Compte Rendu - TP ETI5-IMI
Shaders avancés et Marching Cubes

Di Folco Maxime - Girot Charly

27/10/2017

1 Parallax Mapping - Donner l'illusion du relief

1.1 Utilisation des textures de normales

Habituellement, nous utilisons des textures *colorées* appliquées directement sur un maillage. Exemple : dessiner un mur de briques comme sur la Fig.1

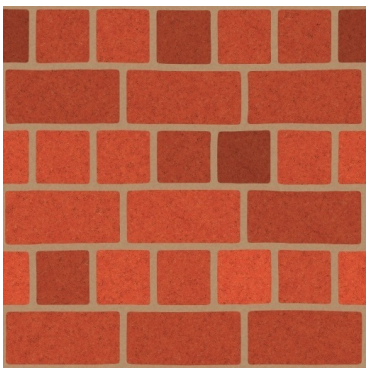
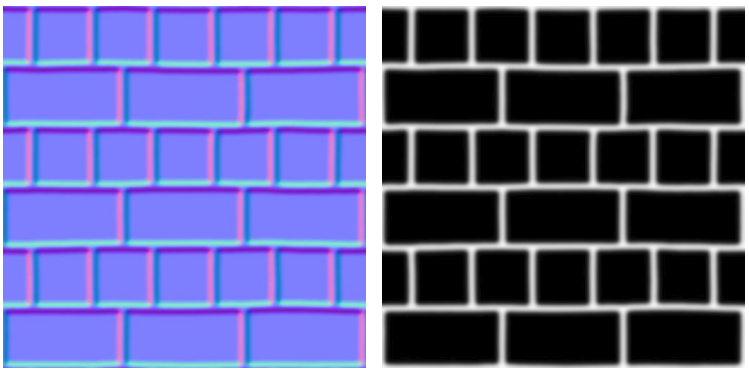


FIGURE 1 – Texture colorée de briques

Dans ce TP, nous souhaitons utiliser des textures contenant d'autres informations afin de les utiliser pour modifier des caractéristiques de nos images 3D comme les normales sur la Fig.2(a), ou pour simuler un effet de profondeur comme sur la Fig.2(b) dans le but d'avoir un rendu plus réaliste.



(a) (b)

FIGURE 2 – Textures contenant les informations de normales (a) et de profondeur (b)

Dans un premier temps, on nous fournit un programme qui implémente le normal mapping, dont le résultat est présenté Fig.3(b). Cette méthode permet de simuler graphiquement des détails géométriques sur les surfaces représentées. En effet, dans cette méthode, les normales ne sont plus seulement orientées selon l'axe principal z de la surface mais sont désormais liées aux axes x,y et z de la surface en fonction de la carte des normales représentée Fig.**??** où la composante rouge indique une orientation de la normale modifiée selon l'axe x, verte pour l'axe y et bleue pour l'axe z. Les normales ne sont ainsi plus toutes orientées dans le même sens selon la surface de l'objet, mais orientées différemment selon chaque fragment constituant notre objet comme schématisé Fig.1.1. On obtient ainsi grâce aux nouvelles orientations de normales, des surfaces contenant plus de détails et donc une impression de relief et un réalisme supérieure comparée à la Fig.3(a).

(a) (b)

FIGURE 3 – Résultat de l'application de texture couleur (a) auquel on applique la gestion des normales (b)

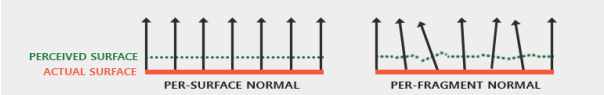


FIGURE 4 – Principe de l'orientation des normales en parallax mapping - tiré de : https ://learnopengl.com/# !Advanced-Lighting/Normal-Mapping

Lors de l'application d'un normal mapping, lorsque la carte des normales possèdent majoritairement des normales selon un axe, il faut que la normale interpolée de la surface soit de même direction. Si tel n'est pas le cas, on se retrouve alors avec un problème d'illumination comme le montre la Fig.1.1. En effet les normales ont étés calculés pour chaque fragment selon la carte des textures et non pas selon l'orientation de la normale surfacique.

Pour résoudre ce problème d'illumination selon les normales, on se place dans l'espace TBN (tangentes, bitangentes, Normales) de chaque triangle. Dans cet espace, les normales y sont calculés dans ce qui se trouve être un espace local pour chaque triangle. Les normales calculées et modifiées par la carte des hauteurs pointent alors toutes approximativement dans la direction z de chaque triangle peut importe l'orientation finale de l'objet. Il est alors possible de transformer les normales depuis l'espace local tangent vers l'orientation finale de la surface.

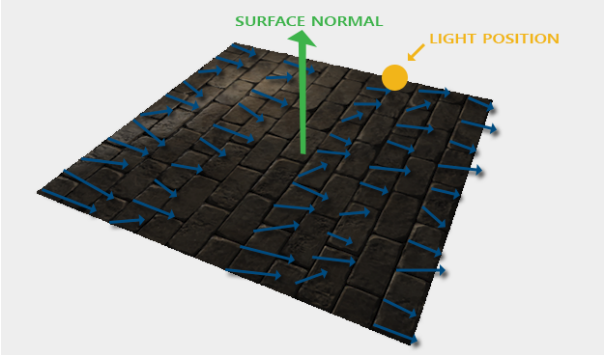


FIGURE 5 – Problème d'orientation des normales pour l'éclairage lors d'un normal mapping classique - tiré de : https ://learnopengl.com/#!Advanced-Lighting/Normal-Mapping

Pour pouvoir utiliser la méthode de normal mapping, puis ensuite la méthode de parallax que nous allons détailler par la suite, il faut utiliser de nombreuses variables pour pouvoir calculer la matrice TBN et ensuite calculer les directions de la vue et de la lumière dans nos shaders. Dans le Vertex Shader nous avons besoin des :
- sommets habituels : in vec3 position
- données des normales : in vec3 normal
- coordonnées de textures : in vec2 tex_coordsb
- tangeantes et bitangeantes calculés precedemment : in vec3 tangent ; in vec3 bitangent

On calcule alors la matrice TBN pour passer de l'espace local tangent vers l'orientation finale de la surface, avec les lignes de codes suivantes :
out vec3 vf_frag_pos = model * position ;
out vec2 vf_tex_coords ;
out vec3 $vf\_tangent\_light\_pos$ = TBN * $light\_pos$;
out vec3 $vf\_tangent\_view\_pos$ = TBN * $view\_pos$;
out vec3 $vf\_tangent\_frag\_pos$ = TBN * vf_frag_pos ;

1.2 Ajout d'un effet de profondeur avec l'utilisation des cartes de hauteur

Le normal mapping nous a permis d'obtenir un niveau de détails supplémentaire sur nos textures. Avec peu de données supplémentaires, il est possible de donner une véritable impression de relief en réalisant un Parallax Mapping. Pour cela nous allons utiliser une carte de hauteur afin de modifier par projection les coordonnées de textures à afficher. On utilisera donc dans les shaders une variable supplémentaire $height\_tex$ afin de connaître l'élévation correspondante à chaque coordonnées de textures.Cette méthode simule une impression de relief.

Supposons que nous sommes à une position $view\_pos$ et regardons vers la position $frag\_pos$ comme indiqué Fig.1.2. Le point réel observé ne devrait pas être $frag\_pos$ correspondant à la coordonnée de texture de départ mais le point C. Il faut alors calculer le point C par projection, ce qui s'avérerait trop complexe. Pour simplifier le problème on cherche alors la position du point P par calcul de l'offset entre le point A ($frag\_pos$) et la projection approximative du point B sur la surface en utilisant la hauteur au point A. Le résultat de cette méthode est présenté Fig.7(b).

Pourquoi ça marche mieux quand on a vue rasante. on applique ensuite l'algorithme suivant pour ajouter l'effet de profondeur

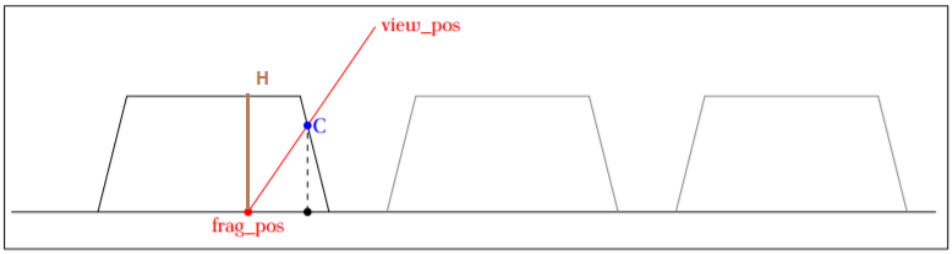
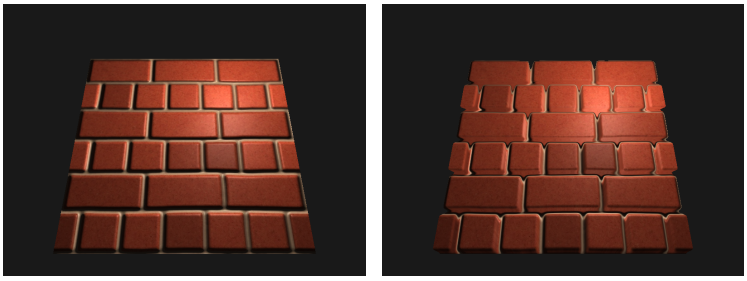


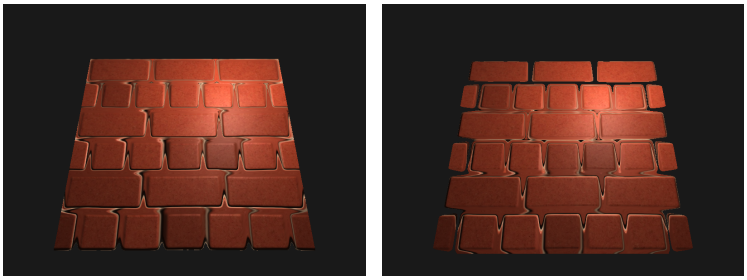
FIGURE 6 – aaa



(a) (b)

FIGURE 7 – Comparaison de la gestion des normales (a) auquel on applique une carte de hauteur (b)

Néanmoins, lors du calcul de nos hauteurs nous utilisons un facteur qui permet d'augmenter ou diminuer les hauteurs des coordonnées de nos textures. Si ce facteur est trop petit ou trop grand le rendu ne parait plus réaliste. En effet, l'offset entre les coordonnées de texture réels et les modifiées devient trop important. La surface supérieure des briques semble alors se séparer du mur, ce qui provoque une perte complète du réalisme de la scène.



(a) (b)

FIGURE 8 – Résultats de la carte des profondeurs pour un facteur de hauteur trop faible (a) trop haut (b)

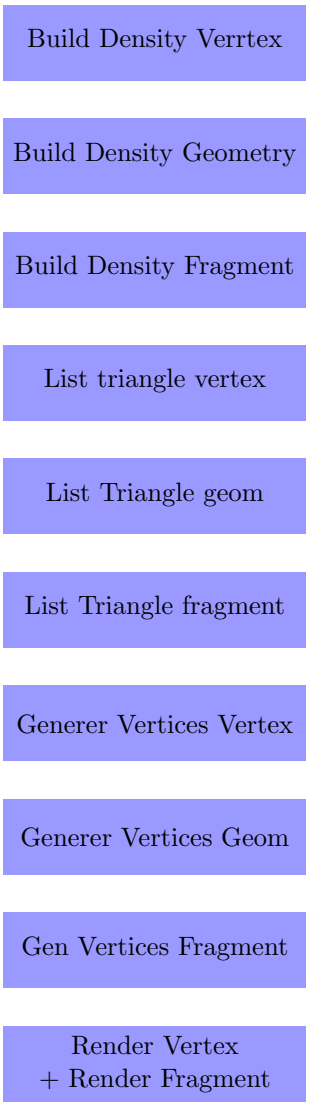
2 Marching Cubes

2.1 Présentation Générale

Une surface pourrait être décrite par une fonction densité qui à chaque point 3D associe une valeur de densité. Une valeur positive de densité correspondrait à un point de la surface solide et une valeur négative à un espace vide. Grâce au GPU, nous allons générer des "blocs" de terrain, et les subdiviser en

sous-blocs de 32x32x32 appelés voxels. Chaque sommet de ce sous-bloc possède une valeur de densité. L'algorithme de marching cubes nous permet de générer les polygones corrects dans chaque voxels en fonction de la valeur de densité de chaque coin.

Question 5 : Repérez quel groupe de shaders et quelle partie du code C++ sont liés à chaque étape. Essayez de faire un schéma indiquant comment les différentes étapes echangent des données via des FBO ou des TF.



Ordre d'utilisation des shaders - Ajouter Fleches et explications

2.2 Analyse Du code

Question 6 : Lisez la documentation de la fonction glDrawArraysInstanced , que réalise cette fonction ? Aurait-on pu s'en passer ? Quelle est l'utilisation plus classique de cette fonction ?

De la même manière que glDrawArrays permet de "dessiner/rendre" une primitive (un triangle dans notre cas), glDrawArraysInstances permet de synthétiser une série instanciée de primitives. On aurait donc pu s'en passer en utilisant glDrawArrays dans une boucle avec itération des indices de chaque primitive (ce qui est fait automatiquement avec cette fonction).

Question 7 : Comment s'utilise la fonction glActiveTexture ?

2.3 Amélioration de l'algorithme

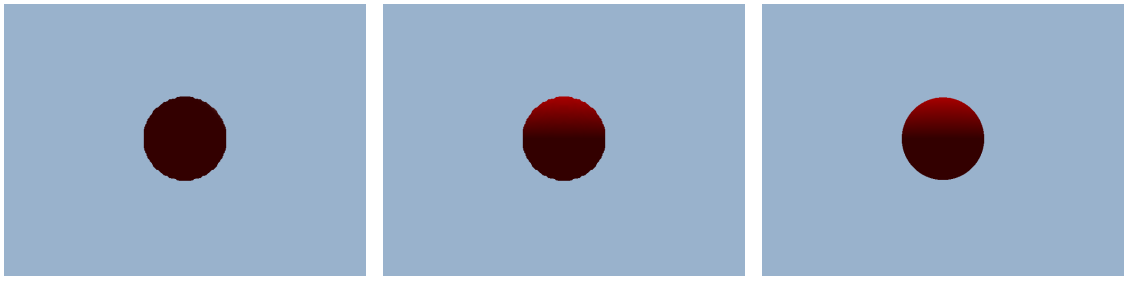
Question 8 : Dans les shaders, on trouve des variables de type isampler2D et d'autres de type sampler2D. Quelle est la différence entre ces deux types ? Que réalise la fonction texelFetch ?

Les documentations font réferences à *g*sampler2d ou g est remplacé par rien, u ou i. Rien signifie que le sampler2D contenant des coordonnées de textures sera exprimé en float, i par des entiers signés et u par des entiers non signés.

TexelFetch recherche le texel (pixel de texture) correspondant à la coordonnées de texture qui lui a été fournit et à la texture.

Question 9 : Corrigez le calcul des normales en supposant que sa direction est donnée par le gradient de la fonction de densité. Quel fichier avez-vous modifié pour cela ?

L'aspect crénelé de la sphère vient du fait que la position des vertex est toujours choisie au milieu des arètes des cubes. Une meilleure approche consiste à réaliser une interpolation linéraire pour choisir cette position. Question 10 : Réalisez cette interpolation de manière à obtenir une sphère bien lisse comme celle de la figure 9.

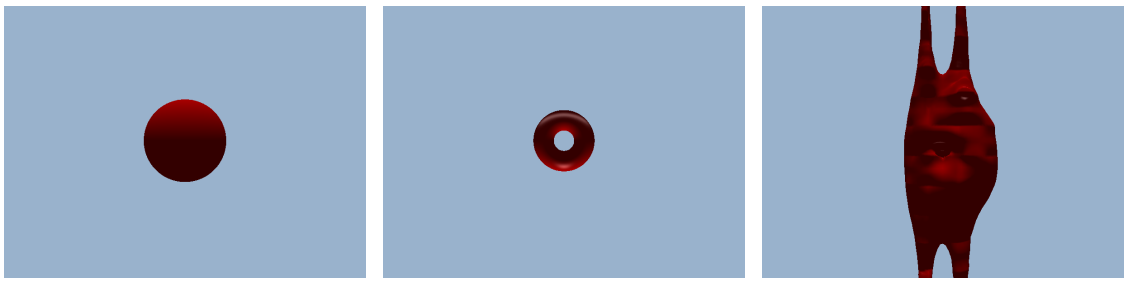


(a) (b) (c)

FIGURE 9 – Marching Cubes Original (a) ; Calcul des normales fonction du gradient de la fonction de densité (b) ; Interpolation linéaire de placement des vertex par rapport aux arrètes (c)

2.4 Fonction de densité

Images des différentes fonctions de densité qu'on peut réaliser + celle de la plus intéressante qu'on ait



(a) (b) (c)

FIGURE 10 – Marching Cubes avec différentes fonctions de densite : Boule (a), Torus (b), Création de roches (c)

2.5 Amélioration graphique & Textures

Question 12 : Mettez en place le calcul pondéré des textures par les normales pour obtenir une image du genre de la figure 18.

2.6 Combinaison Parallax Mapping + Marching Cubes