20 avril 2015

Master Informatique

Programation Graphique Haute-Performance

Documents non autorisés - Durée 1h30

Vos réponses doivent être claires et concises. L'examen est long mais noté sur 24, il n'est donc pas indispensable de répondre à toutes les questions.

A Transformations de l'espace [4pts]

Dans cet exercice nous supposerons que nous avons accès à une fonction *rot(θ,***a***)* qui retourne une matrice de rotation d'angle θ autour de l'axe a, une fonction *trans(***t***)* qui retourne une matrice effectuant une translation d'un vecteur $\mathbf{t}=[t_x, t_y, t_z]^T$, et une fonction *scale(sx,sy,sz)* qui retourne une matrice effectuant une mise à l'échelle non uniforme. Ces trois fonctions retournent des matrices 4x4.

A.1 Préliminaires.

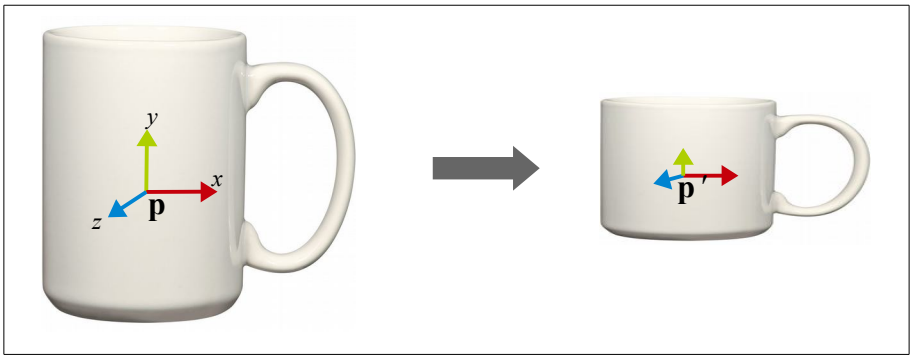
A.1.1 Donner la matrice retournée par *trans(***t***)*.

A.1.2 Donner la matrice retournée par $rot\left(\theta, [0,0,1]^T\right)$

A.2 Comme illustré ci-dessous, nous souhaitons déplacer notre objet de la position **p** à la position **p'** tout en appliquant une mise à l'échelle non-uniforme de manière à ce que l'objet soit deux fois moins haut.

A.2.1 Donner la matrice de transformation à appliquer aux coordonnées des sommets.

A.2.2 Pour un sommet donné, expliquer comment calculer sa nouvelle normale **n'** à partir de sa normale initiale **n**.



B Rendu temps-réel / OpenGL [14 pts]

B.1 Rastérisation [9.5 pts]

B.1.1 Donner le modèle d'éclairement de Blinn-Phong en expliquant chacun des termes. [1.5]

B.1.2 Expliquer comment représenter et visualiser un objet dont la brillance varie à sa surface. [1 pt]

B.1.3 Ombres portées – Pour chacune des situations ci-dessous, donner la méthode de rendu des ombres portées la plus adaptée (justifier chaque choix en quelques mots) : [2 pts]

1. Scène quelconque avec plusieurs sources omnidirectionnelles.
2. Scène composée de nombreux arbres feuillus.
3. Scène quelconque, mais souhait d'une qualité maximale.
4. Simulateur de peinture sur une toile virtuelle (la scène contient un bras, un pinceau, une palette et une toile).

B.1.4 Rappeler le principe des cartes de normales (*normal-mapping*). Quel est le principal intérêt de cette méthode ? [1 pt]

B.1.5 Afin d'éviter les problèmes d'aliasing lorsque de nombreux texels ce projettent sur un seul pixel, peut-on utiliser le mip-mapping pour préfiltrer une carte de normales ? Justifier. [1 pt]

B.1.6 Voici un exemple de shader effectuant le rendu d'un objet blanc, purement diffus pour une source lumineuse ponctuelle. Il y a au moins deux erreurs au niveau du calcul de l'éclairage. Quelles sont elles ? [1 pt]

```
/* vertex shader */
in vec3 vtx_pos;
in vec3 vtx_normal;
out vec3 normal;
out vec3 pos;
uniform mat4 Mobj,  // position/orientation
                    // de l'objet
             Mcam,  // transf. du repère monde
                    // au repère de la caméra
             Mproj; // matrice de projection
void main() {
  gl_Position = Mproj*Mcam*Mobj*vec4(vtx_pos,1);
  pos = vtx_pos;
  normal = vtx_normal;
}
```

```
/* fragment shader */
in vec3 pos;
in vec3 normal;
uniform vec3 light_pos; // position de la caméra
                        // dans le repère monde

void main() {
  vec3 light_dir = normalize(light_pos - pos);
  float d = max(0,dot(normal,light_dir));
  gl_FragColor = vec4(d,d,d,1);
}
```

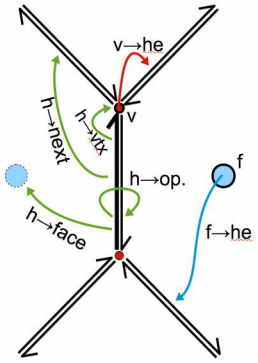
B.1.7 Voici ci-dessous un exemple des structures de données mises en œuvre pour représenter un maillage polygonal par *half-edge*. Nous supposons que les normales des faces ont déjà été calculées. Donner l'algorithme permettant de calculer la normale du sommet numéro i obtenue par la moyenne des normales des faces adjacentes. [2 pts]

```
struct Vertex {
  HalfEdge    *he;
  vec3        pos;
  vec3        normal;
} ;

struct HalfEdge {
  HalfEdge    *next ;
  HalfEdge    *opposite ;
  Face        *face ;
  Vertex      *vertex;
} ;
```

```
struct Face {
  HalfEdge *he;
  vec3     normal;
} ;

struct Mesh {
  vector<HalfEdge> halfedges ;
  vector<Vertex>    vertices;
  vector<Face>      faces;
} ;
```



B.2 Deferred Shading [4.5 pts]

Nous souhaitons mettre en œuvre un moteur de rendu utilisant la technique du *deferred shading* pour des scènes contenant des objets variés : mat, brillant, texturés, etc.

B.2.1 Quelles sont les informations qui doivent être stockées dans le G-buffer ?

B.2.2 Dans quelles conditions cette approche par *deferred shading* peut-elle être plus rapide qu'un rendu direct ?

B.2.3 Notre scène est composée d'une vingtaine de spots, expliquer comment rendre cette méthode de rendu la plus efficace possible.

C CUDA [6 pts]

C.1.1 Donner deux différences fondamentales entre l'architecture d'un CPU et d'un GPU. [1.5pts]

C.1.2 Une opération flottante est plus de 100 fois plus rapide qu'un accès mémoire. Quelle est la stratégie mise en oeuvre par les GPUs pour masquer les temps d'accès mémoire ? [1 pt]

C.1.3 Voici ci-dessous un exemple de code CUDA appliquant un filtre de convolution 3x3 (simple moyenne) à une image en niveaux de gris. Expliquer pourquoi ce code produit un résultat incorrect, et pourquoi il n'est pas optimal d'un point de vue des performances. [1.5 pt]

```
 1 : __GLOBAL__ void kernel(float *img, int width, int height) {
 2 :  int i = threadIdx.x + blockIdx.x * blockDim.x;
 3 :  int j = threadIdx.y + blockIdx.y * blockDim.y;
 4 :  if(i>0 && j>0 && i+1<width && j+1<height)
 5 :    img[i+j*width] = 1./9.*(  img[i-1+(j-1)*width] + img[i+(j-1)*width] + img[i+1+(j-1)*width]
 6 :                            + img[i-1+(j  )*width] + img[i+(j  )*width] + img[i+1+(j  )*width]
 7 :                            + img[i-1+(j+1)*width] + img[i+(j+1)*width] + img[i+1+(j+1)*width]);
 8 :}

 9 : void apply_convolution(float *img, int width, int height) {
10 :   float *d_img;
11 :   cudaMalloc(&d_img, sizeof(float)*width*height);
12 :   cudaMemcpy(d_img, img, sizeof(float)*width*height, cudaMemcpyHostToDevice);
13 :   dim3 DimBlock(16,16);
14 :   dim3 DimGrid((width+DimBlock.x-1)/DimBlock.x, (height+DimBlock.y-1)/DimBlock.y);
15 :   kernel<<< DimGrid, DimBlock >>>(d_img, width, height);
16 :   cudaMemcpy(img, d_img, sizeof(float)*width*height, cudaMemcpyDeviceToHost);
17 :   cudaFree(d_img);
18 : }
```

C.1.4 Nous souhaitons mettre en œuvre en CUDA un rendu de maillages triangulaires par rastérisation. Pour cela, nous proposons le parallélisme suivant : chaque thread CUDA prend en entrée un triangle du maillage et remplit les pixels respectifs de l'image de destination. Discuter des limitations et des difficultés rencontrées par une telle approche. [2 pts]