Université Lille1 Sciences et Technologies

UFR IEEA

M3DS

2012–2013
Master ST-A MS2

Examen 2012-2013
1ere session
Durée : 3 heures
Tout document papier autorisé

Toute réponse doit être **justifiée**. Le barème est donné à titre indicatif.

Exercice 1 : Shader et éclairage *(4 points)*

On considère l'éclairage local de Phong basé sur la réflexion diffuse et la réflexion spéculaire comme vu en cours et tps.

Q 1. On choisit de calculer tout l'éclairage (diffus et spéculaire) soit dans le vertex shader, soit dans le fragment shader : citez les avantages-inconvénients de ces 2 choix (3-4 lignes maxi).

Q 2. On suppose que la normale est normée dans le vertex shader : est-il nécessaire de la renormer dans le fragment shader si on calcule l'éclairage dans le fragment shader ?

Q 3. On propose de calculer la réflexion diffuse dans le vertex shader et la réflexion spéculaire dans le fragment shader : quel est l'avantage ?

Q 4. On dispose d'un shader qui permet de gérer 2 sources lumineuses. Pour le spéculaire, au lieu de calculer $V \cdot R$ avec R le rayon réfléchi du vecteur d'éclairement L, et V le vecteur d'observation, on propose de calculer $L \cdot R'$ avec R' le rayon réfléchi du vecteur d'observation V. Pourquoi ? (schéma rapide bienvenu).

Q 5. Pour que l'éclairage soit toujours correct par rapport à l'observateur, on proposait en tp de diriger la normale vers l'observateur en calculant $N \cdot V$. Une autre façon de faire est de supposer que la normale reçue par le vertex shader correspond toujours à la normale dirigée du coté direct du polygone (la majorité des modèles respecte cette contrainte). Dans le fragment shader, on peut tester si le pixel tracé correspond à un polygone `FRONT` ou un polygone `BACK` (la variable prédéfinie `gl_FrontFacing` est `true` ou `false`). Que suffit-il donc de faire pour avoir un éclairage correct ?

Exercice 2 : Graphes de scène *(3 points)*

On représente explicitement un graphe de scène avec une structure d'arbre. On se contente de pseudo-code simple dans cet exercice.

Un noeud de l'arbre est représenté par le type `Node`. Pour un noeud n donné, $n.parent()$ donne son noeud parent dans la hiérarchie, et $n.matrix()$ donne la matrice homogène qui permet de traduire le changement de repère du noeud père au noeud n (i.e. $n.matrix() = M_{\text{n.parent()}\to\text{n}}$). Le type `Matrix` possède l'opérateur `*` (par ex : `m3=m1*m2`) et `m2=m1.invert()` permet d'obtenir le passage inverse.

Pour le noeud racine on a $n.parent() == VIDE$ et $n.matrix()$ correspond au placement du graphe (i.e. l'ensemble de la scène) par rapport au répère du monde (noté `world`).

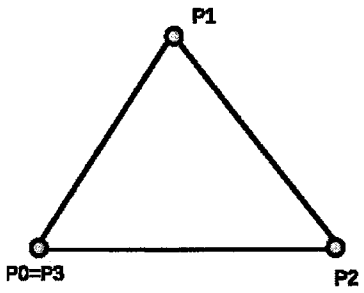
La caméra est placée comme enfant d'un noeud quelconque d'un objet de l'arbre (pour suivre cet objet avec la caméra par exemple).

Q 1. Donnez le pseudo-code de la fonction `Matrix worldToNode(Node n)` qui exprime le passage $M_{\text{world}\to n}$ pour un noeud n quelconque du graphe de scène.

Q 2. En déduire alors le pseudo-code de `Matrix cameraToModel(Node n)` qui exprime le passage $M_{\text{camera}\to n}$. On suppose ici que le noeud de la caméra est accessible par une variable globale `Node camera`.

Q 3. Pour afficher l'objet n par OpenGL il faut fournir la `modelview` : est-ce `cameraToModel(n)` qu'il faut fournir ou l'inverse ?

Exercice 3 : Bézier *(4 points)*



On considère les 4 points de contrôle du schéma (le premier $P0$ étant confondu avec le dernier $P3$).

Q 1. Reprenez à main levée ce schéma et faites apparaitre la construction de De Casteljau pour $t = \frac{3}{4}$ (contentez vous de le faire approximativement et à main levée, à condition que votre schéma soit clair et sans ambiguité).

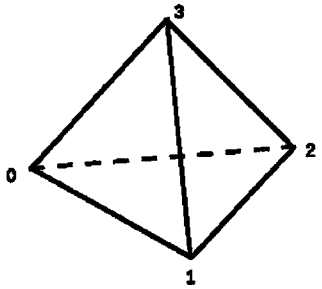
Q 2. Tracez approximativement la forme de la courbe de bézier résultante de ces 4 points de contrôle.

Q 3. On peut considérer les points $P0$ et $P3$ comme un point de raccordement de la courbe de bézier avec elle même : la courbe en ce point est elle de continuité C^0 ? C^1 ? G^1 ?

Q 4. On souhaite à présent interpoler ces 4 points par des béziers cubiques (donc 3 béziers cubiques). Tracez **approximativement** la forme de la courbe résultante en appliquant la méthode de Catmull-Rom (vous ferez apparaitre la direction des tangentes en chacun des sommets).

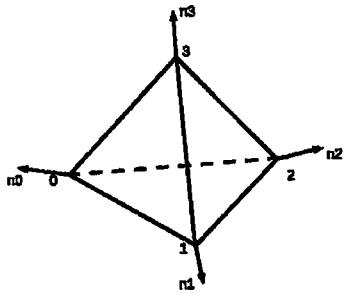
Exercice 4 : Arêtes vives *(6 points)*

Pour cet exercice on s'appuie sur le tétraèdre du schéma.



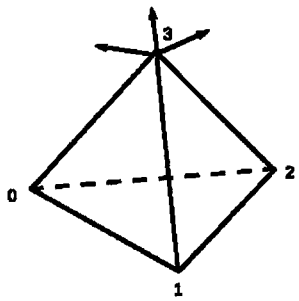
Q 1. En OpenGL, on suppose qu'on a correctement initialisé le buffer (i.e. le tableau) pour les coordonnées des sommets (x_i, y_i, z_i) (dans l'ordre $x_0, y_0, z_0, x_1, y_1, etc, y_4, z_4$). On choisit de tracer ce tétraèdre avec l'instruction `glDrawElements`, avec comme primitive `GL_TRIANGLE_STRIP`. Indiquez alors le tableau d'indice qu'on peut donner à OpenGL (on ne demande aucun code, mais uniquement les valeurs du tableau d'indices, dans le bon ordre, à fournir ; par exemple sous la forme $\{3,8,1,2,8\}$).

Q 2. Pour avoir un tracé avec un éclairage, on fournit également les normales pour chaque sommet (par exemple dans un buffer $n_{0x}, n_{0y}, n_{0z}, n_{1x}, etc$). Ces normales sont calculées par moyenne des normales aux faces adjacentes.



On trace alors le tétraèdre avec le même `glDrawElements` que précédemment. L'éclairage calculé est un éclairage local de Phong. Indiquez alors pourquoi le rendu du tétraèdre n'apparait pas satisfaisant avec ces normales.

Q 3. Une solution pour contrer cet effet est de dissocier les normales selon la face. Sur l'exemple simple du tétraèdre on aura donc 3 normales par sommet (c'est la normale en chacune des faces adjacentes). On traduit ainsi la notion d'arête vive (discontinuité des normales lorsqu'on passe d'une face à sa voisine).



En OpenGL (idem en Direct3D), tous les attributs (coordonnées, normales, coordonnées de texture, ...) d'un sommet donné doivent correspondre à un seul indice de tableau (i.e. un sommet est défini par **tous** ses attributs). Autrement dit, les coordonnées spatiales du sommet i du tétraèdre doivent apparaitre à 3 indices différents pour lui associer 3 normales différentes. Il faut donc dupliquer 3 fois les coordonnées (x_i, y_i, z_i).

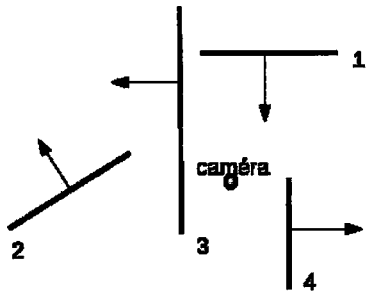
1. Justifiez (en 1 ou 2 lignes) pourquoi le tracé avec un `GL_TRIANGLE_STRIP` n'est alors plus pertinent.
2. En supposant que les coordonnées du sommet 0 sont dupliqués aux indices $\{0,1,2\}$, le sommet 1 aux indices $\{3,4,5\}$, etc, donnez à nouveau une succession des indices pour tracer le tétraèdre avec `glDrawElements`, mais cette fois en `GL_TRIANGLES`. Vous donnerez cette succession de façon à ce que les triangles soient orientés directs vers l'extérieur.

Q 4. Les arêtes vives posent également un problème pour la modélisation des objets. Nous avons par exemple vu, en cours et tp, le modèle des winged-edges sans se préoccuper de ce problème. Lorsqu'on effectue une moyenne des normales en chaque sommet, comme pour les modèles du tp, il suffit de stocker cette normale dans la table des sommets (puisque 1 sommet du modèle = 1 normale). S'il existe des arêtes vives, par contre, il faut pouvoir associer plusieurs normales à un seul sommet.

1. Justifiez (3-4 lignes maxi ; pensez éventuellement à faire un schéma) que le fait de dupliquer tous les sommets n'est pas pertinent sans perdre l'intérêt des winged-edges (par exemple, trouver toutes les faces adjacentes au sommet 0 du tétraèdre).
2. Si le modèle contient uniquement des arêtes vives (par exemple le tétraèdre, le cube, ...), où peut-on stocker la normale ?
3. Dans le cas général (quand il existe des arêtes vives et non vides), proposez un choix pour stocker les normales (répondre en 2-3 lignes maxi). Citez alors un inconvénient de votre choix.

Exercice 5 : Arbres BSP *(3 points)*

On considère une scène dont la schématisation 2D est la suivante. La flêche représente le coté positif de la face.



Q 1. Donnez l'arbre BSP de cette scène : vous prendrez toujours comme pivot l'indice le plus petit (lorsqu'il y a découpe, vous noterez par un '+' ou un '-' ; par exemple 8^{+-} provient de la partie négative de la face 8^{+}, elle même issue de la partie positive de la face 8).

Q 2. Donnez alors l'ordre d'affichage avec la technique du peintre avec la caméra placée sur le schéma.

Q 3. En OpenGL, un pixel qui ne passe pas le test du depth buffer ne met pas à jour sa couleur (et les cartes récentes évitent également d'exécuter le fragment shader sous certaines conditions). Il y a donc un gain de performance en maximisant le nombre de pixels tracés qui ne passent pas le depth buffer : comment exploiter le BSP à cette fin ?