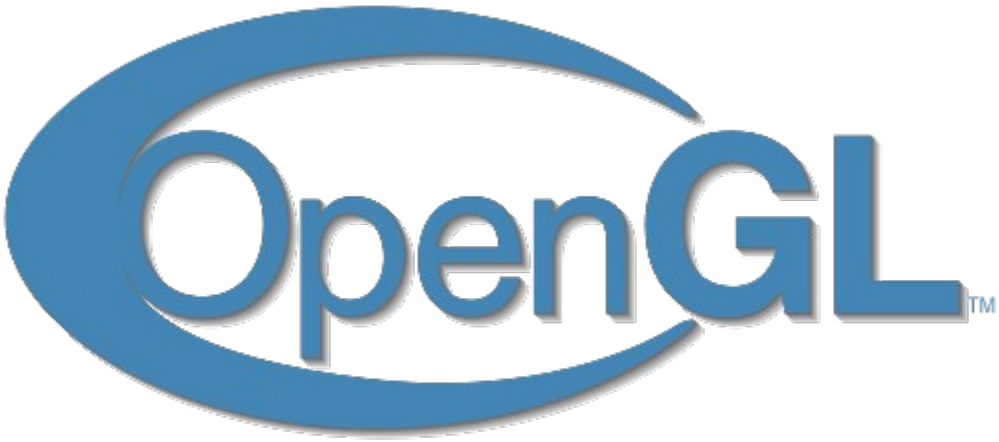
Thibault Fiévet – IMAC 2015
https://github.com/ThibaultFievet/toon-shading

Projet d'OpenGL Avancé :

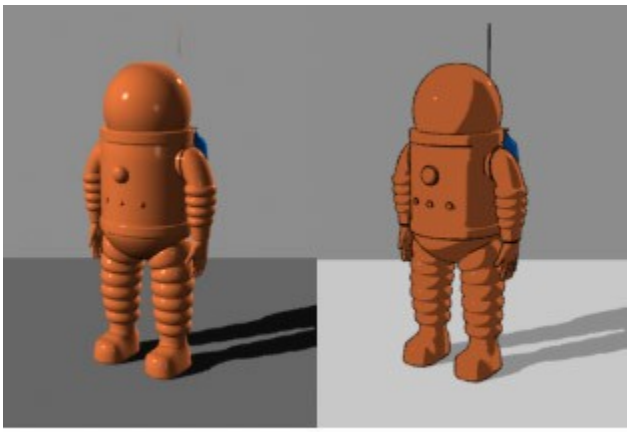
Le Toon Shading



Sommaire

Contexte :

Le Toon Shading (ou Cel Shading) est une technique de rendu non réaliste utilisé pour donner un aspect cartoon ou bande dessinée à une scène 3D. Il s'appuie sur des dégradés de couleurs moins prononcés et cherche à marquer les contours des éléments.



Actuellement très en vogue dans le milieu du jeu vidéo via des productions comme « Borderlands », « The Legend of Zelda Wind Waker », « Dragon Ball Xenoverse » ou les adaptations de comics, mangas et autres, ce système de rendu semble convaincre autant les joueurs pour son esthétique plaisante qu'aux programmeurs pour son implémentation assez facile et peu coûteuse en ressource. Il m'a ainsi semblé naturel de vouloir tester cette technique, qui me parle beaucoup d'un point de vue artistique et culturel, et d'autant plus que je serai probablement amené un jour à travailler sur des jeux vidéo employant cette méthode.

Pour implémenter mon toon shading, j'ai fait le choix d'utiliser des modèles 3D complexes au format OBJ. Cela m'a permis d'avoir un rendu plus agréable à l’œil qu'avec un simple assemblement de primitives, et également de tester plus facilement les algorithmes implémentés. Néanmoins, ayant dû recourir à une librairie externe pour le chargement de ces modèles, une part importante du développement de ce projet a été consacré à l'apprentissage et à l'utilisation de cette dernière. Il s'agit d'Assimp, que j'ai choisi pour son interopérabilité, sa légèreté et le grand nombre de formats de modèles 3D reconnus.

Il est à noter que la démo produite tourne sur OpenG 3.3, qui est la seule version disponible sur mon ordinateur personnel.

Techniques utilisés :

1. Dégradé de lumière

Des éléments caractéristiques des cartoons sont la lumière et le système d'ombrage très démarqué. Contrairement aux dégradés que l'on peut apercevoir dans un éclairage plus réaliste, les coupures dans l'intensité de la lumière dans un toon shading sont très nettes.

La technique consiste à quantifier l'intensité lumineuse et à créer différents niveaux de luminosité. On cherche à former des discontinuités dans la lumière via ces niveaux. Pour cela on calcule le produit scalaire suivant, ce qui ne diffère nullement avec un rendu différent comme le Blinn-Phong :

ndotl = max(0, dot(L,N))
avec L La direction de la lumière et N la normale de la face.

On peut ensuite déterminer la composante diffuse de la lumière :

floor(diffuse * levels) * scaleFactor
avec levels le nombre de discontinuités voulus et scaleFactor l'intervalle de luminosité entre 2 niveaux.

Plus on rajoute de niveaux, plus l'image sera détaillée et tendra vers un système d'éclairage réaliste. Afin de conserver une esthétique cartoon, on évitera de dépasser 5 niveaux de luminosités.

2. Détection de contour

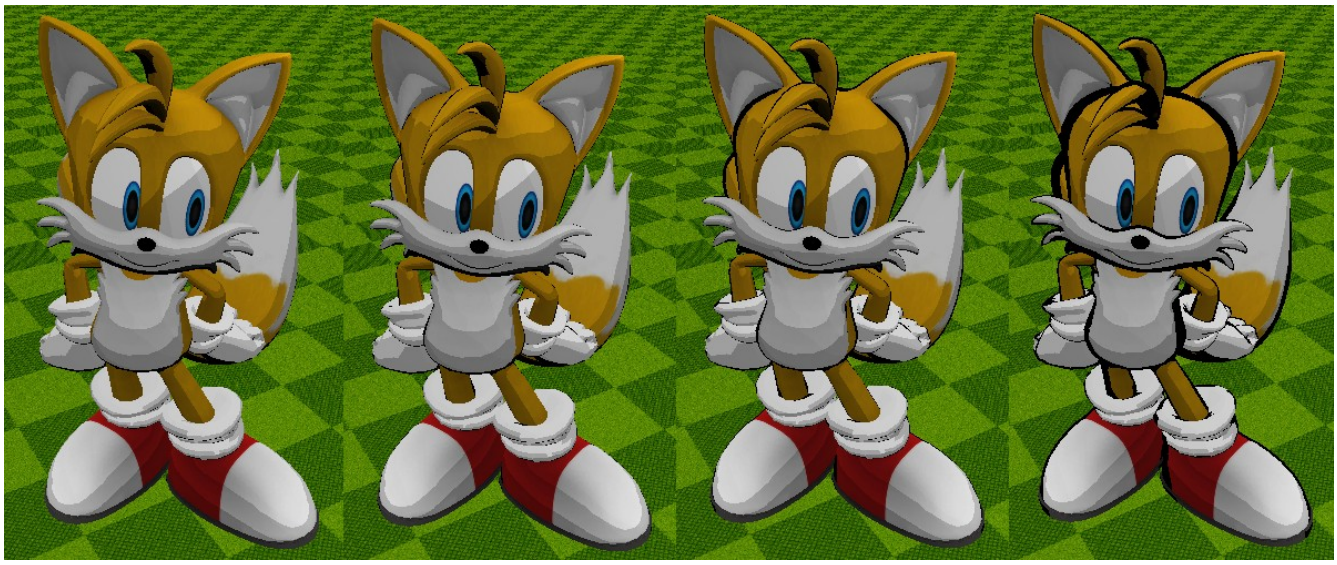
Le toon shading empreintant son esthétique à la culture du dessin, il n'est pas étonnant de le voir essayer d'en reproduire aussi les contours, pour donner un effet « fait main » et marquer d'avantage les contrastes et les formes.

Plusieurs méthodes existent pour dessiner des contours autour de ses éléments 3D. Une méthode intuitive et facile à déployer consiste à créer une silhouette du modèle que l'on veut entourer, légèrement plus grande et entièrement noire. Ainsi lorsque l'on imprimera le modèle en couleur par dessus, les contours resteront visibles. Néanmoins, cette technique pose des soucis, notamment en terme d'aliasing. En effet, la silhouette étant plus grande, des trous dans le maillage peuvent apparaître et les pixels peuvent devenir visibles. J'ai trouvé plus probant d'utiliser un algorithme de détection des contours.

Le principe est très simple, puisqu'il consiste à vérifier via la normale d'un vertex si celui ci est sur un bord du modèle :

if (dot(V, N) < DetectionFactor)
avec V la position de la camera, N la normale de la face et DetectionFactor le seuil de tolérence compris entre 0 (contour très fin) et 1 (le modèle entier est un contour).

Si la condition est remplie, nous sommes sur un point du contour, il nous suffit alors d'en changer la couleur.

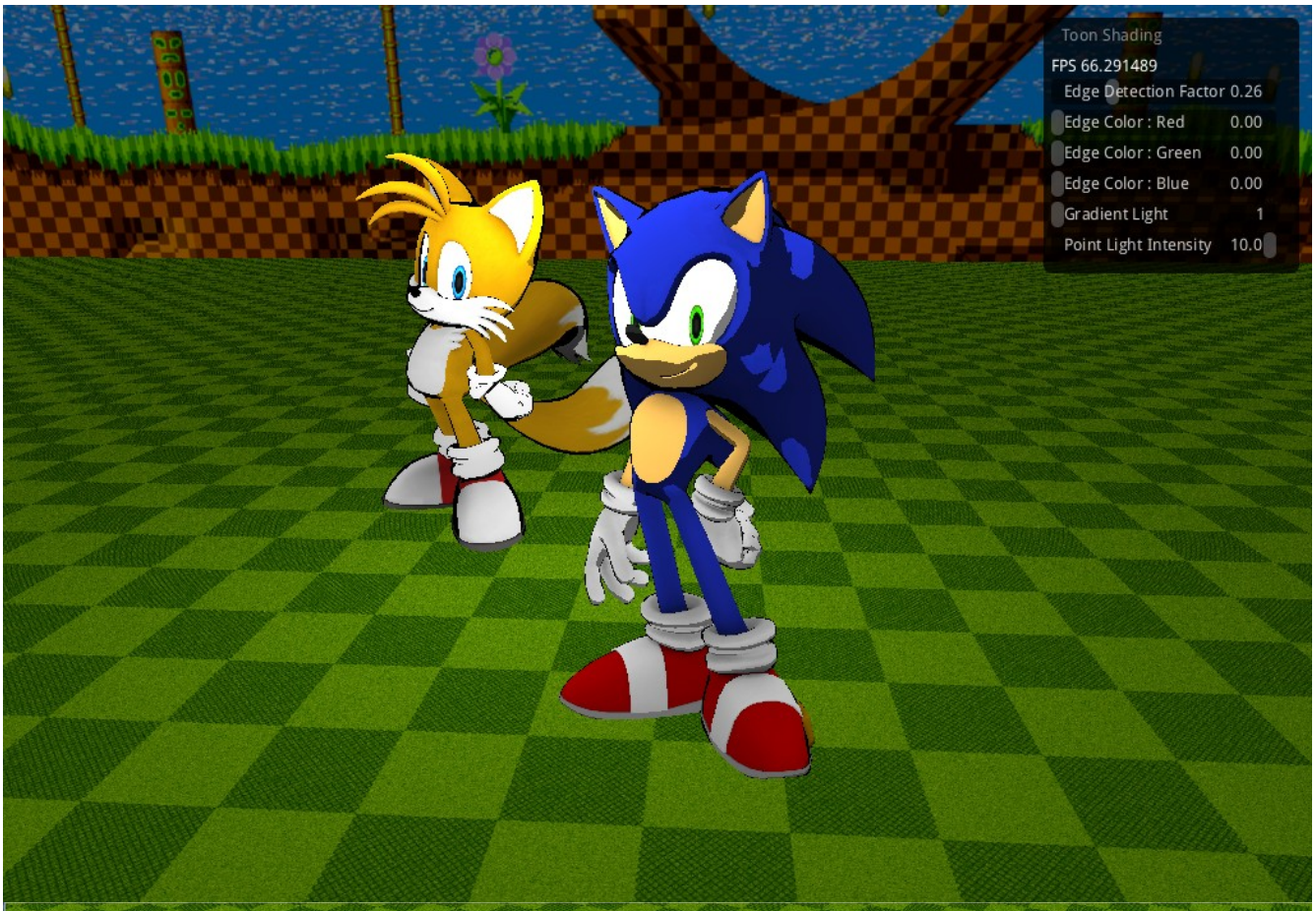


Malheureusement, cette méthode présente aussi ses défauts. En effet, le contour n'est pas homogène, et certaines zones sont plus marquées que d'autres. De l'aliasing peut également apparaître, bien que des algorithmes permettent d'adoucir le contour.

Rendus :


Toon Shading
FPS 61.311623
Edge Detection Factor 0.20
Edge Color : Red 0.00
Edge Color : Green 0.00
Edge Color : Blue 0.00
Gradient Light 3
Point Light Intensity 1.0


Toon Shading
FPS 59.165077
Edge Detection Factor 0.16
Edge Color : Red 0.00
Edge Color : Green 0.00
Edge Color : Blue 0.00
Gradient Light 2
Point Light Intensity 1.0


Toon Shading
FPS 66.291489
Edge Detection Factor 0.26
Edge Color : Red 0.00
Edge Color : Green 0.00
Edge Color : Blue 0.00
Gradient Light 1
Point Light Intensity 10.0


Toon Shading
FPS 59.456928
Edge Detection Factor 0.35
Edge Color : Red 1.00
Edge Color : Green 1.00
Edge Color : Blue 1.00
Gradient Light 3
Point Light Intensity 5.1

Références bibliographiques :

http://sunandblackcat.com/tipFullView.php?l=eng&topicid=15

http://gamedev.dreamnoid.com/2009/03/11/cel-shading-en-glsl/

http://in2gpu.com/2014/06/23/toon-shading-effect-and-simple-contour-detection/

http://en.wikibooks.org/wiki/GLSL_Programming/Unity/Toon_Shading

http://prideout.net/blog/?p=22

http://www.lighthouse3d.com/tutorials/glsl-tutorial/toon-shading/

https://en.wikipedia.org/wiki/Cel_shading