COMPTE-RENDU

HMIN317 – Moteur de Jeux

TP5 – 16 Octobre 2015

Contenu

– Utilisation des Shaders

Sébastien Beugnon
M2 IMAGINA 2015-2016

1. Fonctionnalités

- Affichage d'une pyramide tournant sur elle-même avec deux shaders différents possibles.

2. Démarche de développement

- Triangle

Dans cette partie, je me suis concentré à comprendre le fonctionnement des shaders et de leur utilisation avec le framework Qt. Ainsi à l'aide des classes QopenGLTexture, QopenGLShaderProgram etQOpenGLShader, j'ai pu intégrer rapidement de nouveaux shaders et les textures à la scène simplement. La sphère contenu dans les images suivantes sert à observer l'état de la lumière par défaut au sein de la scène.

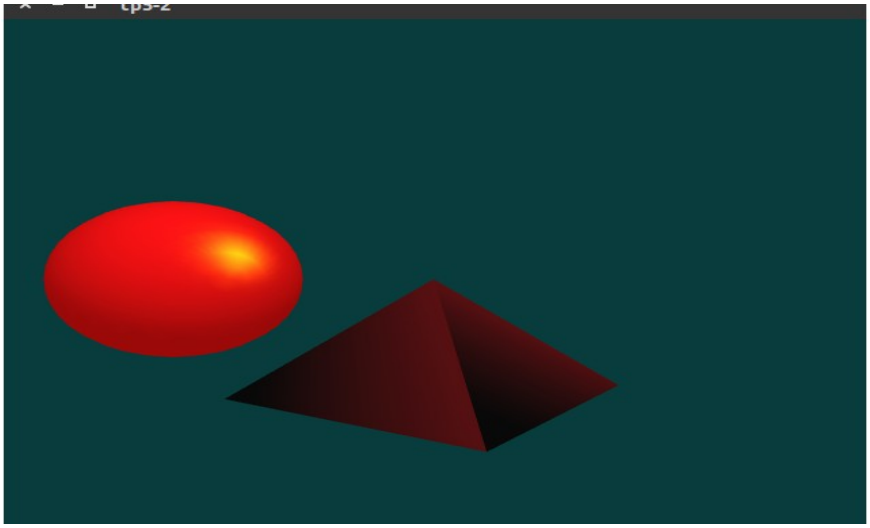

Illustration 1 : Shader « Diffuse Reflection »

Dans la figure précédente, on a appliqué un shader de diffusion de la lumière en utilisant les fonctions et variables built-in du langage GLSL accessibles :

Vertex Shader :

```
varying vec3 N;
varying vec3 v;

uniform highp mat4 matrix;

void main(void)
{

   //v = vec3(gl_ModelViewMatrix * gl_Vertex);// Calcul du vecteur position
   //N = normalize(gl_NormalMatrix * gl_Normal);//Normalisation du vecteur
   gl_Position = vec4(v,1);

}
```

Fragment Shader :

```
varying vec3 N;
varying vec3 v;
//varying vec4 texc;
//varying vec4 col;
uniform sampler2D texture;

void main(void)
{
  //Get Vector between light source and vector v (surely to compute L in
VertexShader)
   vec3 L = normalize(gl_LightSource[0].position.xyz - v);
   //Get Light0 diffuse properties
   vec4 Idiff = gl_FrontLightProduct[0].diffuse * max(dot(N,L), 0.0);
   Idiff = clamp(Idiff, 0.0, 1.0);
   //vec4 texCol = texture2D(texture,gl_TexCoord[0].st);
   //gl_FragColor = texCol * Idiff;
   //vec4 texCol = texture2D(texture,texc.st) * col;
   //gl_FragColor = Idiff * texCol;
   gl_FragColor = Idiff;
}
```

Dans la figure suivante, on a appliqué un shader qui s'occupe de mettre la texture d'une image.

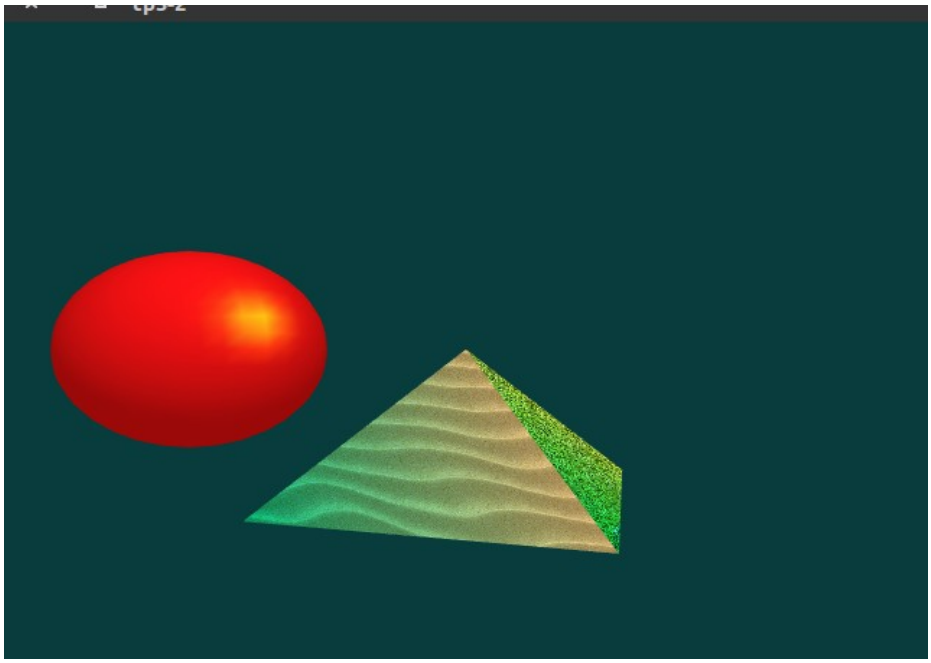

Illustration 2 : Shader « Texture »

On peut remarquer sur cette image que la texture est bleu sur chaque coin gauche des triangles de la pyramide ; en effet, dans ce shader la couleur donnée par la texture a été mélangé avec la couleur du sommet lié.

Vertex Shader :

```
attribute highp vec4 posAttr;
attribute lowp vec4 colAttr;
attribute mediump vec4 texAttr;

varying lowp vec4 col;
varying mediump vec4 texc;

uniform highp mat4 matrix;
//Interface block only after 150
void main() {
  // col = gl_FrontColor; black or default color for front face color
  //col = gl_Color; // Default color used with glColor3f()
   col = colAttr;
   texc = texAttr;

   gl_Position = matrix * posAttr;
}
```

Fragment Shader :

```
uniform sampler2D texture;
varying mediump vec4 texc;
varying lowp vec4 col;
void main() {
    //Blend color with texture
 gl_FragColor = texture2D(texture,texc.st) * col;
// gl_FragColor = gl_Color;//Work but hideous not except color
//Ignore color of texture vertex
//gl_FragColor = col;
//Just texture color
//   gl_FragColor = texture2D(texture,texc.st);
}
```

L'utilisation d'une seule matrice dans le shader est fortement déconseillé, il est parfois nécessaire de transmettre plusieurs matrices comme la matrice du modèle, celle des normales, celle de projection, ou bien celle du monde pour pouvoir réaliser plus d'opérations dans les shaders. De ce fait j'ai accumulé des matrices en les multipliant :

```
    QMatrix4x4 viewMatrix = m_camera->viewMatrix();
    QMatrix4x4 modelViewMatrix = viewMatrix * m_modelMatrix;
    QMatrix3x3 worldNormalMatrix = m_modelMatrix.normalMatrix();
    QMatrix3x3 normalMatrix = modelViewMatrix.normalMatrix();
    QMatrix4x4 mvp = m_camera->projectionMatrix() * modelViewMatrix;
    m_program->setUniformValue( "modelMatrix", m_modelMatrix );
    m_program->setUniformValue( "modelViewMatrix", modelViewMatrix );
    m_program->setUniformValue( "worldNormalMatrix", worldNormalMatrix );
    m_program->setUniformValue( "normalMatrix", normalMatrix );
    m_program->setUniformValue( "mvp", mvp );
```

- Terrain

L'utilisation des QopenGLShaderProgram précédemment m'ont gêné dans l'ajout au précédent TP car ce dernier était rédigé en OpenGL Classique et n'utilisait pas de Qmatrix pour définir les éléments du monde.

3. Structure de données

Pour simplifier l'envoi de données dans les VBO pour OpenGL, j'ai crée une structure VertexData contenant la position du sommet et des coordonnées de texture afin que cette information.

Structure VertexData :

```
struct VertexData
{
    QVector3D position;
    QVector2D texCoord;
};
```

Pour que la caméra de l'environnement s'intègre parfaitement avec les shaders, on doit la définir comme une matrice.