SIRE – TD

Programmation Graphique : Premiers pas (1)

Le but de ce TD, qui débute une série de 3, est de se familiariser avec les concepts liées à la synthèse d'image sur carte graphique, et donc leur programmation. Chaque séance verra une progression dans le transfert des calculs du CPU vers le GPU, et dans le passage d'une bibliothèque de programmation graphique d'ancienne génération vers celle actuellement utilisée (OpenGL > 3.0, OpenGL ES > 2.0, WebGL, Direct3D > 10, …). Cette progression se fera au sein d'un seul et unique programme de génération de Terrain (tp1.tgz).

Pour toute documentation, profitez d'Internet et de tout ce qui s'y trouve ! Creusez, fouillez, apprenez à trouver la solution par vous-même ! Sinon, vous pouvez toujours consulter les pages concernant OpenGL.

http://www.opengl.org/resources/

1. Implémentation du calcul du frame-rate

Afin d'évaluer l'impact de nos changements, il nous faut d'abord implémenter une méthode de calcul du frame-rate. Pour cela, observer le corps du programme `Main.cpp`. Il est constitué principalement d'une initialisation du rendu (du premier `glutInit` à `init`), de la fonction d'affichage (`glutDisplayFunc`) et de l'action à faire lors que le CPU est libre (`glutIdleFunc`). Observez bien ces fonctions, et l'ordre des appels, qui seront presque toujours les mêmes !

Pour le calcul du frame-rate, il faut implémenter un compteur qui s'incrémente à chaque affichage, et mesurer le temps passé T (en seconde) pour afficher X images. Les fichiers `timer.cpp` et `timer.h` peuvent vous y aider. Le frame-rate est simplet X/T fps (ips pour images par seconde).

2. Manipulation des variables uniformes

Le rendu se fait à travers les shaders dont le code se situe dans les fichiers `vert.sha` (`vertex shader`) et `frag.sha` (`fragment shader`). Observer la séquence de chargement du shader dans `DefaultShader.cpp`. Les données du terrain sont générées et affichées dans le fichier `Relief3D.cpp`. Observer aussi la manière dont la position de la source de lumière est transferé. Faites en sorte que plus aucun appel à `glLightfv` et autre `glEnable(GL_LIGHT0)` ne soit utilisés :ces appels fixes étant obsolètes dans les bibliothèques modernes.

Mots clefs : `uniform`, `glUniform`, `glGetUniformLocation`.

3. Manipulation des attributs de sommets

Le transfert des couleurs, normales, position des sommets se fait actuellement par les fixes `glNormal`, `glColor` et `glVertex` et par l'appel au attribut de sommets pré-définis `gl_Normal`, `gl_Color` et `gl_Vertex`. Vous pouvez les remplacez par les versions programmables.
Attention ! Pour des raisons de compatibilités, l'attribut vertex doit être à la position 0 !

Mots clefs : `in`, `glVertexAttrib`, `glGetAttribLocation`, `glBindAttribLocation`.

4. Transfert du calcul des couleurs sur GPU.

4.1 Actuellement, le calcul de la couleur par sommet est faite sur CPU puis transféré sur la carte graphique. Évaluez la complexité de calcul sur CPU et de transfert mémoire en terme d'initialisation et par image.
4.2 Transférer ce calcul sur GPU et évaluez de nouveaux ces coûts et leur impacte sur les performances.
4.3 Introduisez un facteur d'échelle pour les hauteurs au-dessus du niveau de la mer, et un autre pour ceux en-dessous.

5. Réduisons encore plus les transferts.

Actuellement, même si la géométrie est fixe, elle est transféré à chaque image, ce qui a un impact sur le temps de rendu.

Une première solution, est de supprimer les appels obsolètes à `glBegin(GL_TRIANGLE_STRIP)` et `glEnd()`, qui force la synchronisation entre la carte et le CPU. Pour cela, on transmettre l'adresse des données en utilisant `glVertexAttribPointer` et `glEnableVertexAttribArray`; et les afficher par une seule commande par ligne `glDrawRangeElements`.

A partir de maintenant, la position des attributs n'a plus d'importance.