Université de Savoie, UFR SceM — 3 mars 2023
INFO804, M1 INFO — Durée : 1h30

Examen, INFO804, Informatique graphique, Session 1

Documents autorisés : 1 feuille recto-verso.

Les exercices sont indépendants. Le barème est indicatif. Dans tous les exercices, vous pouvez supposer avoir résolu la question précédente. Par ailleurs, vous pouvez utiliser le produit scalaire "$\cdot$" ($\vec{u} \cdot \vec{v}$ en math ou `u.dot(v)` en C++) et le produit vectoriel "$\times$" ($\vec{u} \times \vec{v}$ en math ou `u.cross(v)` en C++) dans tous les exercices. On notera aussi p_x, p_y, p_z les coordonnées d'un point p si nécessaires, pareil pour les composantes des vecteurs. Le barême est indicatif et dépasse volontairement 20 pour que vous puissiez choisir les exercices.

Exercice 1. Quizz (/2)

Les réponses attendues sont un mot, au maximum une ligne.

1. Citez une technologie alternative à OpenGL.
2. Comment s'appelle le langage utilisé par les shaders OpenGL ?
3. Pourquoi le rendu par lancer de rayon peut devenir très coûteux en temps de calcul ?
4. Si on veut utiliser une texture *normal map* pour rendre plus joli le rendu d'un objet, dans quel shader faut-il placer le code calculant la couleur ?
5. Pourquoi parle-t-on d'arbre lorsqu'on parle d'une scène 3D ?
6. Dans le modèle de couleur de Phong, est-ce que la couleur diffuse dépend de la position de l'observateur ?
7. Comment s'appelle l'algorithme, câblé sur les cartes graphiques, qui permet l'affichage aussi rapide de millions de triangles par seconde ?

1. DirectX, WebGL, Metal
2. GLSL
3. car il faut lancer bcp de rayons par pixel rendu et certaines surfaces induisent des réfractions et donc une explosion combinatoire du nombre de rayons
4. dans le fragment/pixel shader
5. car les objets sont placés relativement à un autre objet, donc forme une hiérarchie
6. non, la couleur diffuse ne dépend pas de la position de l'observateur
7. le Z-buffer

Exercice 2. Quels sont les côtés du toit qui voient le soleil ? (/5)

Vous avez construit une maison avec un toit pyramidal, avec les coordonnées suivantes : le sommet S est en $(0, 0, 5)$ et la base du toit est un rectangle avec les points $A = (2, 1, 3)$, $B = (-2, 1, 3)$, $C = (-2, -1, 3)$ et $D = (2, -1, 3)$. Le soleil est dans la direction $\vec{u} = (1, -1, 1)$.

— Quel est le côté du toit qui voit le plus le soleil ?
— Est-ce que tous les pans du toit voient le soleil ?

Justifiez bien sûr vos résultats avec les calculs appropriés. Faire un schéma peut aider (vue de haut conseillée).

$\vec{AB} = (-4, 0, 0)$	$\vec{BC} = (0, -2, 0)$	$\vec{CD} = (4, 0, 0)$	$\vec{DA} = (0, 2, 0)$
$\vec{AS} = (-2, -1, 2)$	$\vec{BS} = (2, -1, 2)$	$\vec{CS} = (2, 1, 2)$	$\vec{DS} = (-2, 1, 2)$
$\vec{n}_{ABS} = (0, 8, 4)/(4\sqrt{5})$	$\vec{n}_{BCS} = (-4, 0, 4)/(4\sqrt{2})$	$\vec{n}_{CDS} = (0, -8, 4)/(4\sqrt{5})$	$\vec{n}_{DAS} = (4, 0, 4)/(4\sqrt{2})$
$\vec{n}_{ABS} \cdot \vec{u} = -1/\sqrt{5}$	$\vec{n}_{BCS} \cdot \vec{u} = 0$	$\vec{n}_{CDS} \cdot \vec{u} = 3/\sqrt{5} \approx 1.34$	$\vec{n}_{DAS} \cdot \vec{u} = 2/\sqrt{2} \approx 1.41$

Il faut diviser par la norme des vecteurs normaux. C'est donc le pan DAS le plus ensoleillé.
Les plans CDS et DAS voient le soleil tandis que le soleil rase le plan BCS. Le plan ABS est à l'ombre.

Note : $\sqrt{2} \approx 1.41, \sqrt{3} \approx 1.73, \sqrt{5} \approx 2.24, \sqrt{6} \approx 2.45, \sqrt{7} \approx 2.6, \sqrt{8} \approx 2.83$.

Exercice 3. Rayon laser (/4)

Au commande de votre vaisseau en position p, vous vous apprêtez à tirer avec un rayon laser dans la direction $\vec{w}$. Votre objectif est un véhicule située en position a placé sur le sol, qui est relativement plat dans ce coin et de normale $\vec{n}$. Votre rayon laser fait une explosion au point d'impact où tout est détruit dans un rayon r. Ecrivez une fonction CibleDétruiteAuSol, qui retourne vrai si et seulement si la cible a est détruite.

CibleDétruiteAuSol(E a : Point, E $\vec{n}$: Vecteur, E $(p, \vec{w})$: Rayon, E r : Réel) : booléen

PS : Il n'est pas demandé de simplifier les expressions, juste de faire la suite de calcul qui permet de conclure.

```
// On cherche le point d'impact q = p + tw. Il est  sur le plan  (a,n).
// (q-a).n = 0 <=> (tw+p-a).n = 0 <=> t = (a-p).n / w.n
t <- (a-p).n / w.n
Si t < 0 Alors retourner Faux; // On tire du mauvais côté  !
q <- p + t w
Retourner (a-q).(a-q) <= r*r
```

Exercice 4. Interpolation de couleurs (/4)

Soit un triangle ABC dans le plan, avec $A = (0,0), B = (3,1), C = (-1,5)$. Le point A est en rouge, codé en RGB $(1,0,0)$, le point B est jaune $(1,1,0)$, tandis que le point C est bleu $(0,0,1)$. On suppose que les couleurs sont interpolées linéairement dans le triangle ABC.

— (/1) Calculez la couleur au milieu de A et B. Visuellement, c'est quelle couleur ?

C'est simplement la moyenne du rouge et du jaune, donc (1,0.5,0), soit du orange.

— (/1) On se place au point D de coordonnées barycentriques $(\frac{1}{6}, \frac{1}{3}, \frac{1}{2})$. Calculez la couleur de D.

$1/6(1,0,0) + 1/3(1,1,0) + 1/2(0,0,1) = (1/2, 1/3, 1/2)$

— (/2) On se place maintenant au point M de coordonnées $(0,2)$. Calculez sa couleur.

$$
\begin{aligned}
2 * \text{Aire}(ABC) &= \det((3,1),(-1,5)) = 16 \\
2 * \text{Aire}(MBC) &= \det((3,-1),(-1,3)) = 8 && \Rightarrow \alpha = 8/16 = 1/2 \\
2 * \text{Aire}(MCA) &= \det((-1,3),(0,-2)) = 2 && \Rightarrow \beta = 2/16 = 1/8 \\
2 * \text{Aire}(MAB) &= \det((0,-2),(3,-1)) = 6 && \Rightarrow \beta = 6/16 = 3/8
\end{aligned}
$$

On verifie $\alpha + \beta + \gamma = 1$.
La couleur est $4/8(1,0,0) + 1/8(1,1,0) + 3/8(0,0,1) = (5/8, 1/8, 3/8)$.

Exercice 5. Tracé de courbes de Bézier (/5)

On se donne une courbe de Bézier $P(t)$ quelconque, $t \in [0,1]$. On pourrait tracer la courbe avec une couleur c avec une boucle de la forme :

```
1 Fonction TraceBézierNaïf(E P : Courbe de Bézier, E c : Couleur);
2 t ← 0 ;
3 Tant Que t ≤ 1 Faire
4 |   p ← Arrondi(P(t)) ;
5 |   TracePixel(p);
6 |   t ← t + Δt;
```

sachant que l'on dispose des fonctions suivantes :

— Arrondi(E p : Point) qui retourne le point de coordonnées entières le plus proche de p.

— TracePixel(E p : Point, E c : Couleur) qui trace le pixel donné p de la couleur choisie c.

— Distance(E p, q : Point) et qui calcule la distance maximum en abscisse et en ordonnée entre p et q.

Le problème est que l'on ne sait pas trop choisir quel Δt : trop grand et il peut y avoir des trous entre les points, trop petit et on retrace inutilement plein de fois tous les points.

Proposez une fonction qui réalise ce tracé avec une complexité proportionnel au nombre de pixels traversé par la courbe de Bézier.

NB : Vous supposerez que vous pouvez écrire $P(t)$ pour avoir le point correspondant sur la courbe.

(Bonus) Vous pouvez plutôt écrire la fonction qui fait ce tracé avec une couleur interpolée entre la couleur c_0 en $P(0)$ et c_1 en $P(1)$.

Le plus simple est de faire une fonction récursive, qui prend en entrée deux t (un gauche, un droit), qui trace un nouveau point si nécessaire et se rappelle dans ce cas à gauche et à droite. Lorsque 2 points sont à distance moins de (1,1), on s'arrête.

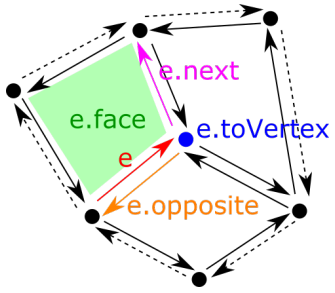
```
1  Fonction TraceBézierRec(E P : Courbe de Bézier, E c0, c1 : Couleur, E g, d : Réel);
   Var : Q, R : Point ;
2  début
3  |  Q ← Arrondi(P(g));
4  |  R ← Arrondi(P(d));
5  |  if Distance(Q, R)) ≤ 1 then
6  |  |  TracePixel(Q, (1 − g)c0 + gc1);
7  |  else
8  |  |  TraceBézierRec(P, c0, c1, g, (g + d)/2.0);
9  |  |  TraceBézierRec(P, c0, c1, (g + d)/2.0, d);
10 |  end
11 fin
12 Fonction TraceBézier(E P : Courbe de Bézier, E c0, c1 : Couleur);
13 début
14 |  TraceBézierRec(P, c0, c1, 0, 1);
15 fin
```

Exercice 6. Structure demi-arête (/5)

On rappelle que les structures demi-arête (ou *half-edge data structure*) permette de représenter des surfaces constituées de faces (souvent planes comme des triangles, des quadrangles, etc) collées entre elles par des arêtes, avec éventuellement des bords.

Comme illustré sur la figure, l'idée principale est de diviser chaque arête en deux demi-arêtes (chacune représente une orientation différente de l'arête). Ensuite, chaque demi-arête connait la demi-arête suivante, la demi-arête opposée, le sommet pointé et la face adjacente. Pour éviter les pointeurs, on modélise tout à l'aide de tableaux et d'indices dans des tableaux, dont voilà une écriture possible :

```
using namespace std;
struct HalfEdgeSurface {
  typedef int Index; // les indices
  vector<Index> opposite; // demi-arête opposée
  vector<Index> next; // demi-arête suivante sur sa face
  vector<Index> face; // face adjacente
  vector<Index> toVertex; // sommet pointé
  vector<Index> face2he; // une demi-arête touchant la face
  vector<Index> vertex2he; // une demi-arête pointant le
      sommet
  vector<Point> positions; // positions des sommets
};
```



On va écrire des méthodes de cette classe HalfEdgeSurface, on a donc accès aux données membres ci-dessus.

(/1) Ecrivez la méthode **double** longueur(Index he) qui calcule la longueur de l'arête spécifié par la demi-arête he.

```
double HalfEdgeSurface::longueur( Index he ) const
{
  Point p1 = positions[ toVertex[ he ] ];
  Point p2 = positions[ toVertex[ opposite[ he ] ] ];
  return sqrt( (p2 - p1).dot( p2 - p1) );
}
```

(/2,5) Ecrivez la méthode **double** aire(Face f) qui calcule l'aire de la face f. On suppose que la face donnée est bien planaire et on rappelle que le produit vectoriel est utile pour calculer une aire.

```
double HalfEdgeSurface::aire( Face f ) const
{
  std::vector< Point > P;
  Index he = face2he[ f ];
  Index hc = he;
  do {
    P.push_back( positions[ toVertex[ hc ] ] );
    hc = next[ hc ];
  } while ( hc != he );
  double A = 0.0;
  for ( auto i = 2; i < P.size(); i++ )
  A += ( P[ i-1 ] - P[ 0 ] ).cross( P[ i ] - P[ 0 ] ).norm();
  return A / 2.0;
}
```

(/1,5) Ecrivez enfin la méthode bool sommetSurLeBord(Index v) qui retourne vrai si et seulement si le sommet d'indice v est sur le bord de la suurface, c'est-à-dire qu'il touche la face infinie numérotée -1.

```
bool sommetSurLeBord( Index v ) const
{
  Index he = vertex2he[ v ]; // demi-arete pointant vers v
  Index hc = he;
  do {
    if ( face[ hc ] == -1 ) return true;
    hc = opposite[ next[ hc ] ];
  } while ( hc != he );
  return false; // aucune half-edge touche le bord
}
```