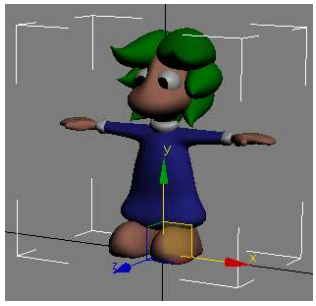
TP OpenGL : Séance 2

Opérations par fragments :
Fragment Shaders, éclairage et depth-buffer

Dans cette séance, un modèle 3D *(un gentils petit Lemming)* comportant positions, normales et couleurs aux sommets est fournit. Ses coordonnées sont définies dans le repère représenté sur la figure ci-contre.



Question 1 : « Précédemment dans OpenGL… »

Avant d'attaquer le cœur de ce TP et afin de rappeler ce qui a été fait à la séance précédente (en particulier les matrices de transformation) nous allons commencer par créer une petite scène animée composée de 4 Lemmings disposés en cercle tournant autours du centre de la scène.

Le code fournit affiche simplement le Lemming sans ses couleurs. La première chose à faire est d'ajouter (dans la fonction `initLemming()`) des informations de couleurs par sommets. Ces informations sont fournies par la fonction `getModelColors(0)` et peuvent être passées au *vertex shaders* dans un attribut `vertexColor`.

Pour la disposition et l'animation de la scène, vous avez besoin de définir une matrice « ModelToWorld » qui disposera les Lemmings dans la scène, ainsi qu'une matrice « WorldToView » qui placera la caméra contrôlée au clavier et à la souris. Le contrôle de la caméra doit être « à la quake » c'est à dire que les déplacements avant/arrière et droite/gauche du point de vue sont contrôlés au clavier et sa rotation (mouvements de tête) contrôlée à la souris. La position de la vue est définie dans la variable `viewPos`, sa direction dans `viewOrient` (déjà définis à partir du clavier/souris). La rotation autours du centre est définie dans la variable `rotAngle`.

Les deux matrices « ModelToWorld » et « WorldToView » doivent être multipliées en une matrice « ModelView » avant d'être passée au *Vertex Shader* afin d'éviter que cette multiplication soit faite pour chaque sommet.

Question 2 : Illumination dans le vertex shader

2.1 Phong

Maintenant que nous avons une jolie scène animée et colorée, nous allons ajouter le calcul d'un modèle d'illumination tel que celui vu en cours sur nos Lemmings afin de leur donner une apparence un peu moins « plate ». Ce calcul sera effectué dans un premier temps dans le vertex shader.

Vous disposez pour cela de la position de la source de lumière définie dans la variable `lightPos` et controlée à la souris (après appui sur la touche 'l'). Celle-ci-doit etre transmise au vertex shader dans lequel une fonction `shade` doit être completée afin de calculer une illumination de Phong (cf. http://fr.wikipedia.org/wiki/Ombrage_Phong). Vous utiliserez

ensuite cette fonction pour calculer la couleur (varying `color`) transmise au fragment shader à partir des vecteurs N (normale transmise par sommet), L (direction du sommet vers la source de lumière) et V (direction du sommet vers le point de vue). Faite bien attention d'exprimer tous ces vecteurs dans le même espace (je vous conseil le repère de la vue vue).

2.2 Blinn-Phong

Modifiez la fonction shade afin d'appliquer la modèle de Blinn-Phong à la place de Phong (cf. http://en.wikipedia.org/wiki/Blinn%E2%80%93Phong_shading_model). Quel est l'avantage de ce modèle ? Vous pouvez observer rapidement la différence avec Phong sans relancer l'application en modifiant le *#define* dans le shader et en rechargent le shader à l'aide de la touche 'r'.

2.3 Problème

Que pensez-vous de la qualité de cette illumination ? Quel est le problème ?

Question 3 : Illumination dans le fragment shader

Bon la solution ne va pas être bien longue à trouver, la voici ;-)

Nous allons transférer le calcul d'illumination dans le fragment-shader (Quel est l'avantage ?). Pour cela copiez simplement la fonction shade pour l'utiliser dans le fragment shader. Vous avez également besoin de passer les vecteurs L, N et V à l'aide de varyings.

Pour le modèle d'illumination ces vecteurs doivent être normalisés, ou doit etre réalisée cette normalisation (vertex ou fragment shader) ?

Question 4 : Cell-Shading

Nous allons maintenant nous amuser un peu avec un shading un peu plus sympa en réalisant un « Toon-Shading » dans un fragment shader. Cet effet se décompose en deux parties :

- La couleur interne des objets est calculée à partir de l'angle formé entre la normale à la surface et le vecteur L (light). On peut prévoir 2 angles par exemple et atténuer ou amplifier la couleur de base de l'objet (passée par le vertex shader) par une constante entre ces angles. Le plus simple pour cela est de réaliser un test sur N°L et d'affecter la couleur en fonction du résultat. Vous pouvez utiliser par exemple couleur foncée pour les grands angles, normale pour les angles intermédiaires et blanc pour les petits angles (lumière à la verticale de la surface). Commencez donc par coder cet effet :-)

- Ce qui manque maintenant c'est une petite bordure noire autour des objets. Cette bordure peut être réalisée à l'aide d'un deuxième rendu (après le rendu normal) des faces arrières des objets en mode « fil de fer » (`glPolygonMode(GL_BACK, GL_LINE)`). Tous les fragments sont ainsi éliminés par le depth-test (que vous devez configurer correctement), sauf ceux en bordure et dépassant du rendu original. Pour que ca fonctionne l'épaisseur des lignes doit etre suffisante (`glLineWidth(4.0)`). Les shaders doivent également etre configurés afin d'appliquer du noir lors de ce rendu.

Question 5 : Traitements en espace image

Comment réaliseriez-vous le type d'effet présenté dans l'image suivante sachant que la position écran d'un fragment peut être récupérée dans la variable gl_FragCoord et que les fonctions trigonométriques sont disponibles dans la librairie standard des shaders ? (Il n'y avait absolument aucun indice dans cette phrase).

Question 5 : Sources multiples

Pour allez plus loin amusez-vous à placer plusieurs sources tournant autour de votre scène.