CPE LYON

18/20

Année universitaire 2015/2016

NOM : MAZOYER **PRENOM :** Thomas

Consignes relatives au déroulement de l'épreuve

A remplir obligatoirement par l'enseignant responsable du contrôle

Date : Vendredi 21 Janvier 2016

Contrôle de : Traitement et Synthèse d'Image (partie Synthèse d'Image)

Durée totale : 2h

Professeurs responsables : D. ROHMER

Documents : ☒ **autorisés** ☐ **non autorisés**
Si oui : type(s) de documents autorisés : Feuille A4 recto/verso

Calculatrices : ☒ **autorisées** ☐ **non autorisées**
Si oui : type(s) de calculatrices autorisées : numériques uniquement

LES TELEPHONES PORTABLES ET AUTRES APPAREILS DE STOCKAGE DE DONNEES NUMERIQUES NE SONT PAS AUTORISES.

Les téléphones portables doivent être éteints pendant toute la durée de l'épreuve et rangés dans les cartables.

S'agissant de contrôle sans document, les trousses doivent être rangées dans les cartables.

Les cartables doivent être fermés et posés au sol.

Les oreilles des candidats doivent être dégagées.

Rappels importants sur la discipline lors des examens

La présence à tous les examens est strictement obligatoire ; tout élève présent à une épreuve doit rendre une copie, même blanche, portant son nom, son prénom et la nature de l'épreuve.
Une absence non justifiée à un examen invalide automatiquement le module concerné.
Toute suspicion sur la régularité et le caractère équitable d'une épreuve est signalée à la direction des études qui pourra décider l'annulation de l'épreuve; tous les élèves concernés par l'épreuve sont alors convoqués à une épreuve de remplacement à une date fixée par le responsable d'année.
Toute fraude ou tentative de fraude est portée à la connaissance de la direction des études qui pourra réunir le Conseil de Discipline. Les sanctions prises peuvent aller jusqu'à l'exclusion définitive du (des) élève(s) mis en cause.

NOM : MAZOYER

PRENOM : Thomas

Examen [TSI] - 4ETI
Partie Synthèse d'Images - OpenGL

CPE Lyon

2015-2016 (1ere session)

Une feuille A4 recto-verso manuscrite autorisée. Tout autre document interdit. Calculatrice numérique autorisée.

Répondez aux questions directement sur l'énoncé

Le sujet comporte 6 pages

Le temps approximatif ainsi que le barème sont indiqués pour les grandes parties. Notez que le barème est donné à titre purement indicatif et pourra être adapté par la suite.

En cas de doute sur la compréhension de l'énoncé, explicitez ce que vous comprenez et poursuivez l'exercice dans cette logique.

Question 1 *Assurez-vous d'avoir écrit votre nom et prénom sur la première page.*

1- Question de cours

- 15 min, 7 points -

Question 2 *Expliquez succinctement ce qu'est un Vertex Buffer Object (VBO) ainsi que son rôle.*

Un VBO est un buffer (donc un tableau) regroupant des données qu'on envoie ensuite à la carte graphique. Ils servent par exemple à donner à la CG (carte graphique) les coordonnées des sommets d'un triangle (ou leurs indices si on veut plusieurs triangles).

2/5

Question 3 *Quel est le rôle et le fonctionnement des variables qualifiées de* **varying** *dans le vertex et le fragment shader.*

Les variables varying permettent de faire passer des variables du vertex shader vers le fragment shader.
La CG réalise automatiquement une interpolation barycentrique linéaire pour ces variables (comme elles sont définies pour un sommet dans le vertex shader, il faut les avoir pour tous les pixels dans le fragment shader).

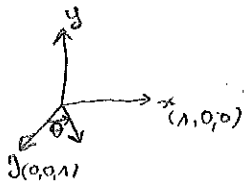
(4)

2- Déplacement d'objets

- 5 min, 3 points -

Un objet 3D de taille approximativement unitaire est initialement orienté suivant la direction donnée par le vecteur $(x, y, z) = (0, 0, 1)$. On applique une rotation à cet objet autour de l'axe y. Lorsque la rotation est de $\pi/2$, l'objet se retrouve en direction du vecteur $(1, 0, 0)$.
Similairement au cas étudié en TP, on souhaite que l'objet puisse se translater dans la direction désignée par son orientation pour une rotation quelconque d'angle θ.

Question 4 *Donnez le vecteur de translation que vous considérerez en fonction de l'angle θ.*



On définit le vecteur de translation :

$$t_i = \begin{pmatrix} \sin\theta \\ 0 \\ \cos\theta \end{pmatrix}$$

(3)

3- Analyse de code

- 20 min, 10 points -

Soit deux fichiers correspondants aux codes suivants:

Fichier 1

```
#version 120

varying vec4 color;
void main (void){
    color  = gl_Color;
    vec4 t = vec4(gl_Normal.z,0.0,0.0,0.0);
    gl_Position = gl_Vertex - 0.5*t;
}
```

Fichier 2

```
#version 120

varying vec4 color;
void main (void){
    gl_FragColor = color;
}
```

Question 5 *Quel fichier correspond au Vertex Shader et lequel correspond au Fragment Shader.*

Vertex Shader : Fichier 1

Fragment Shader : Fichier 2

Ces fichiers sont associés à un code C utilisant la bibliothèque de gestion de fenêtre et d'événements GLUT de manière similaire aux TP. Ce code contient une fonction d'initialisation des données appelée une fois en début de programme, et une fonction d'affichage appelée après la fonction d'initialisation.

Les deux fonctions sont les suivantes

```
GLuint shader_program_id;

GLuint vbo  = 0;
GLuint vboi = 0;

//Fonction d'initialisation
static void init()
{
    float data[] = {0,1,0 , 0,0,0.5 ,
                    0,0,1 , 0,0.5,0 ,
                    1,0,0 , 0.5,0,0 ,
                    0,0,0 , 0.5,0.5,0 ,
                    0,0,1 , 0,0,1 , 0,0,1 , 0,0,1};

    int index[]  = {0,1,3 , 0,3,2};

    glGenBuffers(1,&vbo);
    glBindBuffer(GL_ARRAY_BUFFER,vbo);
    glBufferData(GL_ARRAY_BUFFER,sizeof(data),data,GL_STATIC_DRAW);
```

```
    glGenBuffers(1,&vboi);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,vboi);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER,sizeof(index),index,
                                                    GL_STATIC_DRAW);

    // Chargement du shader
    shader_program_id = read_shader("shader.vert", "shader.frag");

    //activation de la gestion de la profondeur
    glEnable(GL_DEPTH_TEST); PRINT_OPENGL_ERROR();

}

//Fonction d'affichage
static void display_callback()
{
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    int offset = 3*sizeof(float);
    glBindBuffer(GL_ARRAY_BUFFER,vbo);

    glEnableClientState(GL_VERTEX_ARRAY);
    glEnableClientState(GL_COLOR_ARRAY );
    glEnableClientState(GL_NORMAL_ARRAY);

    glVertexPointer( 3, GL_FLOAT, 2*offset, (void *)( offset) );
    glColorPointer ( 3, GL_FLOAT, 2*offset,                0    );
    glNormalPointer(    GL_FLOAT,      0   , (void *)(8*offset) );

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,vboi);
    glDrawElements(GL_TRIANGLES, 2*3, GL_UNSIGNED_INT, 0);

    glutSwapBuffers();
}
```

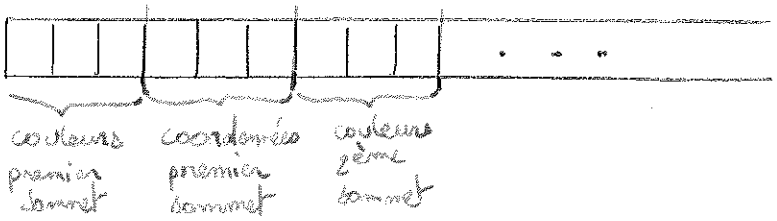
À titre d'information, la documentation des fonctions `glVertexPointer`, `glColorPointer`, et `glNormalPointer` est fournie en annexe en fin de cet énoncé.

Question 6 *Expliquez le plus précisément possible ce que vous observez à l'écran en expliquant votre analyse.*

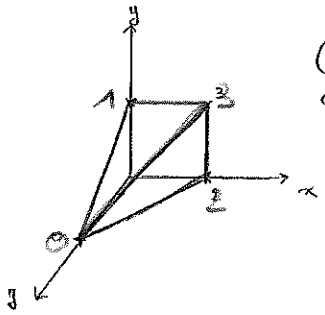
On voit que la fonction d'initialisation passe deux buffers à la CG, un avec 36 floats (6 couleurs (vec3) et 6 sommets (vec3)) et un buffer d'indices (entiers), à priori pour définir deux triangles.

La fonction d'affichage définit que pour les sommets, le vertex shader doit commencer à partir du deuxième sommet (offset de 3 floats) et sauter deux sommets entre chaque sommet (2* offset = 6 floats = 2 sommets). Pour la couleur on enlève l'offset.

Finalement le vbo est :



On obtient à la fin deux triangles :



(couleurs non représentatives, c'est juste pour séparer les 2 triangles).

6.5

Pour les couleurs, le sommet 0 est vert, le 1 bleu, le 2 rouge et le 3 noir (RGB 0, 0, 0 = noir).

Pour le reste des pixels la couleur est définie par un dégradé barycentrique entre les couleurs des sommets. (plus on est proche du sommet 1, et plus le pixel sera bleu)

4- Documentation annexe

4.1 Documentation de glVertexPointer

Name
glVertexPointer — define an array of vertex data

C Specification

```
void glVertexPointer(GLint  size,
                     GLenum  type,
                     GLsizei  stride,
                     const GLvoid *  pointer);
```

Parameters

- **Size**: Specifies the number of coordinates per vertex. Must be 2, 3, or 4. The initial value is 4.
- **Type**: Specifies the data type of each coordinate in the array.
- **Stride**: Specifies the byte offset between consecutive vertices. If stride is 0, the vertices are understood to be tightly packed in the array. The initial value is 0.
- **Pointer**: Specifies a pointer to the first coordinate of the first vertex in the array. The initial value is 0.

4.2 Documentation de glColorPointer

Name
glColorPointer — define an array of colors

C Specification

```
void glColorPointer(GLint  size,
                    GLenum  type,
                    GLsizei  stride,
                    const GLvoid *  pointer);
```

4.3 Documentation de glNormalPointer

Name
glNormalPointer — define an array of normals

C Specification

```
void glColorPointer(GLenum  type,
                    GLsizei  stride,
                    const GLvoid *  pointer);
```