Master Informatique, Synthèse d'Images et Conception Graphique

Rapport de Projet Moteur 3D

KADRI Marouen

10 Janvier 2022

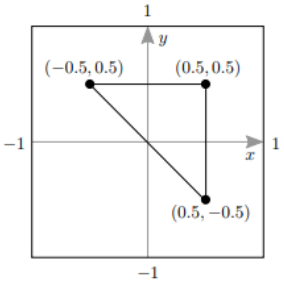
Table des figures

Table des matières

1 TP1

1.1 Objectif du TP

L'objectif de ce TP est simplement afficher un triangle en 2D sur le plan image.



1.2 Shaders :

- **Un shader** est simplement un programme qui s'exécute dans le pipeline graphique et indique à l'ordinateur comment rendre chaque pixel
- **Vertex shader** est le shader qui va définir où est le point que l'on veut afficher
- **Fragment shader** est le shader qui va définir la couleur du point que l'on veut afficher.

lw1.vert* | lw1.frag | lab_work_1.cpp | lab_work_manager.hpp

```
1  #version 450
2  layout (location = 0) in vec3 Position;
3
4  void main()
5  {
6
7  gl_Position = vec4(Position.x, Position.y,Position.z,1.0);
8
9  }
10
```

FIGURE 1 – Vertex Shader 1

lw1.vert* | lw1.frag* | lab_work_1.cpp | lab_work_manager.hpp

```
1  #version 450
2
3  layout( location = 0) out vec4 FragColor;
4
5  void main()
6  {
7
8      FragColor = vec4(1.0f, 0.0f, 0.0f, 0.0f);
9
10 }
```

FIGURE 2 – Fragment Shader 1

1.3 Création du programme OpenGL

- **Création des Vertex** :Nous avons besoin de 3 sommets pour créer un triangle

```
vertices = {
    Vec3f( -0.5f, 0.5f, 0.0f ),
    Vec3f( 0.5f, 0.5f, 0.0f ),
    Vec3f( 0.5, -0.5f, 0.0f ),

};
```

FIGURE 3 – Positions des sommets 2

- **Configuration de VAO & VBO** :Nous avons utilisé les Buffers pour transférer les données du CPU vers le GPU .En appelant **glVertexAttribPointer** pour déclarer le type et l'organisation mémoire de l'attribut , nous avons également appeler **glEnableVertexAttribArray** (Index) pour indiquer à la carte graphique que l'attribut de sommet indexé comme Index est maintenant activé.

```
77      glGenVertexArrays( 1, &VAO );
78      glGenBuffers( 1, &VBO );
79      glBindVertexArray( VAO );
80      glBindBuffer( GL_ARRAY_BUFFER, VBO );
81      glBufferData( GL_ARRAY_BUFFER, sizeof(Vec3f)*vertices.size(), vertices.data(), GL_STATIC_DRAW );
82      glVertexAttribPointer( 0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof( float ), (void *)0 );
83      glEnableVertexAttribArray( 0 );
84      glBindBuffer( GL_ARRAY_BUFFER, 0 );
85      glBindVertexArray( 0 );
```

FIGURE 4 – Configuration VAO VBO 1

1.4 Rendu

1. Liez le VAO au programme avec glBindVertexArray.
2. Lancez le pipeline avec glDrawArrays.
3. Déliez le VAO et le programme avec glBindVertexArray en lui passant 0.

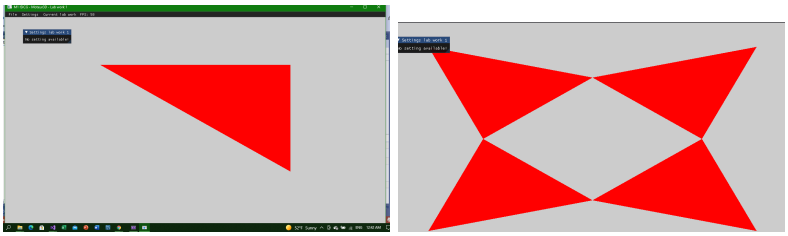
```
void LabWork1::render() {

glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
//draw One triagnle triangle
glBindVertexArray( VAO);
glDrawArrays( GL_TRIANGLES, 0, 3 );
glBindVertexArray( 0 );

}
```

FIGURE 5 – Algorithme 1 de Rendu

Dans un premier temps pour afficher le triangle J'ai utilisé trois points :V1 & V2 & V3,après j'ai joué avec les points pour avoir une forme géométrique (ensemble des triangles)



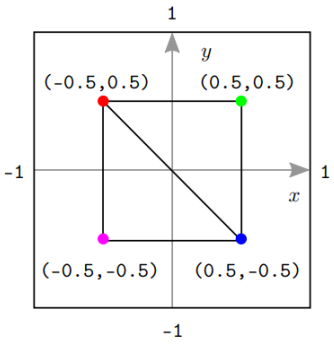
(a) (b)

FIGURE 6 – (a) Triangle (b)Forme géométrique

2 TP2

2.1 Objectif du Tp

Dans cet exercice, nous allons afficher un quad, constitué de deux triangles



2.2 Réalisation

- Nous avons utilisé le Buffer **EBO** pour stocker les indices des positions

```
Position_Vertices = { Vec3f( -0.5f, 0.5f, 0.0f ),
                      Vec3f( 0.5f, 0.5f, 0.0f ),
                      Vec3f( 0.5, -0.5f, 0.0f ),
                      Vec3f( -0.5f, -0.5f, 0.0f ) };


indices= { 0, 1, 3, 1, 2, 3 };
```

- Nous avons utiliser la fonction **glDrawElements** au lieu de **glDrawArrays**

```
glDrawElements( GL_TRIANGLES, _cube._indices.size(), GL_UNSIGNED_INT, 0 );
```

- à l'aide d'un variables de contrôle **"Uniform"** nous pouvons envoyer les variables de CPU vers le GPU
- Glissement sur l'axe X de quat se fait a travers une fonction **sin** en fonction de temps dans le boucle **Animation**

2.3 Résultat

FIGURE 7 – Résultat 2

3 TP3

3.1 Objectif du Tp

Dans ce TP , nous allons afficher un cube basé sur la projection perspective

3.2 Réalisation

1. Ajout de la View Matrix à travers une variable uniforme
2. Calcul de la matrice dans la fonction **_computeViewMatrix**
3. mise à jour la variable uniforme dans **_updateViewMatrix()**
4. Ajout d'une variable uniforme pour la **Projection Matrix**
5. calcul de la matrice da fonction **_computeProjectionMatrix()**
6. mise à jour la variable uniforme dans la fonction **_updateProjectionMatrix()**
7. Positionner la caméra

3.3 Résultat

à travers la matrice MVP que nous avons codé on a ce Rendu :

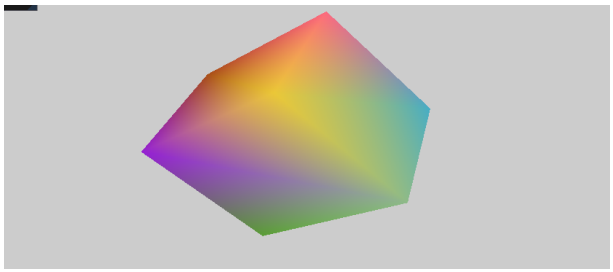
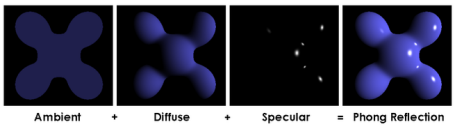


FIGURE 8 – Résultat 3

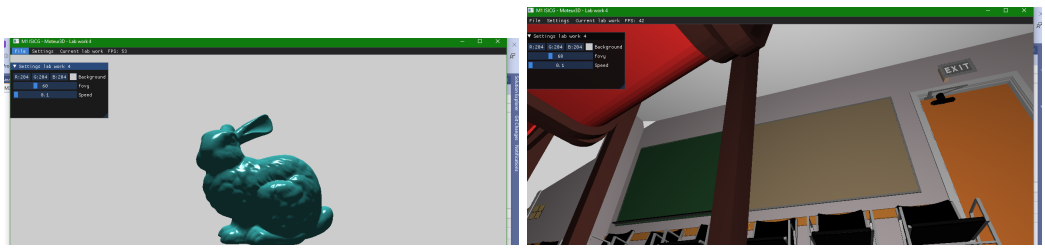
4 TP4

4.1 Objectif du Tp

Dans ce TP, vous allez afficher un modèle 3D chargé depuis un fichier et calculer l'éclairage local selon le modèle de Phong



4.2 Réalisation



(a) (b)

FIGURE 9 – (a) Bunny de Stanford (b) Salle de Conférence

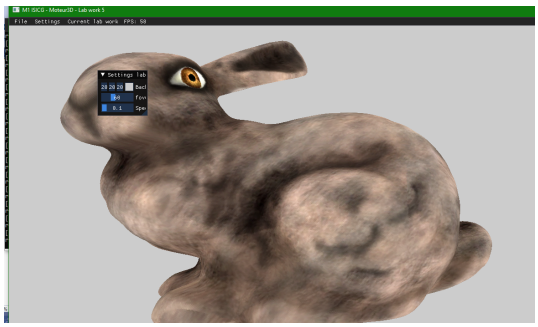
5 TP5

5.1 Objectif du Tp

Dans ce TP , on va appliquer les textures discrètes pour changer l'apparence de nos objects.

5.2 Réalisation

- **Bunny de Stanford**



- **Sponza**

6 Bibliographie

Ci-dessous une collection de références vers les différentes ressources m'ayant servi lors de la réalisation de ce projet

Références

[1] Modèle d'illumination de Phong *:*
`https://fr.wikipedia.org/wiki/Ombrage_de_Phong`