

Rapport de Projet

Implémentation d'un visioneurs de fichiers gltf

Clément Chomicki

April 4, 2021

Contents

1	Généralités	1
1.1	Utilisation	1
1.2	Limitation des FPS	1
1.3	Modes d'affichage	1
2	Normal Maps	2
2.1	Modèles possédant des tangentes	2
2.2	Modèles sans tangentes	2
2.3	Calculer les tangentes à la volée	2
2.4	Difficultés rencontrées	2
3	Connaissances acquises	4

1 Généralités

1.1 Utilisation

- Compilation `cmake_prepare && cmake_install`
- lancer le programme `./build/bin/gltf-viewer viewer <path-to-gltf>`
Les commandes `view_helmet` et autres fonctionnent toujours.

1.2 Limitation des FPS

Ma carte graphique produisait un fort sifflement en faisant tourner le programme (ma version et la version “triche” de la même façon). Je me suis longtemps demandé pourquoi, et j'ai finalement trouvé: le programme est

trop performant et on était dans les 1000 fps, ce qui faisait probablement vibrer mon bus CPU \Leftrightarrow GPU ou alors un bus interne à la carte graphique. Je n'aime pas quand mon matériel menace d'explorer, j'ai donc ajouté une limite de FPS en créant un objet attendant un certain délai avant sa destruction à chaque itération.

1.3 Modes d'affichage

Pour pouvoir débugger mais aussi par curiosité, j'ai décidé de mettre plusieurs modes d'affichages, permettant de voir quelques grandeurs comme les normales ou bien les dérivées des UVs.

2 Normal Maps

2.1 Modèles possédant des tangentes

Les modèles 3D peuvent être exportés avec leurs tangentes, ce qui se traduit par un attribut "TANGENTE" supplémentaire pour les primitives des fichiers gltf. Dans ce cas, il suffit de traiter cet attribut comme les autres dans `createVertexArrayObject`.

Une fois que l'on a accès aux tangentes dans le shader, on peut procéder comme dans ce tutoriel.

2.2 Modèles sans tangentes

On pourrait précalculer les tangentes des modèles. J'ai tenté de faire cela, mais je n'ai pas eu le temps d'obtenir quelque chose de viable. J'avais néanmoins une alternative déjà implémentée qui fonctionne.

2.3 Calculer les tangentes à la volée

Le fragment shader permet de donner accès à dérivée en x et en y des variables, ainsi on peut avoir accès aux dérivées des coordonnées de textures ainsi que de position dans le viewspace. Avec la capacité de différentier des variables, on a la possibilité de déduire les tangentes dans le fragment shader. C'est la méthode présentée par cet article, j'ai adapté le codé présenté, et le résultat est fortement concluant.

2.4 Difficultés rencontrées

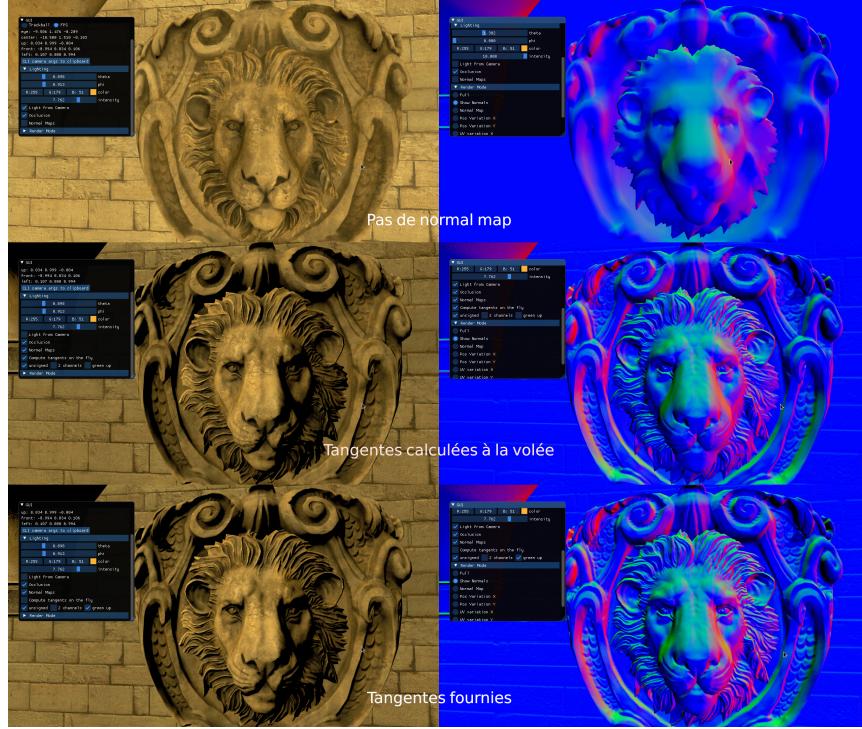
Un problème rencontré est le manque de standard pour les normal maps. En effet, certains utilisent des normal map dont les bits peuvent être signés et non-signés, certains représentent le vert comme la direction vers le haut ou vers le bas, voir même certains n'incluent pas de coordonnée verte, qui doit alors être déduite des deux autres (le vecteur normal étant alors supposé normé). Il n'y a pas de moyen d'y remédier automatiquement. Aussi, j'ai rajouté des options pour l'interprétation des normal maps.

Un second problème évident est le temps. Si j'avais eu plus de temps, j'aurais probablement en plus de l'approche temps réel essayé de précalculer les tangentes des modèles n'en présentant pas (bien que ce ne soit peut-être pas forcément bien plus avantageux).

Un autre problème rencontré, et qui est toujours là (et qui n'est peut-être pas de mon dû), est une aberration au niveau de l'axe vertical pour certains triangles en mode précalculé, où la normale est déviée dans la direction opposée à ce qu'elle devrait être. Cela arrive notamment aux endroits reflétris horizontalement, où l'axe z se retrouve d'un côté à l'endroit et de l'autre à l'envers.



Ce problème est inexistant avec la méthode temps réel.



3 Connaissances acquises

J'ai approfondis les connaissances sur les normal maps que j'avais déjà acquises dans des projets antérieurs, et j'ai renforcé mes connaissances en OpenGL et glsl, notamment en découvrant les opérateurs de différentiations.