Université Lille1 Sciences et Technologies

UFR IEEA

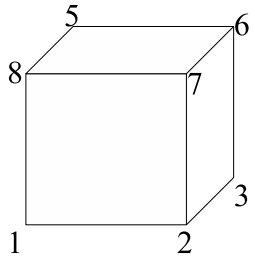
M3DS

2010–2011
Master ST-A MS2

Examen 2010-2011
1ere session
Durée : 3 heures
Tout document papier autorisé

Exercice 1 : Tracé OpenGL *(3 points)*

Q 1.



On souhaite activer l'élimination des faces arrières sur ce cube.

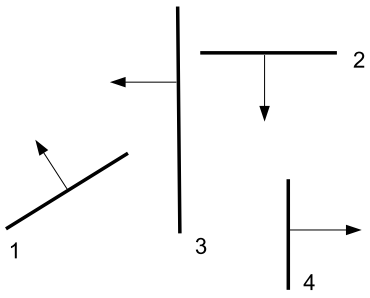
Dans quel ordre doivent être donnés les indices des faces $(5,6,3,4)$ et $(7,2,3,6)$, pour qu'elles soient orientées "correctement" (face directe orientée vers l'extérieur). Le sommet 4 est celui qui n'est pas visible sur le schéma.

Q 2. Dessinez (rapidement à main levée, mais clairement) le tracé obtenu avec l'élimination des faces arrières, si **seule** la face $(7,2,3,6)$ n'est pas orientée correctement.

Q 3. On souhaite tracer un tétraèdre défini par ses 4 sommets $V0, V1, V2, V3$ avec un **unique** `glDrawElements(GL_TRIANGLE_STRIP,...)`. Donnez un tableau d'indices (contenant les valeurs 0,1,2,3) qu'il faudra donner à la mémoire OpenGL pour tracer correctement les 4 faces du tétraèdre (on ne demande aucune instruction OpenGL, mais seulement les valeurs du tableau d'indices).

Exercice 2 : Arbres BSP *(6 points)*

On souhaite traiter la scène 2D suivante en exploitant les arbres BSP.



Q 1. Pour les 4 segments donnés sur la figure, donnez l'arbre BSP obtenu en prenant toujours comme pivot l'indice le plus grand (lorsqu'il y a découpe, vous noterez par un '+' ou un '-' ; par exemple 8^{+-} proviendrait de la partie négative de la face 8^{+}, elle même issue de la partie positive de la face 8).

Q 2. Même question, mais en prenant toujours l'indice le plus petit comme pivot.

Q 3. On dispose d'une scène structurée en BSP, et on veut effectuer un lancer de rayon sur cette scène. Etant donné un rayon primaire (O,u), on cherche le point de la scène à éclairer.

On suppose disponible `IntersectionInfo intersecte(Rayon r,Face f)` qui retourne les informations concernant l'intersection entre r et la face f (on dispose de `inter.lambda()` et `inter==VIDE` pour `inter` de type `IntersectionInfo`).

On veut minimiser le nombre d'appels à cette fonction en exploitant **au mieux** les propriétés de l'arbre BSP.

Proposer alors un pseudo-code pour `IntersectionInfo intersecte(Rayon r,BSP arbre)` qui donne l'intersection souhaitée pour le lancer de rayon (on considère ici que r est un rayon primaire). Vous pourrez utiliser `arbre.negatif()`, `arbre.positif()`, `arbre.face()`, `arbre==VIDE`, `f.signe(p)==PLUS` pour obtenir, respectivement, les deux sous arbres, le noeud de `arbre`, le test de vacuité de l'arbre, et le signe d'un point par rapport à une face).

Q 4. On peut apporter une optimisation supplémentaire en considérant que l'intersection souhaitée est celle dont `inter.lambda()>0` (i.e. celle qui se trouve "devant" l'observateur). Si on suppose que `r.direction()` et `f.normale()` donne le vecteur directeur d'un rayon et la normale **positive** d'une face, proposez une amélioration de votre pseudo-code (inutile de refaire tout le code ; contentez vous d'être clair et convaincant).

Exercice 3 : Collisions *(7 points)*

On anime des rectangles orientables (i.e. des OBB) dans une scène 2D. Pour déterminer si des boites sont en collision, on teste si elles se recouvrent à un l'instant t considéré (i.e. méthode de détection statique).

Q 1. Questions à réponses brèves :

1. Indiquez un inconvénient d'une méthode statique pour la détection des collisions.
2. On applique la méthode des axes séparateurs pour tester si deux OBB sont en recouvrement. Pour conclure si les boites sont en recouvrement ou non : combien de directions au maximum faut il considérer pour ces 2 boites ? combien au minimum ? Indiquez dans quels cas on aura ce maximum et ce minimum.
3. Dans une direction donnée, pour une boite b_1 on obtient l'intervalle $[\lambda_1^{min},\lambda_1^{max}]$ et pour la boite $b2$, $[\lambda_2^{min},\lambda_2^{max}]$. Donnez le test à effectuer pour conclure que la direction n'est pas séparatrice.
4. On suppose que les boites ne sont pas orientables et leurs axes (i.e. leur largeur et leur hauteur) sont toujours alignés avec les axes X et Y du repère de la scène (boites appelées AABB pour "Axis-Aligned Bounding Box"). Quelles sont alors les directions à tester pour conclure que les boites se recouvrent ou non ?

Q 2. On revient aux OBB, mais on considère uniquement les directions des axes X et Y de la scène. Donnez un exemple de 2 OBB b_1 et b_2 (faire un schéma rapide à main levée) avec b_1 et b_2 disjointes mais où les axes X et Y ne sont pas séparateurs. Justifiez alors qu'un algorithme de détection basée uniquement sur les axes X et Y constitueraient une broad-phase.

Q 3. On met en place, une optimisation de la détection de la collision entre n boites, en se basant uniquement sur l'axe X de la scène.

On dispose déjà d'une procédure `narrowPhase(NumeroBoite i,ListeBoite a_tester)` qui effectue la détection de collision entre la boite i et toutes les boites de la liste `a_tester`. L'objectif est d'appeler cette procédure pour chaque boite i de la scène, mais en minimisant le nombre de boites dans `a_tester`.

Pour cela, on considère n boites OBB projetées sur l'axe X, ce qui donne une liste d'intervalle $[x_i^{min}, x_i^{max}]$ (avec $i \in [1,n]$, i identifiant chacune des boites).

On suppose alors **triées**, dans l'ordre croissant, toutes ces valeurs x pour obtenir la liste d'éléments $(e_1, e_2, ..., e_{2n})$. Pour chaque `Element` e de cette liste triée, `e.x` donne la valeur x, `e.boite` donne le numéro de la boite et enfin `e.min` vaut `true` si `e.x` est le minimum (i.e. le "début") de l'intervalle de la boite `e.boite` (et vaut `false` si `e.x` est le maximum).

Donnez le pseudo-code de la procédure `detectCollision(ListeElement triee)` qui parcourt toute la liste d'éléments, et qui exploite **au mieux** le fait que la liste est triée (le problème étant de minimiser le nombre d'appels à `narrowPhase` et de gérer au mieux la liste `a_tester` ; elle est initialement vide).

Vous disposez de :`e=triee.tete()`, `triee=triee.reste()`, `trier.estVide()`, `a_tester.add(e.boite)` pour ajouter une boite, et `a_tester.supp(e.boite)` pour l'enlever.

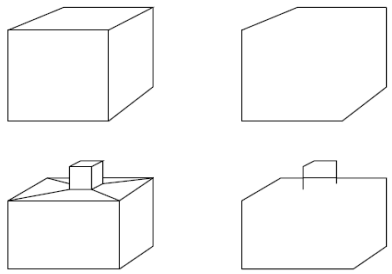
Q 4.

1. Dans quel cas on constate que la liste `a_tester` est toujours vide à l'appel de `narrowPhase` ?
2. Décrire un cas où on se retrouve à tester toutes les paires de boites.

Q 5. Si l'algorithme précédent permet de limiter les paires de boites à tester, il repose cependant sur le fait que la liste des éléments doit être triée. Pour éviter un tri complet, à chaque instant de la simulation, on peut exploiter la cohérence temporelle : on peut supposer que les boites ne bougent pas beaucoup entre 2 instants successifs. Justifiez alors qu'un tri à bulles peut s'avérer pertinent (rédigez en 2-3 lignes maxi).

<u>Exercice 4</u> : Winged-edges *(4 points)*

Dans cet exercice on cherche à déterminer les lignes de contour d'un objet représenté par winged-edges, dont on donne 2 exemples sur la figure. Plusieurs définitions de la notion de "contour" existent. Ici, on considère qu'une arête appartient au contour si elle est commune à une face frontale et une face arrière (i.e. intuitivement c'est la "frontière" entre ce qui peut être vu et ce qui ne peut pas l'être). Cette définition repose sur un objet correctement orienté vers l'extérieur. Le contour d'un objet dépend du point de vue.



On rappelle que la structure de winged-edges (arêtes ailées) est constituée de 3 tables : une table des sommets dont chaque sommet s contient un champ $s.arete$; une table des facettes dont chaque facette f contient un champ $f.arete$; une table d'arête dont chaque arête a contient les champs `debut`, `fin` (sommets), `droite`, `gauche` (les deux faces de l'arête), ainsi que les champs `precGauche`, `succGauche`, `precDroite` et `succDroite`.

Chaque facette f a également un champ `orientVue` qui peut prendre deux valeurs `FRONTALE` ou `ARRIERE`.

Q 1. On connait les coordonnées de chaque sommet `s.point`, la normale de chaque facette `f.normale` et les coordonnées de l'observateur `obs`. Toutes les coordonnées sont connues dans le repère de scène G. Quel est le calcul à faire pour savoir si f est frontale ou arrière par rapport à l'observateur ?

Q 2. On suppose ici qu'une seule ligne de contour existe et est fermée (i.e. en partant d'une arête du contour, et en le suivant arête par arête, on retombe sur l'arête initiale). On suppose de plus qu'on connait déjà une arête du contour.

On souhaite construire toute la ligne polygonale du contour de proche en proche : on part d'une arête e_1 que l'on **sait** appartenant au contour, puis on détermine une arête e_2 du contour, adjacente à e_1, en parcourant uniquement ce qui est **nécessaire** (on réitère ensuite le processus sur e_2).

1. Donnez le pseudo-code pour passer de e_1 à e_2 : `Arete nextContour(Arete e)` (toute solution n'optimisant pas le nombre d'arêtes parcourues ne sera pas évaluée).
2. En déduire le pseudo-code `ListeSommet contour(Arete e)` pour construire toute la ligne de contour dont on connait une arête `e`.

Q 3. Indiquez brièvement quels sont les problèmes si l'objet n'est pas convexe ?