41-Shader

L'un des principaux objectifs des cartes graphiques est d'effectuer le rendu d'objets 3D dans un tampon d'images 2D afin de les afficher. La façon dont l'objet est rendu est déterminée par ce que l'on appelle le "modèle d'ombrage", qui reçoit généralement des informations telles que la couleur et la position de l'objet, la couleur et la position de l'éclairage, les coordonnées de la texture et les caractéristiques du matériau, comme la brillance ou la matité de l'objet. . Le programme qui s'exécute dans le matériel ou le logiciel pour effectuer ce calcul s'appelle le ***shader***. Traditionnellement, les cartes graphiques utilisaient un pipeline d'ombrage fixe pour appliquer un modèle d'ombrage à un objet, mais ces dernières années, les cartes graphiques ont acquis des pipelines programmables afin que des shaders personnalisés puissent être exécutés à la place du pipeline fixe. Pour un résumé de plusieurs des façons de contrôler le pipeline OpenGL fixe, voir *Tutoriel 35*: ***Éclairage et brouillard***.

> **Configuration matérielle requise**: pour profiter pleinement de ce didacticiel, vous aurez besoin d'une carte graphique qui prend en charge les shaders programmables, par ex. les cartes graphiques ATI Radeon 9200, NVIDIA GeForce 5000 ou plus récentes. Il est également recommandé de mettre à jour votre pilote OpenGL avec la dernière version disponible pour votre carte graphique. Sur Macintosh, cette mise à jour est fournie avec la dernière mise à jour du système d'exploitation.

Ombrage plat

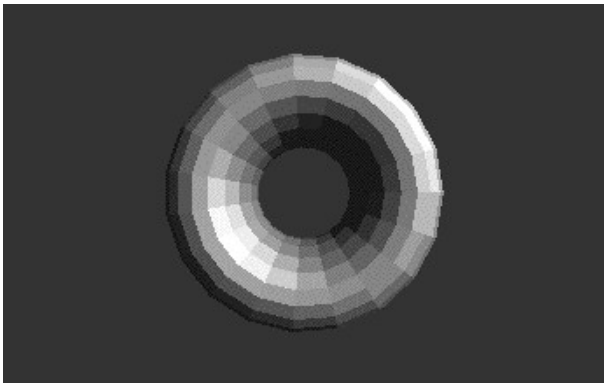
L'un des modèles d'ombrage les plus simples qui prend en compte l'éclairage est appelé ***Flat Shading*** ou ***Facet Shading***, où chaque polygone a une couleur unique sur l'ensemble du polygone, en fonction de la normale à la surface et des propriétés d'éclairage. Comme nous l'avons vu dans le Tutoriel 35, cela peut être accompli dans Jitter en définissant l'attribut **lighting_enable** d'un objet JGL OpenGL (par exemple *jit.gl.gridshape*) à **1**.

Pour commencer

• Ouvrez le patch Tutorial *41jShaders* dans le dossier Jitter Tutorial. Cliquez sur le *toggle* intitulé ***Start Rendering***.

• Cliquez sur la boîte *toggle* située au-dessus de l'objet boîte de *message* lisant **lighting_enable $1** pour activer l'éclairage pour l'objet *jit.gl.gridshap*e dessinant le tore. L'attribut **smooth_shading** est à **0** par défaut.

• Vous devriez maintenant constater un changement dans l'éclairage du tore. Au lieu de l'apparence grise terne qu'il avait au départ, vous verrez une apparence grise brillante comme ceci:

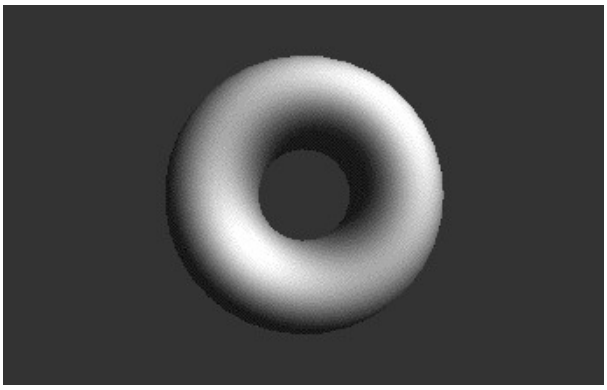


Le rendu d'un tore montrant un ombrage plat.

Ombres lisses

Bien que ce soit un aspect souhaitable, la couleur de la plupart des objets dans le monde réel change de façon régulière sur la surface. Le modèle "Gouraud Shading" (1971) a été l'un des premiers modèles d'ombrage à traiter efficacement ce problème en calculant une couleur par sommet en fonction de la normale à la surface et des propriétés d'éclairage. Il interpole ensuite linéairement la couleur sur le polygone pour obtenir un aspect d'ombrage lisse. Si les artefacts de cette approche simple peuvent sembler étranges lorsqu'on utilise un petit nombre de polygones pour représenter une surface, lorsqu'on utilise un grand nombre de polygones, les artefacts visibles de cette approche sont minimes. En raison de l'efficacité de calcul de ce modèle d'ombrage, "Gouraud Shading" a été assez populaire et reste le principal modèle d'ombrage utilisé aujourd'hui par les cartes graphiques d'ordinateur dans leur pipeline d'ombrage fixe standard. Dans Jitter, ceci peut être accompli en définissant l'attribut **smooth_shading** sur **1** (on). Faisons-le dans notre patch, en augmentant et en diminuant le nombre de polygones en modifiant l'attribut **dim** de l'objet *jit.gl.gridshape*.

• Cliquez sur la la boîte *toggle* attachée à la boîte de *message* indiquant **smooth_shading $1**. Essayez d'augmenter et de diminuer le nombre de polygones en modifiant la boîte de *nombres* attachée à la boîte de *message* indiquant **dim $1 $1** dans le coin inférieur droit du patch.



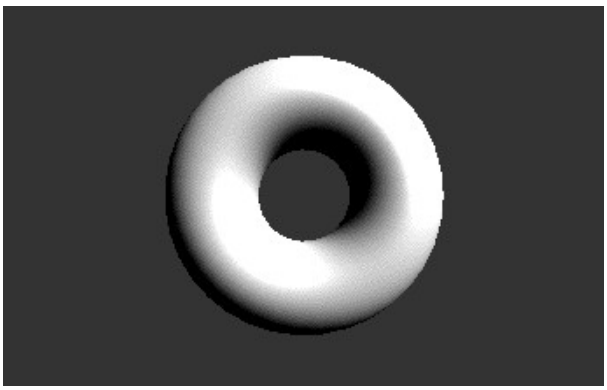
Le rendu d'un tore avec un ombrage lisse.

Éclairage par pixel

Comme nous l'avons mentionné, l'ombrage lisse avec un faible nombre de polygones présente des artefacts visibles qui résultent de l'exécution du calcul de l'éclairage sur une base par sommet, en interpolant la couleur à travers le polygone. Phong Shading (1975) lisse non seulement la couleur des sommets à travers un polygone, mais lisse également les normales de surface pour chaque sommet à travers le polygone, en calculant une couleur par fragment (ou pixel) sur la base des normales lissées et des paramètres d'éclairage. Le calcul des valeurs d'éclairage par pixel est appelé "éclairage par pixel". Cette méthode est plus coûteuse en calcul que le Gouraud Shading mais donne de meilleurs résultats avec moins de polygones. L'ombrage par pixel n'est pas dans le pipeline fixe OpenGL; cependant, nous pouvons charger un ***shader*** personnalisé dans le pipeline programmable pour appliquer ce modèle d'ombrage à notre objet.

• Chargez le shader d'éclairage par pixel en cliquant sur la boîte de *message* qui indique **read mat.dirperpixel.jxs** connecté à l'objet *jit.gl.shader*.

• Appliquez le shader à notre objet en cliquant sur la boîte de *message* **shader shademe** connectée à l'objet *jit.gl.gridshape*. Ceci définit l'attribut **shader** de l'objet pour référencer l'objet *jit.gl.shader* par son nom (**shademe**).



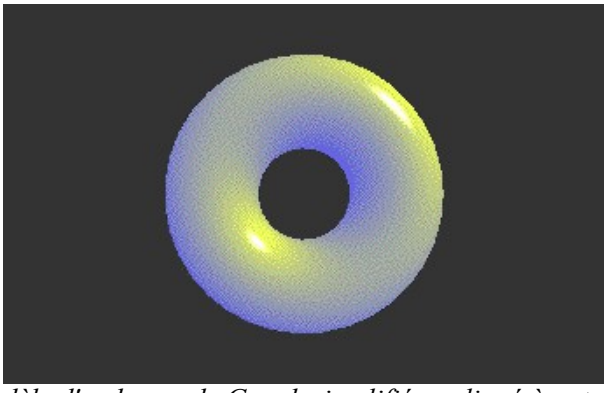
Ombrage par pixel appliqué au tore.

Shaders programmables

En 1984, Robert Cook a proposé le concept d'un arbre d'ombrage, qui permet de construire des modèles d'ombrage arbitraires à partir de certaines primitives fondamentales en utilisant un "langage d'ombrage". Ce langage d'ombrage permettrait d'étendre un pipeline de rendu afin de prendre en charge un nombre infini de shaders plutôt qu'une poignée de shaders prédéfinis. Ken Perlin a étendu le langage d'ombrage de Cook pour contenir les structures de contrôle et est devenu le modèle utilisé par le populaire langage d'ombrage RenderMan de Pixar. Plus récemment, les langages d'ombrage Cg et GLSL axés sur les GPU ont été créés sur la base de principes similaires. Les shaders personnalisés utilisés dans ce didacticiel, y compris le calcul d'éclairage par pixel, ont été écrits en GLSL.

Une brève introduction à l'écriture de vos propres shaders sera abordée dans le prochain tutoriel. Pour l'instant, continuons à utiliser des shaders préexistants et montrons comment nous pouvons modifier dynamiquement les paramètres des shaders. L'ombrage de Gooch est un modèle d'ombrage non photo-réaliste développé principalement pour l'illustration technique. Il dessine des contours de l'objet et utilise des couleurs chaudes et froides pour donner le sentiment de profondeur souhaitée pour les illustrations techniques. Nous avons un shader personnalisé qui implémente une version simplifiée de Gooch Shading qui ignore l'application des contours.

• Chargez le shader de Gooch simplifié en cliquant sur la boîte de *message* **read mat.gooch.jxs** connecté à l'objet *jit.gl.shader*.



Le modèle d'ombrage de Gooch simplifié appliqué à notre tore.

Vous devriez remarquer que le ton chaud est une couleur jaune et le ton froid est une couleur bleue. Ces valeurs ont été définies comme des valeurs par défaut dans notre fichier de shader, mais elles sont exposées comme des paramètres qui peuvent être remplacés par des messages vers l'objet *jit.gl.shader*.

• En utilisant les boîtes de *nombres* du patch, envoyez les messages **param warmcolor <red> <blue> <green> <alpha>** et **param coolcolor <red> <green> <blue> <alpha>** à l'objet *jit.gl.shader* pour changer les tons utilisés pour les couleurs chaudes et froides.

Nous pouvons déterminer les paramètres disponibles pour le shader en envoyant à l'objet *jit.gl.shader* le message **dump params** pour imprimer les paramètres dans la console Max, ou en envoyant le message **getparamlist**, qui produira une liste de paramètres à la sortie la plus à droite (dump) de l'objet. La valeur actuelle d'un paramètre individuel peut être interrogée avec le message **getparamval <parameter-name>**. La valeur par défaut d'un paramètre peut être interrogée avec le message **getparamdefault <parameter-name>**, et le type du paramètre peut être interrogé avec **getparamtype <parameter-name>**.

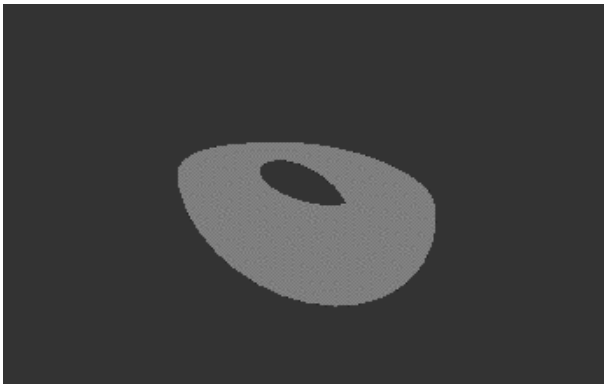
Programme des sommets

Les shaders peuvent être utilisés non seulement pour déterminer la couleur de surface des objets, mais aussi la position et les attributs des sommets de notre objet, comme nous le verrons dans nos prochains tutoriels. Pour l'instant, intéressons-nous au traitement des sommets. Dans les shaders précédents, nous avons juste discuté de la façon dont les différents shaders rendaient les pixels. En fait, pour chacun de ces exemples, nous avons exécuté deux programmes: un pour traiter les sommets (le programme des sommets) et un effectuer le rendu des pixels (le programme des fragments). Le programme des sommets est nécessaire pour transformer l'objet dans l'espace 3D (rotation, translation, mise à l'échelle), ainsi que pour calculer l'éclairage par sommet, la couleur et d'autres attributs. Puisque nous voyons l'objet se déplacer et tourner lorsque nous utilisons l'objet *jit.gl.handle* dans notre patch, il est évident qu'un programme de vertex doit être exécuté. Logiquement, le programme de vertex s'exécute sur le processeur de vertex programmable et le programme de fragment s'exécute sur le processeur de fragment programmable. Ce modèle correspond au pipeline à fonctions fixes qui sépare également les tâches de traitement des sommets et des fragments.

Le programme de sommet personnalisé des exemples précédents n'a cependant pas effectué d'opération visiblement différente dde celle du programme de sommet du pipeline fixe. Chargeons donc un shader de sommet qui a un effet plus spectaculaire. Le shader ***vd.gravity.jxs*** peut pousser et tirer la géométrie en fonction de la distance entre le sommet et un point dans l'espace 3D.

• Chargez le shader simplifié de déplacement de sommet par gravité en cliquant sur la boîte de *message* **read vd.gravity.jxs** connecté à l'objet *jit.gl.shader*.

• Contrôlez la position et la quantité du shader de déplacement des sommets par gravité en modifiant les boîtes de *nombres* connectées à l'objet *pak* et la boîte de *message* produisant les messages **param gravpos <x> <y> <z>** et **param amount <n>,** respectivement.



Vertex Distortion.

Résumé

Dans ce tutoriel, nous avons abordé les pipelines fixes et programmables disponibles sur la carte graphique et montré comment utiliser le pipeline programmable en chargeant des shaders personnalisés avec l'objet *jit.gl.shader*. Les shaders peuvent ensuite leur être appliqués à objets 3D pour obtenir différents effets. Nous pouvons également définir et interroger les paramètres des shaders par le biais de messages adressés à l'objet *jit.gl.shader*.