Université Lille1 Sciences et Technologies

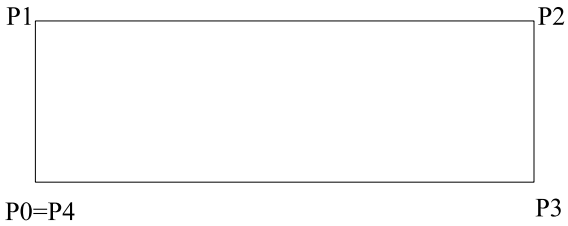
UFR IEEA

M3DS

2011–2012
Master ST-A MS2

Examen 2011-2012
1ere session
Durée : 3 heures
Tout document papier autorisé

Exercice 1 : Bézier *(3 points)*



Q 1. Reprenez le rectangle à main levé (en gardant approximativement les proportions) : il définit la courbe de Bézier $P(t)$ ayant pour points de contrôle $(P_0, P_1, P_2, P_3, P_4)$ (P_4 est confondu avec P_0). Faites apparaitre clairement la construction du point pour $t = \frac{1}{4}$ obtenu avec De Casteljau.

Q 2. Uniquement avec ce point, tracez "grossièrement" (à main levée) la forme de la courbe de Bézier résultante.

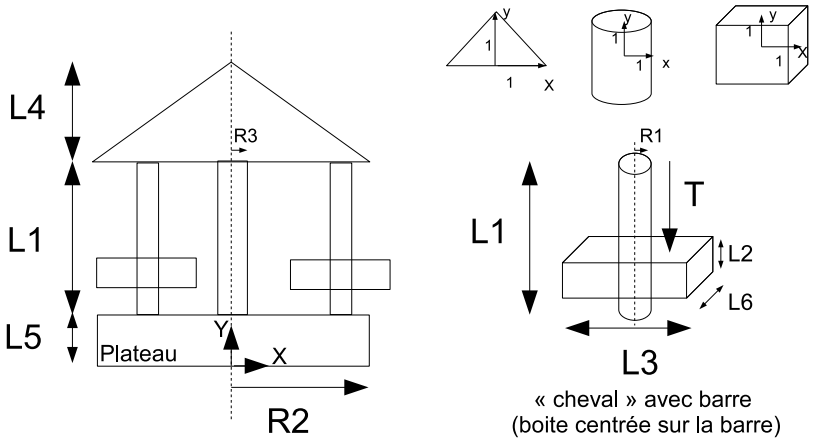
Q 3. Le raccordement au point $P_0 = P_4$ est-il de continuité G^1 ? C^1 ? (justifiez sans faire de calculs).

Exercice 2 : Tracé OpenGL hiérarchique *(3 points)*

On souhaite tracer un manège. Le plateau et le pilier central sont des cylindres. Le toit est un cône. Les chevaux de bois et leur barre de maintien sont simplement représentés par un cube et un cylindre. Il y a 9 chevaux que vous disposerez uniformément sur le contour du manège (vous placerez le centre des barres à l'extrémité du rayon du plateau).

Le manège est en mouvement : l'ensemble, **sauf** le plateau, tourne d'un angle θ. Les « chevaux » sont en translation T le long de leur barre de maintien.

On utilisera les procédures `tracerCone();` `tracerCylindre();` et `tracerCube();` dont les repères naturels sont donnés en schéma (utilisez `glScale` pour respecter les dimensions des éléments du manège). Les rayons, hauteurs et largeurs dont donnés sur le schéma (nom des constantes à utiliser dans votre code).



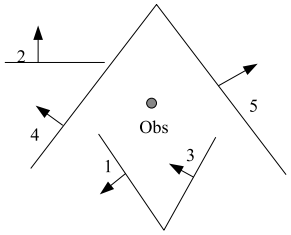
Q 1. Donnez une décomposition hiérarchique du manège (schéma de l'arbre).

Q 2. Donnez le code C/OpenGL `tracerManege(float theta,float T);` qui trace le manège. Le repère de l'ensemble du manège est sur la base du plateau en son centre. Le code doit être extrèmement **clair**. Le respect des dimensions n'est pas important : il sera évalué le respect de l'arbre, ainsi que la cohérence des compositions des transformations avec les `glTranslate, glRotate, glScale, glPushMatrix, glPopMatrix`.

<u>Exercice 3</u> : BSP *(4 points)*

Q 1. Donnez l'arbre BSP de la figure suivante : lorsqu'il y a un choix de facette à faire, vous prendrez **le numéro le plus petit**. La flêche représente le coté positif pour chacune des faces. Une facette décomposée sera indicée par un signe (ex : face 4 découpée en 4+ et 4- ; et si 4+ est redécoupée : 4++ 4+- ; etc).

Vous reprendrez le schéma (à main levée mais en respectant les coupes qui doivent être générées) : vous ferez apparaitre les découpages éventuels.



Q 2. Donnez alors l'ordre d'affichage induit par l'algorithme du peintre sur votre arbre, avec l'observateur placé comme sur le schéma.

Q 3. La structure d'arbre BSP est donnée par le type `Arbre` : si `a` est un `Arbre` alors `a.racine` donne le polygone du noeud, et `a.negatif` et `a.positif` donnent respectivement les sous-arbres (de type `Arbre`) négatifs et positifs par rapport à `a.racine`.

On veut savoir si 2 observateurs, représentés par des points $P1$, $P2$ peuvent se voir l'un, l'autre dans la scène (i.e. il n'y a pas de faces entre eux). Indiquez comment parcourir les faces du BSP pour faire cela de la façon **la plus optimale possible** (on suppose disposer de `signe(p,f)` pour connaitre le signe + ou - du point p par rapport au polygone f, et `intersection(p1,p2,f)` qui indique si oui ou non il y a une intersection entre le segment `[p1,p2]` et le polygone f).

Indication : faire l'étude de cas : `p1` et `p2` tous les deux du coté positifs de `a.racine`, etc.

Exercice 4 : Winged-Edges *(4 points)*

On utilise la structure suivante :
- le type `Sommet` donnant une arête pour un sommet donné `s` par `s.arete` (de type `Arete`).
- le type `Face` donnant une arête pour une face donnée `f` par `f.arete` (de type `Arete`).
- le type `Arete` qui donne pour une arête donnée `a` les champs `a.debut`, `a.fin`, `a.faceGauche`, `a.faceDroite`, `a.prevGauche`, `a.succGauche`, `a.prevDroite`, `a.succDroite`.
- On dispose également de `ListeFace` pour représenter une liste de faces (`l.add(f)` ajoute la face `f` à la fin).

Q 1. Ecrire le pseudo-code `ListeFace adjacent(Face f)` qui donne la liste des faces qui sont adjacentes à la face f donnée (c'est-à-dire toutes les faces qui ont une **arête** en commun avec f). Toute solution qui parcourt plus de faces que nécessaire ne sera pas évaluée.

Q 2. Y a t'il un problème si l'objet possède un bord ?

Q 3. Ecrire le pseudo-code `ListeFace adjacent2(Face f)` qui donne la liste de toutes les faces qui ont un **sommet** en commun avec f (l'objet est supposé sans bord).

Exercice 5 : Eclairement de Phong *(3 points)*

Les questions sont indépendantes (justifiez vos réponses en 2 lignes maximum !).

Q 1. On considère un cube représenté par 6 polygones (les 6 faces du cube).

1. Pourquoi n'est il pas pertinent d'appliquer un éclairement spéculaire en rendu projectif avec interpolation de Gouraud sur cet objet ? (3 lignes max).
2. Pourquoi n'est il pas pertinent ici de faire la moyenne des normales des faces en chaque sommet ?
3. Que faudrait il faire pour avoir un rendu «correct» du spéculaire ? (faites 2 réponses : en gardant Gouraud d'une part, et en proposant une alternative à Gouraud d'autre part).

Q 2. On trace un triangle de 100 pixels à l'écran avec shaders :

1. combien de calculs de composante spéculaire sont effectués avec une interpolation des couleurs ?
2. combien de calculs de composante spéculaire sont effectués avec une interpolation des normales ?

Q 3. On souhaite faire le tracé d'un ensemble de polygones avec des shaders en calculant le spéculaire dans le fragment shader. La scène est constituée de plusieurs sources lumineuses : il faut donc calculer, pour chaque source, le $V \cdot R_i$ (où V est le vecteur d'observation et R_i est le vecteur réfléchi du vecteur d'éclairement L_i de chaque source). On choisit, à la place, de calculer $L_i \cdot R_v$, où R_v est le vecteur réfléchi de V.

1. Sans calcul, et sans démonstration, mais avec un schéma convaincant, indiquez que cela donne le même résultat.
2. Quel est alors l'intérêt ?

Exercice 6 : Arbre CSG *(3 points)*

La première question est totalement indépendante des autres.

Q 1. On considère la structure d'arbre CSG suivante : si a est de type `CSG` alors

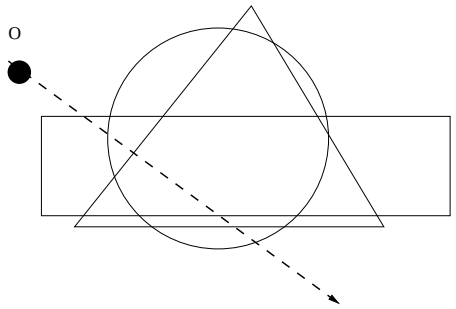
– `a.gauche`, `a.droite` donnent les sous arbres gauche et droite (de type `CSG`),
– `a.operation` donne l'opération (constante `INTERSECTION`, `UNION`, ou `DIFFERENCE`) et
– `a.primitive` donne une référence sur une primitive (de type `Primitive *` ; `a.primitive==NULL` si `a` est un noeud interne).

On dispose également de la fonction `bool in(m:Vector3;Primitive *p)` qui indique si un point *m* est à l'intérieur de la primitive p.

Faire le pseudo-code de `bool in(m:Vector3;CSG a)` qui indique si le point *m* est à l'intérieur de l'arbre CSG `a` (a supposé non vide).

Q 2. On considère les primitives de la figure suivante (en 2D). Le CSG est construit par `G=(cercle DIFFERENCE rectangle) INTERSECTION triangle`.

Shématisez, à main levée, l'objet résultant.



Q 3. En considérant le rayon représenté sur la figure, et en partant de O, on note λ_1^T et λ_2^T les 2 intersections pour le triangle, λ_1^C et λ_2^C les 2 intersections pour le cercle, et λ_1^R et λ_2^R celles pour le rectangle. Quelle sont alors les listes d'intersections en chacun des noeuds du CSG en appliquant l'algorithme vu en Cours-TD/TP ?

Q 4. Pour trouver l'intersection la plus proche de l'observateur, on propose de parcourir l'arbre CSG en appliquant le principe suivant : pour trouver le point le plus proche en un noeud N de l'arbre, on détermine l'intersection I_G la plus proche dans le sous-arbre gauche et l'intersection I_D la plus proche dans le sous-arbre droit, et selon l'opération, on garde soit I_G soit I_D pour le noeud N. Expliquez pourquoi ce principe est incorrect en général (donnez un exemple simple qui met en défaut ce raisonnement).