Compte rendu :

Alexandre DUCLOS

Step 1 :

On doit dessiner un triangle rouge avec un fond bleu, pour cela :

-On reprend le code playground.cpp

-On rajoute les librairies :

```
#include <stdio.h>
#include <stdlib.h>
#include <vector>
#include <GL/glew.h>
#include <GLFW/glfw3.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <common/shader.hpp>
#include <common/objloader.hpp>
#include <common/texture.hpp>
```

-On rajoute les lignes demandées dans le tp :

```
GLuint VertexArrayID;
glGenVertexArrays(1, &VertexArrayID);
glBindVertexArray(VertexArrayID);

GLuint programID = LoadShaders("playground_steps/step1/SimpleVertexShader.vertexshader", "playground_steps/step1/SimpleFragmentShader.fragmentshader");
```

-On rajoute cette constante qui déclare les coordonnées des sommets du triangle :

```
static const GLfloat g_vertex_buffer_data[] =
{
    -1.0f,-1.0f,0.0f,
    1.0f,-1.0f,0.0f,
    0.0f,1.0f,0.0f,
};
```

-On déclare : GLuint vertexbuffer;

-On clear l'écran avec : glClear(GL_COLOR_BUFFER_BIT);

-On rajoute :

```
glGenBuffers(1, &vertexbuffer);
glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(g_vertex_buffer_data), g_vertex_buffer_data, GL_STATIC_DRAW);
```

-On utilise notre shader avec : glUseProgram(programID);

-On configure l'attribut de vertex dans le programme de shader OpenGL :

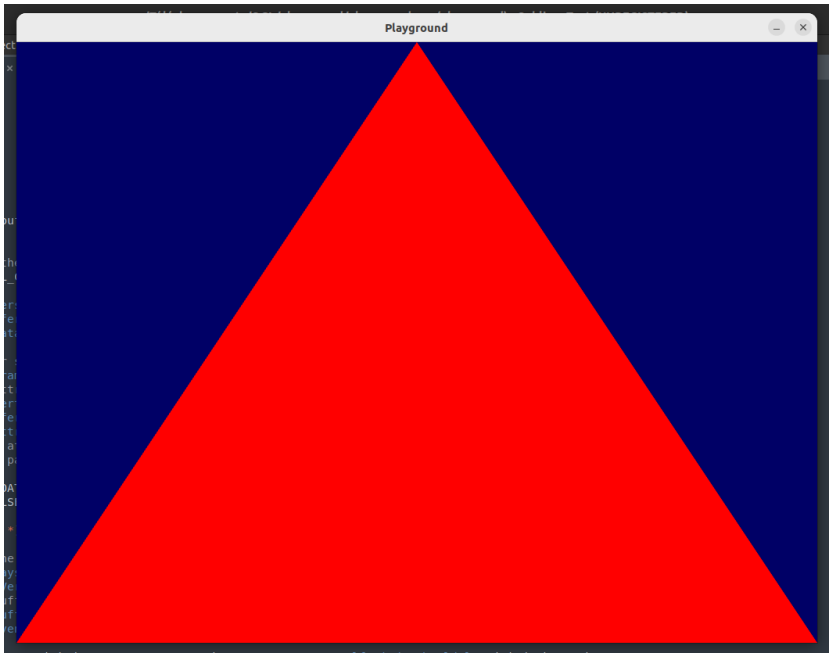
```
// 1rst attribute buffer : vertices
glEnableVertexAttribArray(0);
glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
glVertexAttribPointer(
    0, // attribute 0.
    // No particular reason for 0, but must match the layout in the shader.
    3,          // size
    GL_FLOAT, // type
    GL_FALSE, // normalized?
    0,          // stride
    (void *)0 // array buffer offset
);
```

-On dessine notre triangle (3 vertices = 1 triangle) :

```
// Draw the triangle !
glDrawArrays(GL_TRIANGLES, 0, 3); // 3 indices starting at 0 -> 1 triangle
glDisableVertexAttribArray(0);
```

-Lorsque l'on quitte, on nettoie avec : glDeleteBuffers(1, &vertexbuffer);

On obtient ce résultat :



Step 2 :

On doit dans un premier temps dessiner 3 triangles avec de 3 couleurs différentes puis que chaque sommet de chaque triangle ait une couleur différente :

On adapte la ligne suivante :

```
GLuint programID = LoadShaders("playground_steps/step1/SimpleVertexShader.vertexshader", "playground_steps/step1/SimpleFragmentShader.fragmentshader");
```

Pour dessiner les 3 triangles, on se retrouve avec 9 coordonnées :

```
// Vertex data for three triangles
static const GLfloat g_vertex_buffer_data[] = {
    -1.0f, -1.0f, 0.0f,
    -1.0f,  1.0f, 0.0f,
     1.0f,  1.0f, 0.0f,

     1.0f,  1.0f, 0.0f,
     1.0f, -1.0f, 0.0f,
    -1.0f, -1.0f, 0.0f,

    -1.0f,  1.0f, 0.0f,
     0.0f,  1.5f, 0.0f,
     1.0f,  1.0f, 0.0f
};
```

On définit les couleurs pour chaque sommet des 3 triangles :

```
    // One color for each vertex.
    static const GLfloat g_color_buffer_data[] = {
        0.0f, 1.0, 0.0f,
        0.0f, 1.0, 0.0f,
        0.0f, 1.0, 0.0f,

        1.0, 0.0f, 0.0f,
        1.0, 0.0f, 0.0f,
        1.0, 0.0f, 0.0f,

        0.0f, 0.0f, 1.0,
        0.0f, 0.0f, 1.0,
        0.0f, 0.0f, 1.0,
    };
```

On déclare ce buffer : `GLuint colorbuffer;`

Puis on rajoute ceci dans la boucle :

```
glGenBuffers(1, &colorbuffer);
glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(g_vertex_buffer_data), g_vertex_buffer_data, GL_STATIC_DRAW);
```

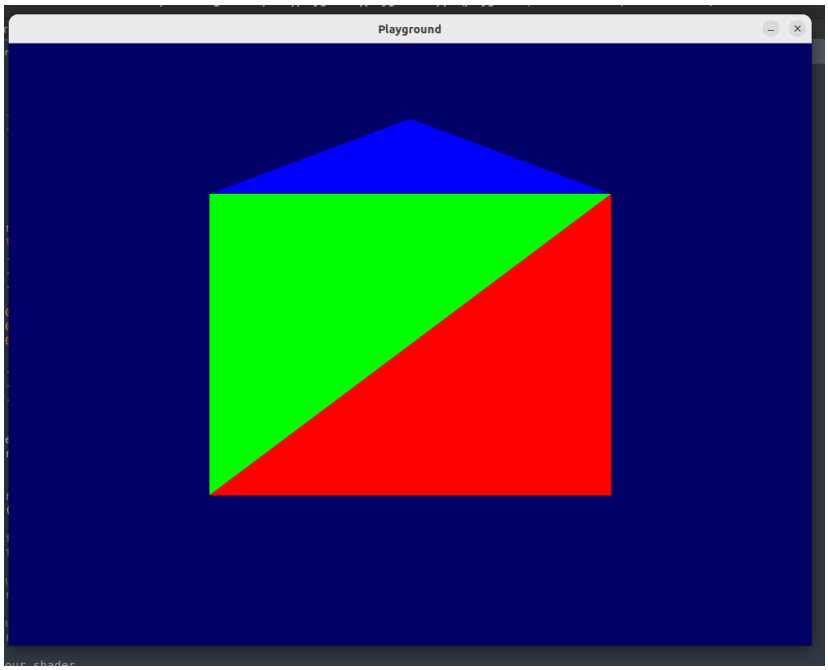
On rajoute cette étape pour attribuer les couleurs :

```
// 2nd attribute buffer : colors
glEnableVertexAttribArray(1);
glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
glVertexAttribPointer(
1, // attribute 1.
//No particular reason for 1, but must match the layout in the shader.
3, // size
GL_FLOAT, // type
GL_FALSE, // normalized?
0, // stride
(void*)0 // array buffer offset
);
```

On dessine nos 3 triangles :

```
// Draw the triangles
glDrawArrays(GL_TRIANGLES, 0, 9); // 9 vertices (3 triangles)
```

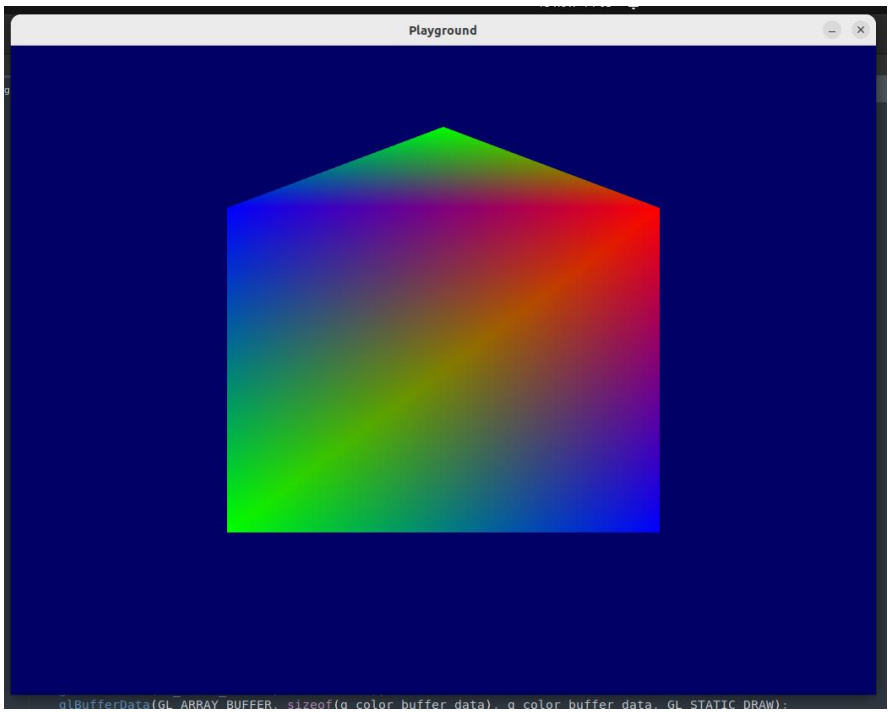
On obtient :



Pour obtenir des triangles qui n'ont pas une couleur uniforme, avec chaque sommet une couleur différente, on modifie :

```
// One color for each vertex.
static const GLfloat g_color_buffer_data[] = {
    0.0f, 1.0, 0.0f,
    0.0f, 0.0f, 1.0,
    1.0, 0.0f, 0.0f,

    1.0, 0.0f, 0.0f,
    0.0f, 0.0f, 1.0,
    0.0f, 1.0, 0.0f,


    0.0f, 0.0f, 1.0,
    0.0f, 1.0, 0.0f,
    1.0, 0.0f, 0.0f,

};
```

On obtient :



Step 3 :

On doit tracer la même forme que dans la step 2 mais de 2 manières différentes, j'ai utilisé des lines dans la v1 puis des line_strip dans la v2 :

Pour la v1 : en utilisant les line on doit relier les points de cette manière :

Par exemple : 0-1, 2-3, 4-5, 6-7, et 8-9

```
static const GLfloat g_vertex_buffer_data[] = {
    -1.0f, -1.0f, 0.0f,
    -1.0f,  1.0f, 0.0f,

    -1.0f,  -1.0f, 0.0f,
     1.0f,  -1.0f, 0.0f,

     1.0f, -1.0f, 0.0f,
     1.0f, 1.0f, 0.0f,

    -1.0f,  1.0f, 0.0f,
     0.0f,  1.5f, 0.0f,

     1.0f,  1.0f, 0.0f,
     0.0f,  1.5f, 0.0f,

     1.0f,  1.0f, 0.0f,
     1.0f,  0.0f, 0.0f,

     -1.0f,  1.0f, 0.0f,
     1.0f,  1.0f, 0.0f,

     1.0f,  1.0f,   0.0f,
     -1.0f, -1.0f,  0.0f

};
```

On adapte le nombre de couleurs au nombre de lignes tracées, ici 16

On remplace le draw triangle par :

```
// Draw
glDrawArrays(GL_LINES, 0, 16); // 16 indices starting at 0 -> 8 lines
```

On utilise ce mode de remplissage pour les lignes :

```
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
```

Pour la v2 : en utilisant line_strip on doit relier les points de cette manière :

Par exemple : 0-1-2-3-4

```
static const GLfloat g_vertex_buffer_data[] = {
    -1.0f, 1.0f, 0.0f,
    -1.0f,  -1.0f, 0.0f,

     1.0f,  -1.0f, 0.0f,
     1.0f,  1.0f, 0.0f,

    -1.0f, 1.0f, 0.0f,
     0.0f, 1.5f, 0.0f,

     1.0f,  1.0f, 0.0f,
    -1.0, -1.0, 0.0


};
```

On adapte le nombre de couleurs au nombre de lignes tracées, ici 8

On remplace le draw triangle par :

```
// Draw
//glDrawArrays(GL_LINES, 0, 14); // 14 indices starting at 0 -> 7 lines
glDrawArrays(GL_LINE_STRIP, 0, 8); // 14 indices starting at 0 -> 7 lines
```

On utilise ce mode de remplissage pour les lignes

```
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
```

Step 4 :

On se base sur ce que l'on a fait au step 2 :

Et on rajoute le triangle blanc après les 3 autres triangles et en le mettant devant en modifiant son Z.

Ce qui donne le code suivant :

```
// Vertex data for 4 triangles

static const GLfloat g_vertex_buffer_data[] = {

    -1.0f, -1.0f, 0.0f,
    -1.0f,  1.0f, 0.0f,
     1.0f,  1.0f, 0.0f,

     1.0f,  1.0f, 0.0f,
     1.0f, -1.0f, 0.0f,
    -1.0f, -1.0f, 0.0f,

    -1.0f,  1.0f, 0.0f,
     0.0f,  1.5f, 0.0f,
     1.0f,  1.0f, 0.0f,

     -1.0f, -1.0f, 0.2f,
     0.0f,  1.0f, 0.2f,
     1.0f,  -1.0f, 0.2f
};
```

```
// One color for each vertex.
static const GLfloat g_color_buffer_data[] = {

    1.0, 0.0f, 0.0f,
    1.0, 0.0f, 0.0f,
    1.0, 0.0f, 0.0f,

    0.0f, 1.0, 0.0f,
    0.0f, 1.0, 0.0f,
    0.0f, 1.0, 0.0f,

    0.0f, 0.0f, 1.0,
    0.0f, 0.0f, 1.0,
    0.0f, 0.0f, 1.0,

    1.0, 1.0f, 1.0f,
    1.0, 1.0f, 1.0f,
    1.0, 1.0f, 1.0f,
```

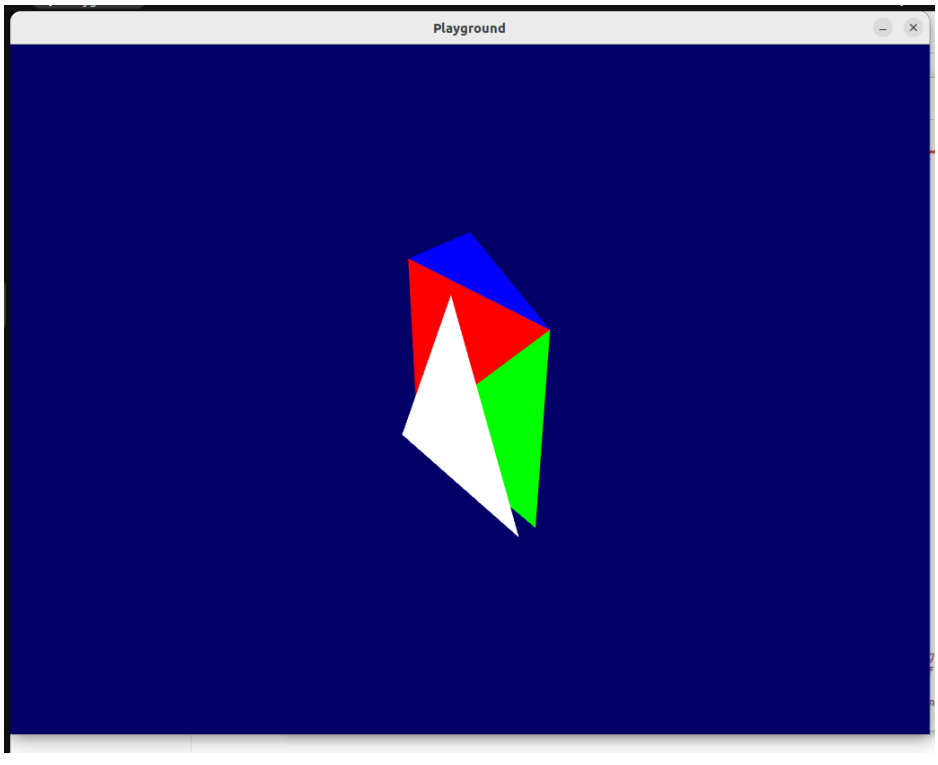
On rajoute ce code en dessous pour gérer la caméra :

```
GLuint MatrixID = glGetUniformLocation(programID, "MVP");

// Projection matrix : 45 degrees Field of View, 4:3 ratio, display range : 0.1 unit <-> 100 units
glm::mat4 Projection = glm::perspective(45.0f, 4.0f / 3.0f, 0.1f, 100.0f);
// Or, for an ortho camera :
//glm::mat4 Projection = glm::ortho(-10.0f,10.0f,-10.0f,10.0f,0.0f,100.0f); // In world coordinates

// Camera matrix
glm::mat4 View = glm::lookAt(
glm::vec3(4,3,3), // Camera is at (4,3,3), in World Space
glm::vec3(0,0,0), // and looks at the origin
glm::vec3(0,1,0) // Head is up (set to 0,-1,0 to look upside-down)

);

// Model matrix : an identity matrix (model will be at the origin)

glm::mat4 Model = glm::mat4(1.0f);

// Our ModelViewProjection : multiplication of our 3 matrices

glm::mat4 MVP = Projection * View * Model;

// Remember, matrix multiplication is the other way around
```

Et celui-ci dans la boucle principale :

```
glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MVP[0][0]);
```

Ce qui donne ce résultat :



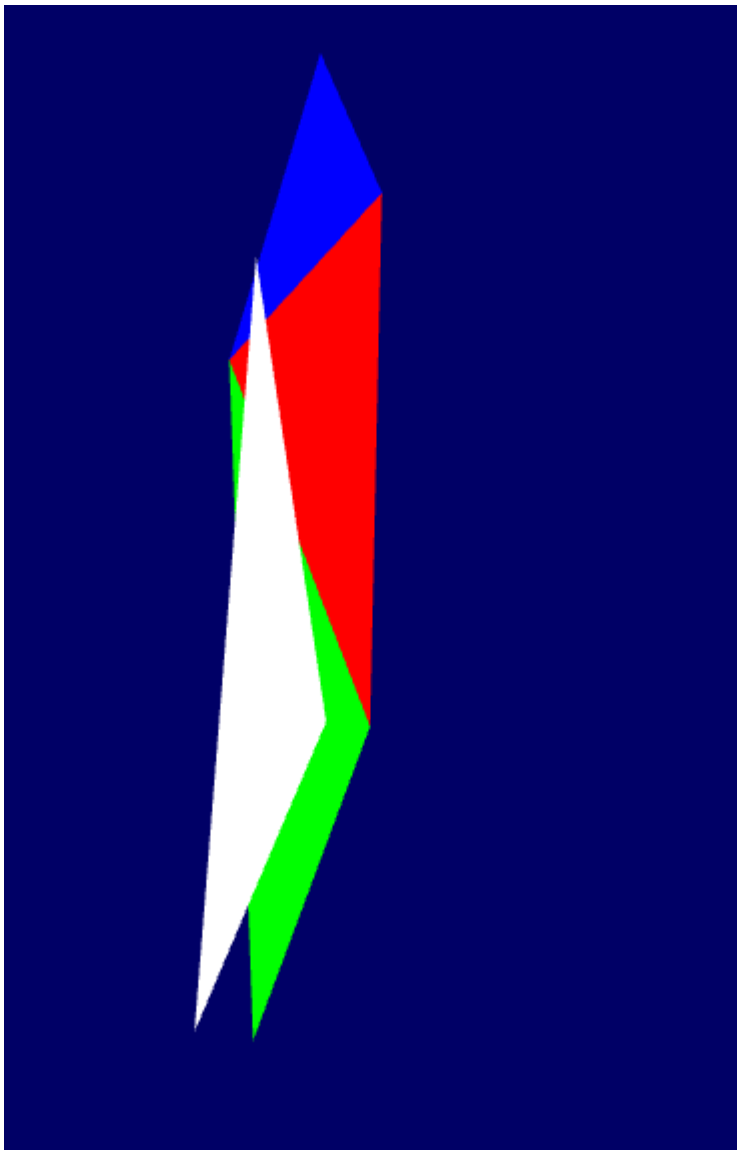
Step 5 :

Pour obtenir la rotation, translation, zoom : on place glm ::mat4 MVP dans la boucle principale,

Ainsi que les 3 modèles qui forment une matrice de multiplication

```
// Our ModelViewProjection : multiplication of our 3 matrices
glm::mat4 MVP = Projection * View * Model;
// Remember, matrix multiplication is the other way around

Model = glm::translate(Model, glm::vec3(0.0f, 0.001f, 0.0f));
Model = glm::scale(Model, glm::vec3(1.001f, 1.001f, 1.0f));
Model = glm::rotate(Model, glm::radians(1.0f), glm::vec3(0.0f, 1.0f, 0.0f));
```

On obtient une rotation, un zoom et un déplacement de notre figure :



Step 6 :

Afin d'obtenir comme dans l'exemples 2 formes qui tournent, on déclare hors de la boucle ces 2 lignes ainsi que lastFrame et angle que l'on va utiliser dans la boucle :

```
glm::mat4 Model1 = glm::mat4(1.0f);
glm::mat4 Model2 = glm::mat4(1.0f);
float lastFrame=0,angle=0;
```

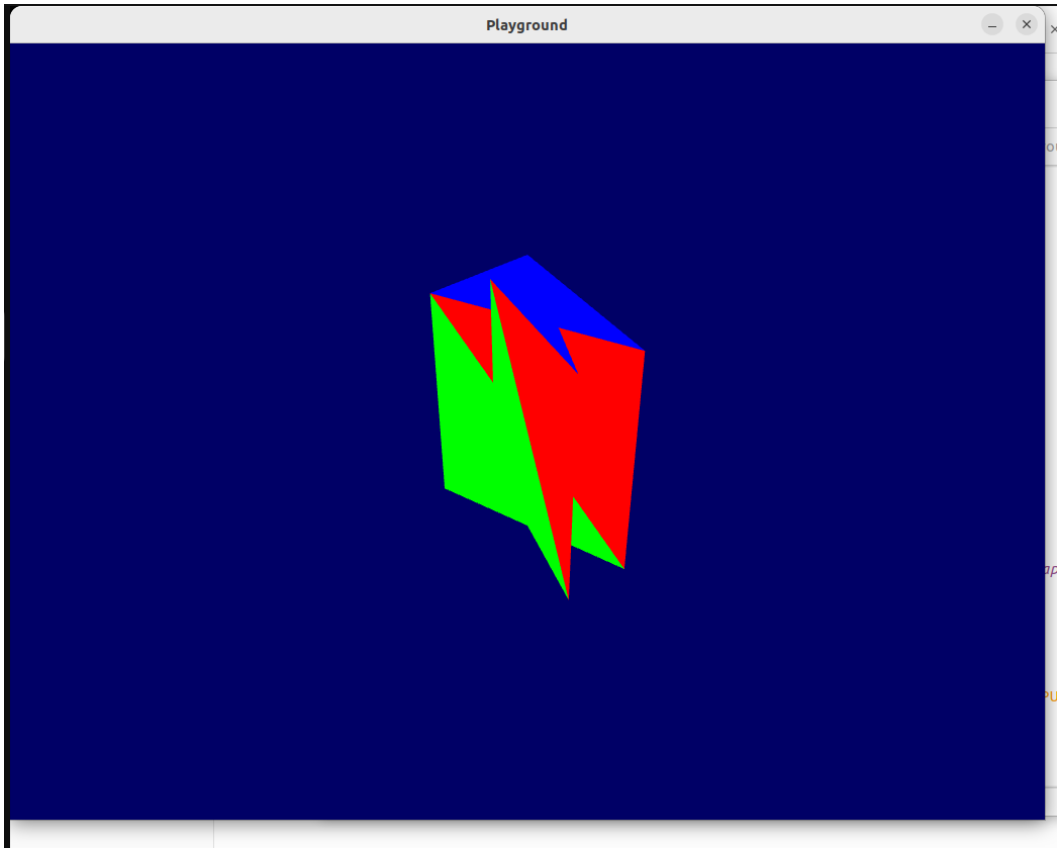
Dans notre boucle principale on retrouve ce bloc qui permet la rotation des 2 formes :

```
float currentFrame = glfwGetTime();
float deltaTime = currentFrame - lastFrame;
lastFrame = currentFrame;


angle += 3.14159f / 2.0f * deltaTime;
float c = (float)cos(angle);
float s = (float)sin(angle);


Model1 = {c, 0.0f, s, 0.0f,
        0.0f, 1.0f, 0.0f, 0.0f,
        -s, 0.0f, c, 0.0f,
        0.0f, 0.0f, 0.0f, 1.0f};


Model2 = {c, 0.0f, -s, 0.0f,
        0.0f, 1.0f, 0.0f, 0.0f,
        s, 0.0f, c, 0.0f,
        0.0f, 0.0f, 0.0f, 1.0f};
```

Plus bas dans la boucle on dessine les 2 formes et en utilisant la matrice MVP pour chacune des formes :

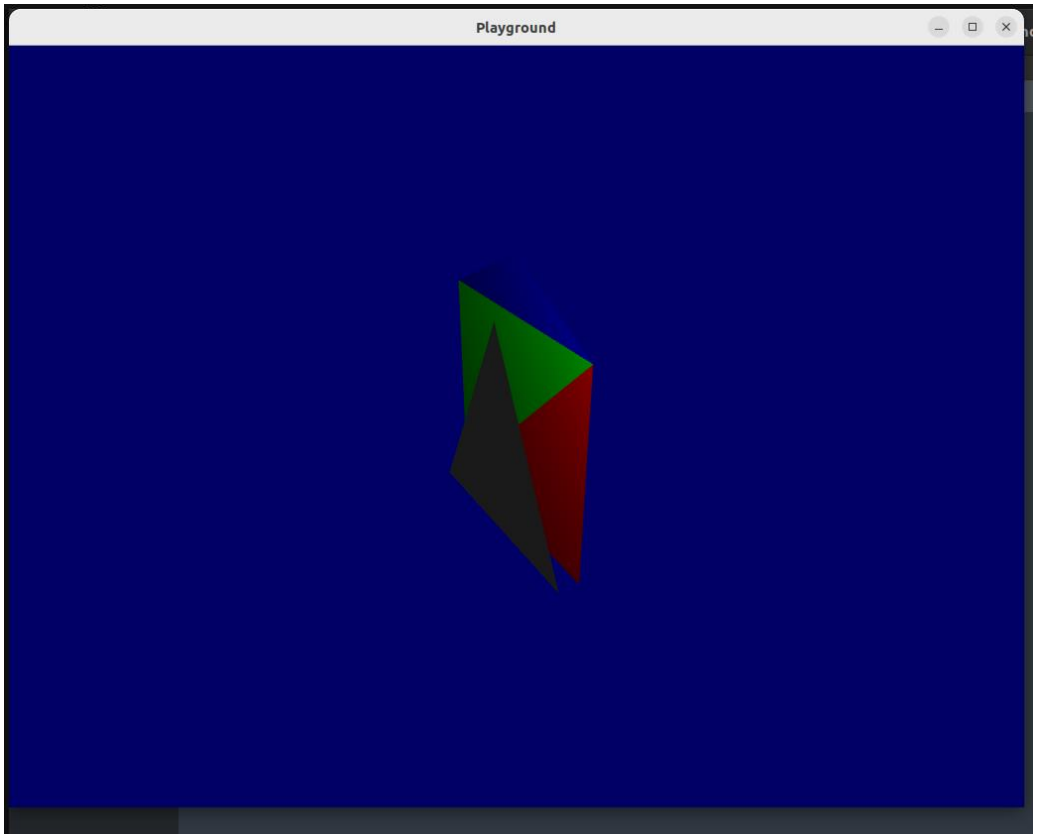
```
glm::mat4 MVP;
//Forme 1
MVP = Projection * View * Model1;
glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MVP[0][0]);
// Draw the triangles
glDrawArrays(GL_TRIANGLES, 0, 9);

//Forme 2
MVP = Projection * View * Model2;
glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MVP[0][0]);
// Draw the triangles
glDrawArrays(GL_TRIANGLES, 0, 9);
```

Ce qui permet d’obtenir ce résultat :



Step 7 :

On ajoute les codes donnés dans le TP et on obtient cette figure en utilisant 1 forme :



Step 8 :

On redéfinit les coordonnées de nos triangles afin d'obtenir des faces et de former un cube,

On change les couleurs et enfin on adapte les normales :

Les normales :

```
    // Normal
static const GLfloat g_normal_buffer_data[] = {
        //1
        0.0f, 0.0f, 1.0f,
        0.0f, 0.0f, 1.0f,
        0.0f, 0.0f, 1.0f,

        0.0f, 0.0f, 1.0f,
        0.0f, 0.0f, 1.0f,
        0.0f, 0.0f, 1.0f,
        //2
        0.0f, 0.0f, -1.0f,
        0.0f, 0.0f, -1.0f,
        0.0f, 0.0f, -1.0f,

        0.0f, 0.0f, -1.0f,
        0.0f, 0.0f, -1.0f,
        0.0f, 0.0f, -1.0f,
        //3
        -1.0f, 0.0f, 0.0f,
        -1.0f, 0.0f, 0.0f,
        -1.0f, 0.0f, 0.0f,

        -1.0f, 0.0f, 0.0f,
        -1.0f, 0.0f, 0.0f,
        -1.0f, 0.0f, 0.0f,
        //4
        1.0f, 0.0f, 0.0f,
        1.0f, 0.0f, 0.0f,
        1.0f, 0.0f, 0.0f,

        1.0f, 0.0f, 0.0f,
        1.0f, 0.0f, 0.0f,
        1.0f, 0.0f, 0.0f,
        //5
        0.0f, 1.0f, 0.0f,
        0.0f, 1.0f, 0.0f,
        0.0f, 1.0f, 0.0f,

        0.0f, 1.0f, 0.0f,
        0.0f, 1.0f, 0.0f,
        0.0f, 1.0f, 0.0f,
        //6
        0.0f, -1.0f, 0.0f,
        0.0f, -1.0f, 0.0f,
        0.0f, -1.0f, 0.0f,

        0.0f, -1.0f, 0.0f,
        0.0f, -1.0f, 0.0f,
        0.0f, -1.0f, 0.0f};
```

Les coordonnées des triangles :

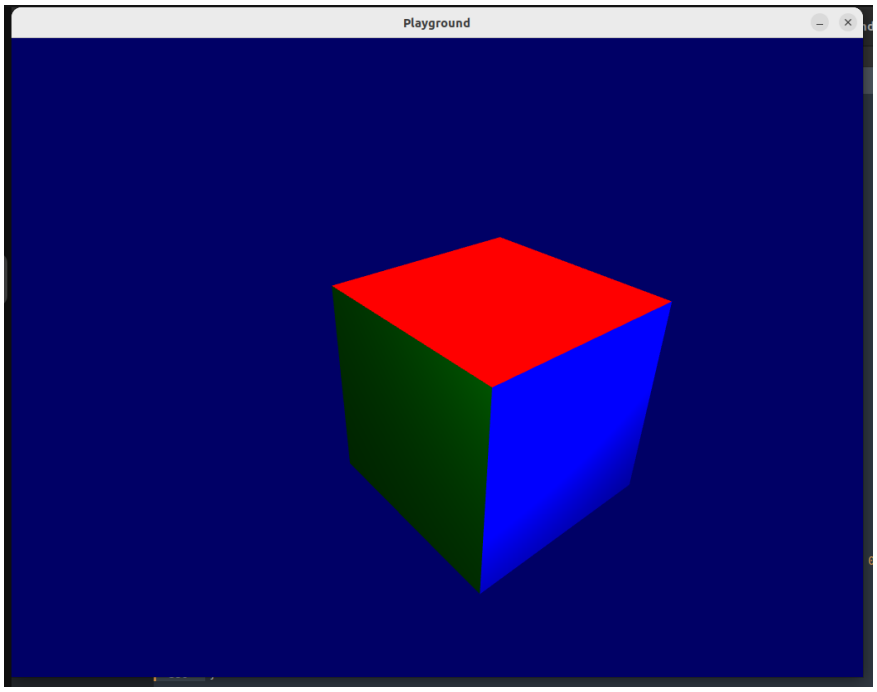
```
// Vertex data for a cube
static const GLfloat g_vertex_buffer_data[] = {
    //Face 1
    -1.0f, 1.0f, 0.2f,
    1.0f, 1.0f, 0.2f,
    -1.0f, -1.0f, 0.2f,

    1.0f, 1.0f, 0.2f,
    1.0f, -1.0f, 0.2f,
    -1.0f, -1.0f, 0.2f,
    //Face 2
    -1.0f, 1.0f, -1.8f,
    1.0f, 1.0f, -1.8f,
    -1.0f, -1.0f, -1.8f,

    1.0f, 1.0f, -1.8f,
    1.0f, -1.0f, -1.8f,
    -1.0f, -1.0f, -1.8f,
    //Face 3
    -1.0f, 1.0f, -1.8f,
    -1.0f, -1.0f, 0.2f,
    -1.0f, -1.0f, -1.8f,

    -1.0f, 1.0f, -1.8f,
    -1.0f, 1.0f, 0.2f,
    -1.0f, -1.0f, 0.2f,
    //Face 4
    1.0f, 1.0f, 0.2f,
    1.0f, 1.0f, -1.8f,
    1.0f, -1.0f, 0.2f,

    1.0f, 1.0f, -1.8f,
    1.0f, -1.0f, -1.8f,
    1.0f, -1.0f, 0.2f,
    //Face 5
    -1.0f, 1.0f, 0.2f,
    -1.0f, 1.0f, -1.8f,
    1.0f, 1.0f, -1.8f,

    -1.0f, 1.0f, 0.2f,
    1.0f, 1.0f, -1.8f,
    1.0f, 1.0f, 0.2f,
    //Face 6
    -1.0f, -1.0f, 0.2f,
    1.0f, -1.0f, 0.2f,
    1.0f, -1.0f, -1.8f,

    -1.0f, -1.0f, 0.2f,
    1.0f, -1.0f, -1.8f,
    -1.0f, -1.0f, -1.8f};
```

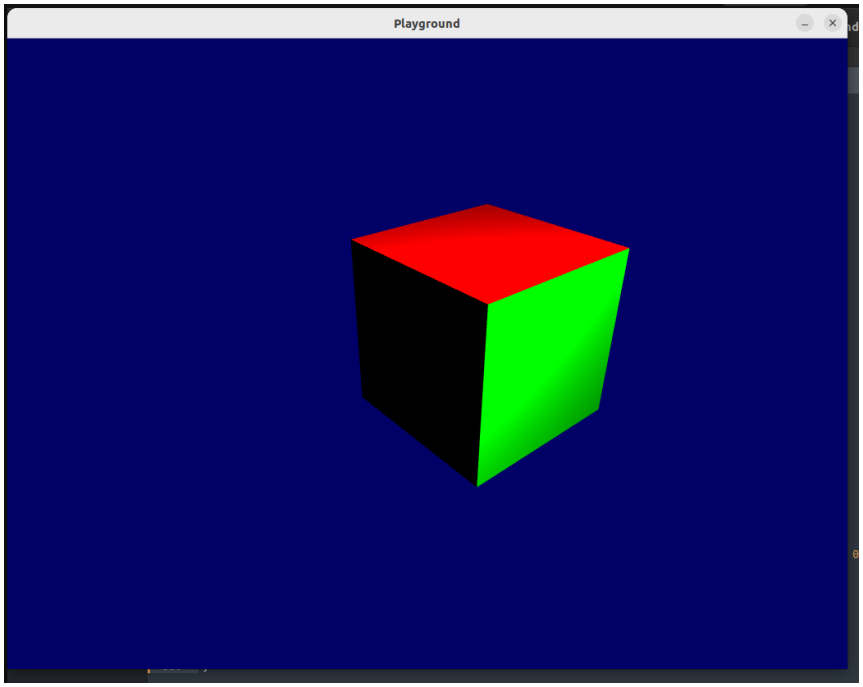
Les couleurs : c’est la même chose

On augmente jusqu’à 36 cette ligne afin que tous les triangles s’affichent :

```
// Draw the triangles
glDrawArrays(GL_TRIANGLES, 0, 36);
```

Pour obtenir ce résultat :





Step 9 :

On sort ces lignes de la boucle :

```
glGenBuffers(1, &vertexbuffer);
glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(g_vertex_buffer_data), g_vertex_buffer_data, GL_STATIC_DRAW);

glGenBuffers(1, &colorbuffer);
glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(g_color_buffer_data), g_color_buffer_data, GL_STATIC_DRAW);
```

On sort de la boucle la lumière :

```
glm::vec3 lightPos = glm::vec3(4, 4, 1);
glUniform3f(LightID, lightPos.x, lightPos.y, lightPos.z);
```

Pour obtenir ce résultat :