible directement depuis le Shader)

Pour la mise en place de la réflexion et de la réfraction nous utilisons « un plan de clipping ». Associé à l'utilisation des Frame Buffer Object (FBO) il nous permet de réaliser un rendu chromatique d'une coupe de la scène directement dans une texture stockée en mémoire « render to texture ». Pour la réflexion « le plan de clipping » nous permet de rendre uniquement la géométrie se trouvant au dessus du plan de coupe de l'eau puis on "retourne" la caméra pour afficher la scène à l'envers. Pour la réfraction on ne conserve que la géométrie sous le plan d'eau. La résolution des texture stockée aura un incidence directe sur la qualité de l'eau au dépend de la vitesse du rendu.

Deux exemple d'utilisation des « Frame Buffer Object »

En haut	:	la scène rendu
A gauche	:	la texture de réflexion
A droite	:	la texture de refraction



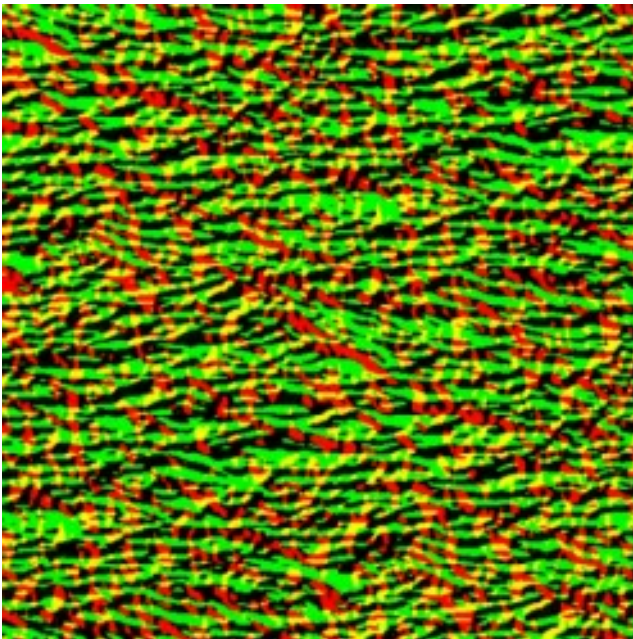
Notre Shader à également besoin de recevoir la matrice de projection qui va servir à projeter les textures de réflexion et réfraction sur le plan de l'eau. Celle ci n'est ni plus ni moins que :

Matrice de transformation * Matrice de projection * Matrice "modelview"

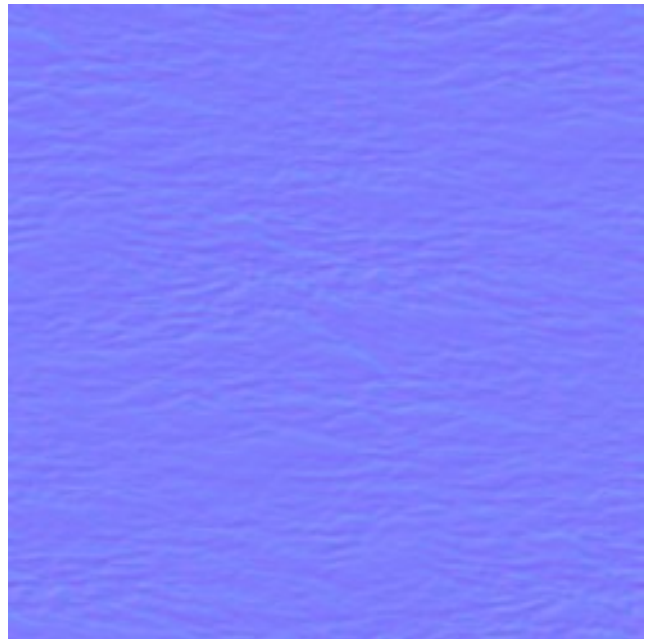
La première transformation permet de bien positionner la texture au coins supérieur gauche du plan d'eau...

Depuis le programme principale nous chargeons également :

- Une texture de normal map pour modéliser les vagues de l'eau.
- Une map du/dv pour précalculer les décalages causé par les réfractions



du/dv map



Normal map

Notre vertex Shader à deux fonction :

- Il sert à calculer la projection des coordonnées de texture à partir de la matrice de projection envoyé au Shader. Le principe de la projection de texture

consiste à modifier les coordonnées de textures afin qu'on ait l'impression que la texture de réflexion/réfraction est plaquée comme par un projecteur (ici la caméra) sur le plan de l'eau.

- On y exprime également le vecteur lumière et vecteur vision dans le même espace que le vecteur normal pour que les calculs nécessaires au bump mapping aient un sens. Dans cet espace les normales sont exprimées dans un système de coordonnées associé à chaque sommet de la surface. Les trois vecteurs formant ce repère orthonormé sont nommés tangent, binormal et normal avec:

vecteur tangent = {1.0, 0.0, 0.0} ou axe des X

vecteur binormal = {0.0, 1.0, 0.0} ou axe des Y

vecteur normal = {0.0, 0.0, 1.0} ou axe des Z.

Les transformations appliquées à notre plan d'eau sont calculées dans le fragment Shader

Pour cela on utilise les 2 textures projetées par la caméra :

- la texture de réflexion qui contient la scène affichée à l'envers en fonction d'un plan.
- la texture de réfraction qui contient la scène affichée dans le sens normal.

Ces deux textures plaquées sur une surface plane sont déformées afin de donner l'impression que l'eau forme des vagues. Pour cela on utilise la normal map qui modélise le relief de l'eau formée par les vagues en perturbant les normales à la surface. La couleur de la texture indique l'angle de perturbation de la normale au vertex correspondant.

Le Shader va donc utiliser cette texture pour "donner du relief" à la surface de l'eau, c'est à dire modifier la "luminosité" de chaque pixel de cette surface car l'intensité diffuse de la lumière atteignant la surface varie suivant l'angle formé entre la normale et le vecteur lumière. C'est ce qui nous donne cette impression

de relief... En réalité la géométrie de la surface reste totalement plane.



Pour créer l'illusion des déformations appliquée au reflet et au "fond" de l'eau nous utilisons une autre méthode qui consiste à utiliser une map du/dv (dérivée de la normal map). qui stocke directement les décalages à appliqué lorsque le fragment Shader viens récupérer la couleur du pixel à une texture donnée.



Pour plus de finesse dans la gestion des mouvement de notre eau nous avons mis en place deux coefficient: un pour régler la fréquence de répétition de la dudv map et jouer sur la fréquence de répétition des vagues (tile), et un second pour accentuer les déformation provoqué par la réfraction (k)

Pour l'animation des vagues un simple déplacement des coordonnée de texture de la normale map dans un sens et de la dudv map dans l'autre sont nécessaire à créer l'illusion des vagues. Ce déplacement est contrôlé par une variable (time) envoyé depuis le code OpenGL vers le Shader.

Une dernière étape consiste à « mixer » la part de réflexion, de réfraction, de luminosité que l'on souhaite pour donner l'aspect final du vertex.

Une fois la déformation des texture de réflexion/réfraction calculée, il est également possible d'ajouter des reflets spéculaires au résultat pour rendre les vagues plus brillantes et d'utiliser le facteur de Fresnel qui joue sur la transparence de l'eau en fonction du vecteur de vision. Ainsi au raz de l'eau seules les réflexions sont visibles alors qu'un placement au dessus de l'eau laisse transparaître le dessous de l'eau.

CONCLUSION

Ce projet a été très formateur et intéressant sous plusieurs angles.

Premièrement, les techniques apprises en TD, que ce soit en programmation avancée ou en GLSL font partie des techniques de pointe de l'infographie temps réel actuelles. Ainsi, ce projet nous a ouvert la porte des Shaders en nous suggérant toute l'étendue de leur puissance. Il est particulièrement flatteur d'implémenter des techniques de pointe qui offrent un rendu d'un niveau professionnel. Sur ce point précis, nos objectifs ont été atteints. Privilégier la simplicité de la scène nous a permis de peaufiner et a donc été payant en terme de rendu.

Ce projet nous a aussi permis d'utiliser les Shaders dans un réel projet. Cela a permis de mettre en exergue des difficultés que ne laissent pas soupçonner le TD ou autre tutoriel. En effet, l'intégration de Shader dans un projet complexe requiert un perpétuel dosage entre un code spécifique et générique. Pour contourner la rigidité qu'implique l'intégration de Shader, une bonne partie de notre réflexion s'est tournée vers le choix d'un pipeline graphique adapté à notre projet. Lors de nos recherches, nous avons utilisé plusieurs types de fonctionnement, comme par exemple un Shader par effet, puis un Shader par objet, pour au final factoriser notre code GLSL au maximum, ce qui ne nous a pas forcément pénalisés en terme de flexibilité.

Enfin, la conception de cette démonstration technique nous a sensibilisés à l'optimisation. En effet, si le GLSL permet de tirer partie de la puissance des cartes graphiques, l'optimisation du code est avant tout le garant de cette performance. Nous avons d'autant plus ressenti cette problématique avec le GLSL, avec lequel le moindre code brouillon peut entraîner une chute drastique des performances.