Nom:

Prénom:

TSI
Synthèse d'images

Aucun document autorisé
Calculatrices numériques autorisées
Répondre directement sur l'énoncé d'examen

Question 1.

- Définissez en quelques mots ce qu'est **GLUT**, et ce qu'est **OpenGL**.

(cours)
OpenGL est un ensemble de spécifications sous forme d'une API de programmation permettant de communiquer avec la carte graphique. L'implémentation d'OpenGL dépend du système (Mesa, implémentation NVIDIA, ATI, embarquée, etc).

GLUT est une bibliothèque de gestion de fenêtre et d'évènements (clic souris, touche clavier, etc).

Rem.
OpenGL n'est pas un logiciel, ce n'est pas un modeleur.
GLUT n'est pas l'outil d'OpenG pour gérer la 3D.

Question 2.

- En **GLSL**, expliquez la différence entre une variable qualifiée de ***uniform*** par rapport à une variable qualifiée de ***varying***.

Une variable **uniform** possède la même valeur pour l'ensemble des shaders (pour une image donnée). On peut utiliser une variable qualifiée de uniform pour transmettre une variable du programme CPU vers les shaders (ex. translation, orientation, temps courant, etc).

Une variable **varying** peut posséder une valeur différente dans chaque shader. Elle permet de faire communiquer une variable du vertex shader vers le fragment shader. La valeur obtenue dans le fragment shader correspond à l'interpolation linéaire (fonctionnement par défaut) des valeurs données dans le vertex shader.

Rem. Une variable varying n'est pas une variable qui "varie" pas au cours du temps.

Question 3.

Considérons le code C suivant:

```
GLuint vbo=0;
float T[]={0,0,0 , 1,0,0 , 0,1,0 , 0,0,1};

glGenBuffers(1,&vbo);
glBindBuffer(GL_ARRAY_BUFFER,vbo);
glBufferData(GL_ARRAY_BUFFER,12*sizeof(float),T,GL_STATIC_DRAW);
```

- **Expliquez** précisément ce que **réalise** ce code (en particulier par rapport à la **carte graphique**). Ce code **affiche-il** quelque chose à l'écran, si oui qu'affiche t-il?

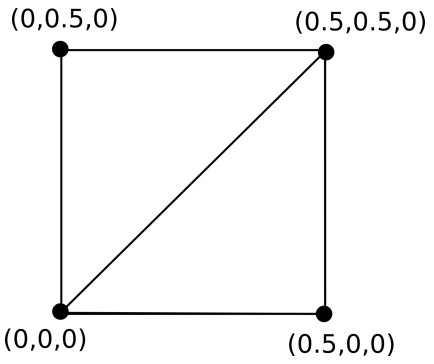
(voir TP)
Ligne 1. Déclaration d'un identifiant entier non signé. On s'en servira pour désigner un VBO.
Ligne 2. Déclaration de 12 flottants contigus en mémoire RAM. Ils définissent probablement 4 coordonnées 3D.
Ligne 3. Demande de création d'un VBO sur le GPU. Son identifiant est stocké dans la variable vbo.
Ligne 4. Définition du VBO courant. Ici, il s'agit du VBO créé juste avant désigné par la variable vbo.
Ligne 5. Transfert de l'ensemble du tableau T du CPU vers le GPU. On lit bien les 12 valeurs du tableau. Ce tableau sera accessible par le VBO désigné par la variable vbo.

Ce code ne fait que le transfert de données vers la mémoire de la carte graphique. Il n'affiche rien. Ce code à sa place dans une fonction d'initialisation et non dans une boucle d'affichage.

Rem. Déclarer une variable GLuint au nom de vbo ne définit pas un VBO sur la carte graphique.

Question 4.

On affiche deux triangles à l'aide d'appels OpenGL. Ces deux triangles mis bout à bout forment un carré. Les coordonnées des sommets sont indiquées sur la figure suivante



On suppose que l'on place une caméra en face de ces triangles. On utilise les shaders suivants lors de l'affichage de ces deux triangles:

Vertex Shader

```
#version 120

varying vec4 p3d;

void main (void)
{
    gl_Position = ftransform();

    p3d=gl_Vertex;
    p3d.x *= 2.0f;
    p3d.y *= 2.0f;
}
```

Fragment Shader

```
#version 120

varying vec4 p3d;

void main (void)
{
    float x=p3d.x; float y=p3d.y; float z=p3d.z;
    float r=0.0f; float g=0.0f; float b=0.0f;
    if(abs(x-0.5f)<0.25f && abs(y-0.5)<0.25f)
    {
        r=1.0f; g=1.0f;
    }
    else
    {
        r=x; g=y; b=z;
    }
    gl_FragColor = vec4(r,g,b,1.0f);
}
```

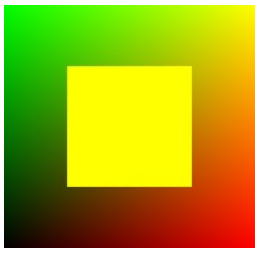
***Note**: Un nombre à virgule terminant par "f" indique un nombre flottant simple précision.*

- Indiquez ce que l'on **observe** sur l'écran. En particulier, décrivez précisément l'**action des shaders** sur le résultat visuel. Donnez le plus de précision possibles sur les **valeurs** associées. Aidez vous d'un **schéma**.

On affiche un carré centré jaune entouré de couleurs variants entre la noir, rouge, vert et jaune.
Les sommets du carré sont formés par les 4 points (0,0,0) (0.5,0,0) (0.5,0.5,0) et (0,0.5,0).

Explication:
Suite à la multiplication par 2 de p3d dans le vertex shader, on travaille (dans le fragment shader) sur un carré dont les coordonnées (x,y) sont comprises entre 0 et 1 (coord. z vaut toujours 0). (x,y) jouent le rôle de coordonnées paramétriques locales au carré.

Lorsque (x,y) est compris dans le carré de centre (0.5, 0.5) et de coté 0.5, alors la couleur est jaune.
Lorsque (x,y) n'est pas dans ce carré, la couleur (r,g,b) vaut (x,y,z).
- r varie donc entre 0 et 1 suivant x.
- g varie donc entre 0 et 1 suivant y.
- b vaut toujours 0.

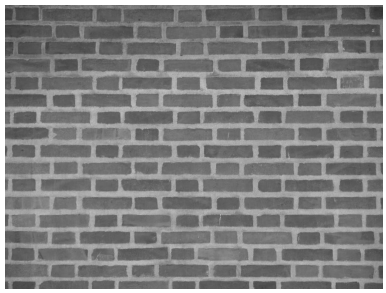


Les couleurs sont interpolées linéairement.
On a donc:
coté inférieur gauche: noir
coté inférieur droit: rouge
coté supérieur gauche: vert
coté supérieur droit: jaune

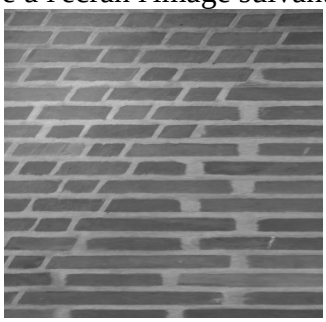
Rem.
|x-0.5|<0.25 && |y-0.5|<0.25 ne définit pas un cercle.
|x-0.5|<0.25 n'est pas la même chose que |x|<0.25 ni que (x-0.5)<0.25.
|x-0.5|<0.25 && |y-0.5|<0.25 n'est pas la même chose que |x-0.5|<0.25 || |y-0.5|<0.25
rouge + vert ne donne pas du marron mais du jaune. Ce n'est pas du noir non plus.

Question 5.

Désormais, on utilise un shader d'affichage standard permettant de gérer les textures (identique à votre shader de TP qui affiche des textures).
On dispose d'un fichier contenant l'image suivante:



On affiche les deux triangles précédents en plaçant à la main les coordonnées de textures pour chacun des sommets. On observe à l'écran l'image suivante:



- Quelles pourraient être les **coordonnées de textures** associées à chaque sommet des triangles? Faites un **schéma** illustrant la **correspondance** entre les sommets du triangles et les coordonnées de texture sur l'image originale.

On supposera que les coordonnées de textures **(0,0)** correspondent au **coté bas à gauche** de l'image, et **(1,0)** au **coté bas à droite**.

***Note**: la valeur précise numérique des coordonnées de texture n'est pas importante.*

```
(0  ,0  ,0) -> vertex[0].texture=vec2(0.4,0);
(0.5,0  ,0) -> vertex[1].texture=vec2(0.6,0);
(0  ,0.5,0) -> vertex[2].texture=vec2(0  ,1);
(0.5,0.5,0) -> vertex[3].texture=vec2(0.6,1);
```