Synthèse d'Images
Travaux Pratiques

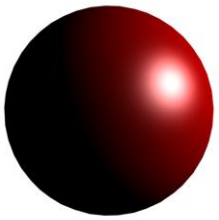
Shaders : Programmation GPU en OpenGL

On partira du code du programme localisé dans l'archive associée au TP et on expérimentera sur les maillages au format OFF du répertoire models. On s'appuiera sur la documentation disponible sur le site www.opengl.org concernant le langage GLSL. Le sujet est écrit pour une implémentation en C++ et GLSL, mais une implémentation en C (structures et fonctions) est autorisée. On pourra librement employer des modèles OFF trouvé le net (quelques modèles sont fournis dans l'archive). Ce TP suit celui sur l'éclairage, les BRDF et l'ombrage. Dans le TP précédent, la qualité du résultat d'éclairage était limitée par la précision géométrique du maillage et la performance du rendu était bridée par l'exécution sur CPU de l'ensemble des calculs. L'objectif est maintenant de calculer l'éclairage (tout du moins l'évaluation de la BRDF) par pixel en programmant directement le GPU à l'aide de programmes spécifiques, communément appelés *Shaders.* En plus de GLUT, GLEW et un GPU supportant OpenGL 2.0+ sont nécessaires.

I. Compiler le programme et utilisez-le pour charger un objet. L'objet est tout blanc car la fonction GLSL régissant la couleur des pixels (dans le code du fragment shader shader.frag) place une valeur erronée par défaut pour tous les pixels (gl_FragColor). Observez le code de chargement des shaders dans le pogramme C++ ainsi que le code GPU pour le vertex (fichier shader.vert) et le fragment shader (fichier shader.frag).

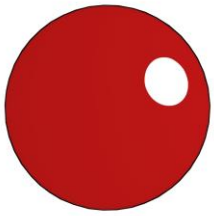
On notera que les shaders sont compilés par le driver OpenGL et envoyé vers le GPU automatiquement. Au passage, OpenGL « nettoie » le shader de toutes les variables non utilisé. Attention donc à toujours utiliser les variables déclarées dans le shader, sous peine d'avoir une erreur à l'exécution lorsqu'on essaie de mettre à jour ces variables de shader depuis le CPU.

II. Implémentez la BRDF de Blinn-Phong dans le fragment shader (fichier shader.frag). Notez que vous disposez de la position (P) et de la normal à la surface (N) pour chaque fragment ainsi que des paramètres de la source de lumière et de la BRDF. Observez la différence avec le rendu OpenGL classique sur l'objet *sphere.off* par exemple.



III. On souhaite maintenant avoir un calcul de radiance plus proche de la réalité physique : implémenter la BRDF GGX. Les différents paramètres d'un shader peuvent être modifiés depuis le programme CPU hôte lorsque qu'ils sont déclarés comme variables uniform. (voir la documentation ici). Modifier le programme de manière à pouvoir controller la rugosité de la BRDF avec les touches +/- et de manière à animée la position et la couleur de la source de lumière en lui faisant suivre une trajectoire fermée.

IV. Modifier le fragment shader pour prendre en compte plusieurs lumières.



V. La programmation GPU permet toutes sortes de rendus différents pour une même scène. On souhaite maintenant produire un rendu *non photo-réaliste* (NPR) de type « cartoon ». Pour se faire, le fragment shader doit générer de grands aplats de couleur unis. L'idée est de choisir trois couleurs (blanc pour le spot spéculaire, arbitraire pour la couleur des objets et noir pour les contours). Implémentez ce rendu au niveau du fragment shader. Votre fonction doit pouvoir discriminer les fragments afin de leur affecter l'une des trois couleurs.