UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

INGENIERÍA EN COMPUTACIÓN

LABORATORIO DE COMPUTACIÓN GRÁFICA e INTERACCIÓN HUMANO COMPUTADORA

REPORTE DE PRÁCTICA Nº 02

NOMBRE COMPLETO: Uriarte Ortiz Enrique Yahir

Nº de Cuenta: 318234757

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 04

SEMESTRE 2025-1

FECHA DE ENTREGA LÍMITE: Sábado 24 de Agosto del 2024

CALIFICACIÓN: _____

REPORTE DE PRÁCTICA:

1. *Ejecución de los ejercicios.*
    1) *Dibujar las iniciales de sus nombres, cada letra de un color diferente.*

Para desarrollar esta primera actividad realice nuevamente como en ejercicios anteriores un boceto de dónde colocar las coordenadas de las letras, pero como en esta actividad era necesario implementar el desplazamiento, cree cada una con respecto al origen y luego las desplace, por eso en el boceto aparece un eje cardinal en medio de cada letra, como guía de como se definieron en un inicio.

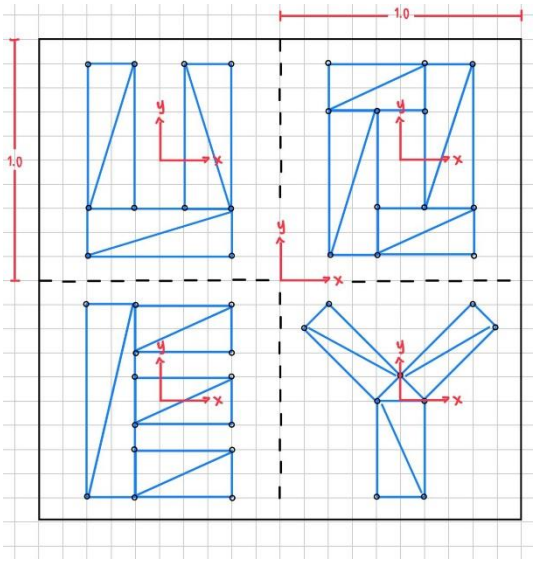

Fig. 1. *Boceto para las coordenadas de los triángulos.*

Para definirlas en el código, cree un ***"GLfloat"*** para cada letra, las únicas diferencias entre cada uno es la posición de los vértices y la cantidad definida, lo demás es igual.

```
void CrearLetras(){
    GLfloat vertices_letra_U[] = {//Letra U
        //    X      Y      Z      R      G      B
              0.1f,-0.2f, 0.1f,  0.0f, 0.0f, 1.0f,
              0.1f, 0.4f, 0.1f,  0.0f, 0.0f, 1.0f,
              0.3f,-0.2f, 0.1f,  0.0f, 0.0f, 1.0f,
              0.3f,-0.2f, 0.1f,  0.0f, 0.0f, 1.0f,
              0.3f, 0.4f, 0.1f,  0.0f, 0.0f, 1.0f,
              0.1f, 0.4f, 0.1f,  0.0f, 0.0f, 1.0f,
             -0.1f,-0.2f, 0.1f,  0.0f, 0.0f, 1.0f,
             -0.1f, 0.4f, 0.1f,  0.0f, 0.0f, 1.0f,
             -0.3f,-0.2f, 0.1f,  0.0f, 0.0f, 1.0f,
             -0.3f,-0.2f, 0.1f,  0.0f, 0.0f, 1.0f,
             -0.3f, 0.4f, 0.1f,  0.0f, 0.0f, 1.0f,
             -0.1f, 0.4f, 0.1f,  0.0f, 0.0f, 1.0f,
             -0.3f,-0.2f, 0.1f,  0.0f, 0.0f, 1.0f,
             -0.3f,-0.4f, 0.1f,  0.0f, 0.0f, 1.0f,
              0.3f,-0.2f, 0.1f,  0.0f, 0.0f, 1.0f,
             -0.3f,-0.4f, 0.1f,  0.0f, 0.0f, 1.0f,
              0.3f,-0.4f, 0.1f,  0.0f, 0.0f, 1.0f,
              0.3f,-0.2f, 0.1f,  0.0f, 0.0f, 1.0f};
    MeshColor* letra_U = new MeshColor();
    letra_U->CreateMeshColor(vertices_letra_U, 108);
    meshColorList.push_back(letra_U);
```

Fig. 2. *Código de triángulos de la letra U.*

```
GLfloat vertices_letra_O[] = {//Letra O
    //    X     Y     Z      R     G     B
         0.1f,-0.2f, 0.1f,  0.0f, 1.0f, 0.0f,
         0.1f, 0.4f, 0.1f,  0.0f, 1.0f, 0.0f,
         0.3f, 0.4f, 0.1f,  0.0f, 1.0f, 0.0f,
         0.3f,-0.2f, 0.1f,  0.0f, 1.0f, 0.0f,
         0.3f, 0.4f, 0.1f,  0.0f, 1.0f, 0.0f,
         0.1f,-0.2f, 0.1f,  0.0f, 1.0f, 0.0f,
        -0.1f,-0.4f, 0.1f,  0.0f, 1.0f, 0.0f,
        -0.1f, 0.2f, 0.1f,  0.0f, 1.0f, 0.0f,
        -0.3f,-0.4f, 0.1f,  0.0f, 1.0f, 0.0f,
        -0.3f, 0.2f, 0.1f,  0.0f, 1.0f, 0.0f,
        -0.3f,-0.4f, 0.1f,  0.0f, 1.0f, 0.0f,
        -0.1f, 0.2f, 0.1f,  0.0f, 1.0f, 0.0f,
         0.1f, 0.2f, 0.1f,  0.0f, 1.0f, 0.0f,
         0.1f, 0.4f, 0.1f,  0.0f, 1.0f, 0.0f,
        -0.3f, 0.2f, 0.1f,  0.0f, 1.0f, 0.0f,
        -0.3f, 0.2f, 0.1f,  0.0f, 1.0f, 0.0f,
        -0.3f, 0.4f, 0.1f,  0.0f, 1.0f, 0.0f,
         0.1f, 0.4f, 0.1f,  0.0f, 1.0f, 0.0f,
        -0.1f,-0.2f, 0.1f,  0.0f, 1.0f, 0.0f,
        -0.1f,-0.4f, 0.1f,  0.0f, 1.0f, 0.0f,
         0.3f,-0.2f, 0.1f,  0.0f, 1.0f, 0.0f,
         0.3f,-0.2f, 0.1f,  0.0f, 1.0f, 0.0f,
         0.3f,-0.4f, 0.1f,  0.0f, 1.0f, 0.0f,
        -0.1f,-0.4f, 0.1f,  0.0f, 1.0f, 0.0f};
MeshColor* letra_O = new MeshColor();
letra_O->CreateMeshColor(vertices_letra_O, 144);
meshColorList.push_back(letra_O);
```

```
GLfloat vertices_letra_E[] = {//Letra E
    //    X     Y     Z      R     G     B
        -0.3f,-0.4f, 0.1f,  1.0f, 0.0f, 0.0f,
        -0.3f, 0.4f, 0.1f,  1.0f, 0.0f, 0.0f,
        -0.1f, 0.4f, 0.1f,  1.0f, 0.0f, 0.0f,
        -0.1f,-0.4f, 0.1f,  1.0f, 0.0f, 0.0f,
        -0.1f, 0.4f, 0.1f,  1.0f, 0.0f, 0.0f,
        -0.3f,-0.4f, 0.1f,  1.0f, 0.0f, 0.0f,
        -0.1f, 0.2f, 0.1f,  1.0f, 0.0f, 0.0f,
        -0.1f, 0.4f, 0.1f,  1.0f, 0.0f, 0.0f,
         0.3f, 0.4f, 0.1f,  1.0f, 0.0f, 0.0f,
        -0.1f, 0.2f, 0.1f,  1.0f, 0.0f, 0.0f,
         0.3f, 0.2f, 0.1f,  1.0f, 0.0f, 0.0f,
         0.3f, 0.4f, 0.1f,  1.0f, 0.0f, 0.0f,
        -0.1f,-0.2f, 0.1f,  1.0f, 0.0f, 0.0f,
        -0.1f,-0.4f, 0.1f,  1.0f, 0.0f, 0.0f,
         0.3f,-0.4f, 0.1f,  1.0f, 0.0f, 0.0f,
        -0.1f,-0.2f, 0.1f,  1.0f, 0.0f, 0.0f,
         0.3f,-0.2f, 0.1f,  1.0f, 0.0f, 0.0f,
         0.3f,-0.4f, 0.1f,  1.0f, 0.0f, 0.0f,
        -0.1f,-0.1f, 0.1f,  1.0f, 0.0f, 0.0f,
         0.3f,-0.1f, 0.1f,  1.0f, 0.0f, 0.0f,
         0.3f, 0.1f, 0.1f,  1.0f, 0.0f, 0.0f,
        -0.1f, 0.1f, 0.1f,  1.0f, 0.0f, 0.0f,
        -0.1f,-0.1f, 0.1f,  1.0f, 0.0f, 0.0f,
         0.3f, 0.1f, 0.1f,  1.0f, 0.0f, 0.0f};
MeshColor* letra_E = new MeshColor();
letra_E->CreateMeshColor(vertices_letra_E, 144);
meshColorList.push_back(letra_E);
```

Fig. 3. y Fig. 4. *Código de triángulos de la letra O y E.*

```
GLfloat vertices_letra_Y[] = {//Letra Y
    //    X     Y     Z      R     G     B
         0.1f, 0.0f, 0.1f,  0.6f, 0.0f, 0.6f,
        -0.1f, 0.0f, 0.1f,  0.6f, 0.0f, 0.6f,
         0.1f,-0.4f, 0.1f,  0.6f, 0.0f, 0.6f,
        -0.1f,-0.4f, 0.1f,  0.6f, 0.0f, 0.6f,
        -0.1f, 0.0f, 0.1f,  0.6f, 0.0f, 0.6f,
         0.1f,-0.4f, 0.1f,  0.6f, 0.0f, 0.6f,
        -0.1f, 0.0f, 0.1f,  0.6f, 0.0f, 0.6f,
         0.1f, 0.0f, 0.1f,  0.6f, 0.0f, 0.6f,
         0.0f, 0.1f, 0.1f,  0.6f, 0.0f, 0.6f,

        -0.1f, 0.0f, 0.1f,  0.6f, 0.0f, 0.6f,
         0.0f, 0.1f, 0.1f,  0.6f, 0.0f, 0.6f,
        -0.4f, 0.3f, 0.1f,  0.6f, 0.0f, 0.6f,
         0.0f, 0.1f, 0.1f,  0.6f, 0.0f, 0.6f,
        -0.3f, 0.4f, 0.1f,  0.6f, 0.0f, 0.6f,
        -0.4f, 0.3f, 0.1f,  0.6f, 0.0f, 0.6f,

         0.1f, 0.0f, 0.1f,  0.6f, 0.0f, 0.6f,
         0.0f, 0.1f, 0.1f,  0.6f, 0.0f, 0.6f,
         0.4f, 0.3f, 0.1f,  0.6f, 0.0f, 0.6f,
         0.0f, 0.1f, 0.1f,  0.6f, 0.0f, 0.6f,
         0.3f, 0.4f, 0.1f,  0.6f, 0.0f, 0.6f,
         0.4f, 0.3f, 0.1f,  0.6f, 0.0f, 0.6f};
MeshColor* letra_Y = new MeshColor();
letra_Y->CreateMeshColor(vertices_letra_Y, 126);
meshColorList.push_back(letra_Y);
```

Fig. 5. *Código de triángulos de la letra Y.*

Así mismo a cada letra le asigne un color diferente, la U de color azul, la O de verde, la E de rojo, y la Y de color morado. Una vez creadas todas las letras, definí la función para estas en la main.

```
CrearLetras();
CreateShaders();
```

Fig. 6. *Llamado de función para crear las iniciales.*

Para poder ver mejor las letras escogí la proyección ortogonal que ya que aún no estoy empleando elementos en 3D, entonces esta proyección nos permitirá visualizar correctamente las letras en 2D.

```
glm::mat4 projection = glm::ortho(-2.0f, 2.0f, -2.0f, 2.0f, 0.1f, 100.0f);
```

Fig. 7. *Definición de proyección ortogonal.*

Definí en el main la proyección de cada letra con su respectivo desplazamiento, cada una en una sección de la ventana dividida en 4, de acuerdo con el boceto de un inicio, en realidad los números de cada coordenada para su posicion son los mismos, salvo por los signos, además el shader definido para el color es el mismo para todos, el *"shadercolor"* que dio el profesor.

```
shaderList[0].useShader();
uniformModel = shaderList[1].getModelLocation();
uniformProjection = shaderList[1].getProjectLocation();

model = glm::mat4(1.0);// Ubicacion de Letra U
model = glm::translate(model, glm::vec3(-0.5f, 0.5f, -4.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[0]->RenderMeshColor();

model = glm::mat4(1.0);// Ubicacion de Letra O
model = glm::translate(model, glm::vec3(0.5f, 0.5f, -4.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[1]->RenderMeshColor();

model = glm::mat4(1.0);// Ubicacion de Letra E
model = glm::translate(model, glm::vec3(-0.5f,-0.5f, -4.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[2]->RenderMeshColor();

model = glm::mat4(1.0);// Ubicacion de Letra Y
model = glm::translate(model, glm::vec3(0.5f,-0.5f, -4.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[3]->RenderMeshColor();
```

Fig. 8. *Desplazamiento de letras.*

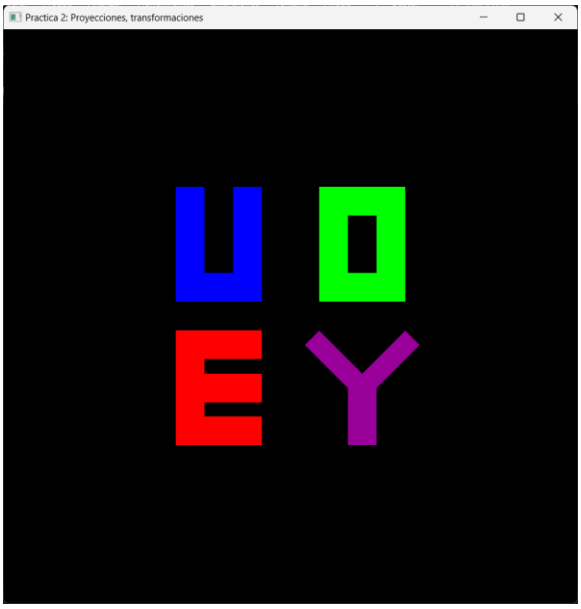

Fig. 9. *Ejecución del código.*

2) *Generar el dibujo de la casa de la clase, pero en lugar de instanciar triángulos y cuadrados será instanciando pirámides y cubos, para esto se requiere crear shaders diferentes de los colores: rojo, verde, azul, café y verde oscuro en lugar de usar el shader con el color clamp.*

Para comenzar con el desarrollo de esta actividad era necesario crear los "shaders" de cada color con sus 2 respectivos archivos, *".flag".* y *".vert".* , tome como base los que se dieron en un inicio con el código, la única modificación realmente fue la asignación de color a cada uno.

```
#version 330
in vec4 vColor;
out vec4 color;
void main()
{
    color= vColor;
}
```

Fig. 10. *Código general de archivos de color ".flag".*

```
#version 330
layout (location =0) in vec3 pos;
out vec4 vColor;
uniform mat4 model;
uniform mat4 projection;
void main(){
    gl_Position=projection*model*vec4(pos,1.0f);
    //vColor=vec4(color,1.0f);
    vColor=vec4(0.0f,0.0f,1.0f,1.0f);}
```

```
#version 330
layout (location =0) in vec3 pos;
out vec4 vColor;
uniform mat4 model;
uniform mat4 projection;
void main(){
    gl_Position=projection*model*vec4(pos,1.0f);
    //vColor=vec4(color,1.0f);
    vColor=vec4(1.0f,0.0f,0.0f,1.0f);}
```

Fig. 11. y Fig. 12. *Código de archivos de color azul y rojo ".vert".*

```
#version 330
layout (location =0) in vec3 pos;
out vec4 vColor;
uniform mat4 model;
uniform mat4 projection;
void main(){
    gl_Position=projection*model*vec4(pos,1.0f);
    //vColor=vec4(color,1.0f);
    vColor=vec4(0.0f,1.0f,0.0f,1.0f);}
```

```
#version 330
layout (location =0) in vec3 pos;
out vec4 vColor;
uniform mat4 model;
uniform mat4 projection;
void main(){
    gl_Position=projection*model*vec4(pos,1.0f);
    //vColor=vec4(color,1.0f);
    vColor=vec4(0.0f,0.5f,0.0f,1.0f);}
```

Fig. 13. y Fig. 14. *Código de archivos de color verde claro y fuerte ".vert".*

```
#version 330
layout (location =0) in vec3 pos;
out vec4 vColor;
uniform mat4 model;
uniform mat4 projection;
void main(){
    gl_Position=projection*model*vec4(pos,1.0f);
    //vColor=vec4(color,1.0f);
    vColor=vec4(0.478f, 0.255f, 0.067f,1.0f);}
```

Fig. 15. *Código de archivo de color café ".vert".*

Una vez creados todos los colores, los definí en el archivo principal *"P02-318234757.cpp".,* asignándolos como contantes de tipo static al inicio con un nombre de acuerdo con el tipo de archivo y el color que tiene, para posteriormente usarlos en la proyección.

```
static const char* vshader_rojo = "shaders/shader_rojo.vert";
static const char* fshader_rojo = "shaders/shader_rojo.frag";
static const char* vshader_azul = "shaders/shader_azul.vert";
static const char* fshader_azul = "shaders/shader_azul.frag";
static const char* vshader_verde01 = "shaders/shader_verde01.vert";
static const char* fshader_verde01 = "shaders/shader_verde01.frag";
static const char* vshader_verde02 = "shaders/shader_verde02.vert";
static const char* fshader_verde02 = "shaders/shader_verde02.frag";
static const char* vshader_cafe = "shaders/shader_cafe.vert";
static const char* fshader_cafe = "shaders/shader_cafe.frag";
```

Fig. 16. *Shaders de los colores para cada prisma.*

Para definir cada figura emplee como base las 2 figuras proporcionadas en un inicio, pero para analizar mejor los vértices utilice Geogebra 3D, desafortunadamente no tome capturas de su uso, para el cubo principal rojo de la casa use la misma función del cubo proporcionada.

```
void Casa() {
    unsigned int cubo_indices[] = {
        //  Cara Frontal    Cara Trasera
              0, 1, 2,         7, 6, 5,
              2, 3, 0,         5, 4, 7,
        // Cara Izquierda   Cara Derecha
              1, 5, 6,         4, 0, 3,
              6, 2, 1,         3, 7, 4,
        // Tapa Superior    Tapa Infetior
              4, 5, 1,         3, 2, 6,
              1, 0, 4,         6, 7, 3 };
    GLfloat cubo_vertices[] = {
        // Uniones frontales
            -0.5f, -0.5f,  0.5f,  0.5f, -0.5f, 0.5f,
             0.5f,  0.5f,  0.5f, -0.5f,  0.5f, 0.5f,
         // Uniones traseras
            -0.5f, -0.5f, -0.5f,  0.5f, -0.5f, -0.5f,
             0.5f,  0.5f, -0.5f, -0.5f,  0.5f, -0.5f };
    Mesh* casa = new Mesh();
    casa->CreateMesh(cubo_vertices, cubo_indices, 24, 36);
    meshList.push_back(casa);
}
```

Fig. 17. *Definición de Cubo principal.*

```
void Puerta() {
    unsigned int cubo_indices[] = {
        //  Cara Frontal    Cara Trasera
              0, 1, 2,         7, 6, 5,
              2, 3, 0,         5, 4, 7,
        // Cara Izquierda   Cara Derecha
              1, 5, 6,         4, 0, 3,
              6, 2, 1,         3, 7, 4,
        // Tapa Superior    Tapa Infetior
              4, 5, 1,         3, 2, 6,
              1, 0, 4,         6, 7, 3 };
    GLfloat cubo_vertices[] = {
        // Uniones frontales
            -0.15f, -0.5f,  0.6f,  0.15f, -0.5f, 0.6f,
             0.15f, -0.2f,  0.6f, -0.15f, -0.2f, 0.6f,
        // Uniones traseras
            -0.15f, -0.5f,  0.3f,  0.15f, -0.5f,  0.3f,
             0.15f, -0.2f,  0.3f, -0.15f, -0.2f,  0.3f };
    Mesh* puerta = new Mesh();
    puerta->CreateMesh(cubo_vertices, cubo_indices, 24, 36);
    meshList.push_back(puerta);
}
```

Fig. 18. *Definición de Cubo para puerta.*

Para las puertas y ventanas utilice el mismo codigo, con la diferencia de que las dimensiones eran para un cubo de 3X3X3, el cubo de la puerta ubicado en la parte inferior del rojo y las ventanas mas arriba casi tocando la cara superior del cubo rojo. Además, en el eje de Z se encontraban mas adelante, por una diferencia de 0.1f a la del cubo rojo, esto para que fueran visibles.

```
void Ventana01() {
    unsigned int cubo_indices[] = {
        //  Cara Frontal    Cara Trasera
              0, 1, 2,         7, 6, 5,
              2, 3, 0,         5, 4, 7,
        // Cara Izquierda   Cara Derecha
              1, 5, 6,         4, 0, 3,
              6, 2, 1,         3, 7, 4,
        // Tapa Superior    Tapa Infetior
              4, 5, 1,         3, 2, 6,
              1, 0, 4,         6, 7, 3 };
    GLfloat cubo_vertices[] = {
        // Uniones frontales
            -0.4f, 0.1f, 0.6f, -0.1f, 0.1f, 0.6f,
            -0.1f, 0.4f, 0.6f, -0.4f, 0.4f, 0.6f,
        // Uniones traseras
            -0.4f, 0.1f, 0.3f, -0.1f, 0.1f, 0.3f,
            -0.1f, 0.4f, 0.3f, -0.4f, 0.4f, 0.3f };
    Mesh* ventana1 = new Mesh();
    ventana1->CreateMesh(cubo_vertices, cubo_indices, 24, 36);
    meshList.push_back(ventana1);
}
```

```
void Ventana02() {
    unsigned int cubo_indices[] = {
        //  Cara Frontal    Cara Trasera
              0, 1, 2,         7, 6, 5,
              2, 3, 0,         5, 4, 7,
        // Cara Izquierda   Cara Derecha
              1, 5, 6,         4, 0, 3,
              6, 2, 1,         3, 7, 4,
        // Tapa Superior    Tapa Infetior
              4, 5, 1,         3, 2, 6,
              1, 0, 4,         6, 7, 3 };
    GLfloat cubo_vertices[] = {
        // Uniones frontales
            0.4f, 0.1f, 0.6f, 0.1f, 0.1f, 0.6f,
            0.1f, 0.4f, 0.6f, 0.4f, 0.4f, 0.6f,
        // Uniones traseras
            0.4f, 0.1f, 0.3f, 0.1f, 0.1f, 0.3f,
            0.1f, 0.4f, 0.3f, 0.4f, 0.4f, 0.3f };
    Mesh* ventana2 = new Mesh();
    ventana2->CreateMesh(cubo_vertices, cubo_indices, 24, 36);
    meshList.push_back(ventana2);
}
```

Fig. 19. y Fig. 20. *Definición de Cubos para Ventanas.*

Para la figura del techo tuve que crear una nueva figura, la cual sería una pirámide cuadrangular, ya que la pirámide proporcionada era una pirámide triangular, la modificación que se le hizo a la función de la pirámide fue agregarle una línea de vértices, ya que la base debía ser cuadrada, formada por 2 triángulos. Así mismo en los índices se agregaron 2 nuevos que forman la base.

```
void Techo() {
    unsigned int indices[] = {
        // Base
        0, 1, 2,    2, 3, 0,
        // Caras
        0, 4, 1,    1, 4, 2,    2, 4, 3,    3, 4, 0 };
    GLfloat vertices[] = {
        // Base
        -0.5f, 0.5f,  0.5f,  // 0
         0.5f, 0.5f,  0.5f,  // 1
         0.5f, 0.5f, -0.5f,  // 2
        -0.5f, 0.5f, -0.5f,  // 3
        // Altura
         0.0f, 1.0f,  0.0f   // 4
    };
    Mesh* techo = new Mesh();
    techo->CreateMesh(vertices, indices, 15, 18);
    meshList.push_back(techo);
}
```

Fig. 21. *Definición de pirámide para techo.*

Para los troncos de los árboles utilicé el mismo modelo de la puerta, pero nuevamente con modificaciones, lo principal fue hacerlos aun mas pequeños por lo tanto las dimensiones que pensé eran ahora de 2X2X2, en esta ocasión con respecto al eje Z ambas estarían más atrás y a los extremos de la casa, creando uno primero fue fácil hacer el otro, ya que como se encontraba al lado contrario de la casa solo cambié los signos del eje X.

```
void Tronco01() {//Cubo.
    unsigned int cubo_indices[] = {
        //  Cara Frontal    Cara Trasera
            0, 1, 2,         7, 6, 5,
            2, 3, 0,         5, 4, 7,
        // Cara Izquierda   Cara Derecha
            1, 5, 6,         4, 0, 3,
            6, 2, 1,         3, 7, 4,
        // Tapa Superior    Tapa Infetior
            4, 5, 1,         3, 2, 6,
           1, 0, 4,          6, 7, 3 };
    GLfloat cubo_vertices[] = {
        // Uniones frontales
           -0.9f, -0.5f,  0.1f, -0.7f, -0.5f,  0.1f,
           -0.7f, -0.3f,  0.1f, -0.9f, -0.3f,  0.1f,
        // Uniones traseras
           -0.9f, -0.5f, -0.1f, -0.7f, -0.5f, -0.1f,
           -0.7f, -0.3f, -0.1f, -0.9f, -0.3f, -0.1f };
    Mesh* tronco01 = new Mesh();
    tronco01->CreateMesh(cubo_vertices, cubo_indices, 24, 36);
    meshList.push_back(tronco01);
}
```

```
void Tronco02() {//Cubo.
    unsigned int cubo_indices[] = {
        //  Cara Frontal    Cara Trasera
            0, 1, 2,         7, 6, 5,
            2, 3, 0,         5, 4, 7,
        // Cara Izquierda   Cara Derecha
            1, 5, 6,         4, 0, 3,
            6, 2, 1,         3, 7, 4,
        // Tapa Superior    Tapa Infetior
            4, 5, 1,         3, 2, 6,
            1, 0, 4,         6, 7, 3 };
    GLfloat cubo_vertices[] = {
        // Uniones frontales
            0.9f, -0.5f,  0.1f,  0.7f, -0.5f,  0.1f,
            0.7f, -0.3f,  0.1f,  0.9f, -0.3f,  0.1f,
        // Uniones traseras
            0.9f, -0.5f, -0.1f,  0.7f, -0.5f, -0.1f,
            0.7f, -0.3f, -0.1f,  0.9f, -0.3f, -0.1f };
    Mesh* tronco02 = new Mesh();
    tronco02->CreateMesh(cubo_vertices, cubo_indices, 24, 36);
    meshList.push_back(tronco02);
}
```

Fig. 22. y Fig. 23. *Definición de cubo para troncos de árboles.*

Para hacer la parte de las hojas de los arboles utilice nuevamente la pirámide que utilice para el techo, y al igual que los troncos, creando uno primero, el otro solo requirió modificar los signos del eje X.

```
void Arbol01() {
    unsigned int indices[] = {
        // Base
        0, 1, 2,    2, 3, 0,
        // Caras
        0, 4, 1,    1, 4, 2,    2, 4, 3,    3, 4, 0 };
    GLfloat vertices[] = {
        // Base
         1.0f,-0.3f, 0.3f,  // 0
         0.6f,-0.3f, 0.3f,  // 1
         0.6f,-0.3f,-0.3f,  // 2
         1.0f,-0.3f,-0.3f,  // 3
        // Altura
         0.8f, 0.3f,  0.0f   // 4
    };
    Mesh* arbol01 = new Mesh();
    arbol01->CreateMesh(vertices, indices, 15, 18);
    meshList.push_back(arbol01);
}
```

```
void Arbol02() {
    unsigned int indices[] = {
        // Base
        0, 1, 2,    2, 3, 0,
        // Caras
        0, 4, 1,    1, 4, 2,    2, 4, 3,    3, 4, 0 };
    GLfloat vertices[] = {
        // Base
        -1.0f,-0.3f, 0.3f,  // 0
        -0.6f,-0.3f, 0.3f,  // 1
        -0.6f,-0.3f,-0.3f,  // 2
        -1.0f,-0.3f,-0.3f,  // 3
        // Altura
         -0.8f, 0.3f,  0.0f   // 4
    };
    Mesh* arbol02 = new Mesh();
    arbol02->CreateMesh(vertices, indices, 15, 18);
    meshList.push_back(arbol02);
}
```

Fig. 24. y Fig. 25. *Definición de pirámides para troncos de árboles.*

Una vez definidas todas las figuras utilizadas agregue a la función *“CreateShaders”* la asignación de los nuevos shaders para cada color, copie de referencia los shaders originales, solo le cambie el nombre *“shade1”* por el numero siguiente y cambie los nombres de referencia en la parte de *“CreateFromFiles”,* con esas modificaciones agregue correctamente el uso de cada color para las figuras.

En la función principal main agregue las referencias a las llamadas de las demás figuras en un inicio, además de redimensionar la pantalla para que se vieran mejor las figuras. Otro cambio importante fue poner como comentario la proyección ortogonal y quitar como comentario la proyección en perspectiva para así ver mejor las figuras en 3D.

Por ultimo copie y pegue las instrucciones para imprimir cada figura con su respectivo color, la única modificación en cada una es el valor en *“shaderList[]”* para el color y *“meshList[]”* para el tipo de figura, de acuerdo al orden asignado.

```
void CreateShaders(){
    Shader *shader1 = new Shader(); //Color como parte del VAO: letras
    shader1->CreateFromFiles(vShaderColor, fShaderColor);
    shaderList.push_back(*shader1);

    Shader *shader2 = new Shader();//Color rojo.
    shader2->CreateFromFiles(vshader_rojo, fshader_rojo);
    shaderList.push_back(*shader2);

    Shader* shader3 = new Shader();//Color azul.
    shader3->CreateFromFiles(vshader_azul, fshader_azul);
    shaderList.push_back(*shader3);

    Shader* shader4 = new Shader();//Color verde claro.
    shader4->CreateFromFiles(vshader_verde01, fshader_verde01);
    shaderList.push_back(*shader4);

    Shader* shader5 = new Shader();//Color verde fuerte.
    shader5->CreateFromFiles(vshader_verde02, fshader_verde02);
    shaderList.push_back(*shader5);

    Shader* shader6 = new Shader();//Color cafe.
    shader6->CreateFromFiles(vshader_cafe, fshader_cafe);
    shaderList.push_back(*shader6);
}
```

Fig. 26. *Definición de colores con archivos shaders.*

```
mainWindow = Window(900, 900);
mainWindow.Initialise();

Casa();
Puerta();
Ventana01();
Ventana02();
Techo();
Arbol01();
Arbol02();
Tronco01();
Tronco02();
```

Fig. 27. *Definición de funciones de figuras en main.*

```
glm::mat4 projection = glm::perspective(glm::radians(60.0f),mainWindow.getBufferWidth() / mainWindow.getBufferHeight(), 0.1f, 100.0f);
```

Fig. 28. *Definición de proyección de perspectiva.*

Como modificación extra se agregó la línea de codigo *“model = glm::rotate …”* en cada llamada a imprimir de las figuras con el fin de que estas rotaran a la misma velocidad y se apreciaran mejor las figuras 3D, rotando en el eje Y.

```
// Casa
shaderList[1].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
angulo += 0.01;

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -3.0f));
model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 0.1f, 0.0f));//Rotar la figura.
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();

// Puerta
shaderList[3].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
angulo += 0.01;

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -3.0f));
model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 0.1f, 0.0f));//Rotar la figura.
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

Fig. 29. *Selección de color para cada figura.*

```
// Ventana 1 y 2
shaderList[3].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
angulo += 0.01;

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -3.0f));
model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 0.1f, 0.0f));//Rotar la figura.
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[2]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -3.0f));
model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 0.1f, 0.0f));//Rotar la figura.
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[3]->RenderMesh();
```

Fig. 30. *Selección de color para cada figura.*

```
// Techo
shaderList[2].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
angulo += 0.01;

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -3.0f));
model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 0.1f, 0.0f));//Rotar la figura.
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[4]->RenderMesh();

// Arbol 1 y 2
shaderList[4].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
angulo += 0.01;

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -3.0f));
model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 0.1f, 0.0f));//Rotar la figura.
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[5]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -3.0f));
model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 0.1f, 0.0f));//Rotar la figura.
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[6]->RenderMesh();
```

Fig. 31. *Selección de color para cada figura.*

```
// Tronco 1 y 2
shaderList[5].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
angulo += 0.01;

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -3.0f));
model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 0.1f, 0.0f));//Rotar la figura.
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[7]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -3.0f));
model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 0.1f, 0.0f));//Rotar la figura.
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[8]->RenderMesh();
```

Fig. 32. *Selección de color para cada figura.*

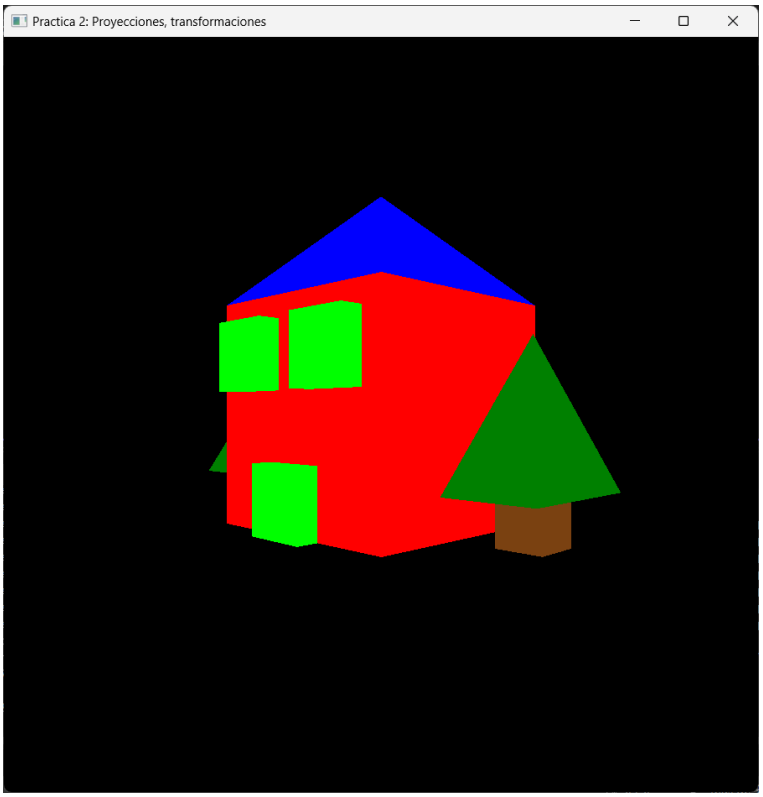

Fig. 33. Ejecución del código.

Fig. 34. *Ejecución del código.*

Fig. 35. *Ejecución del código.*

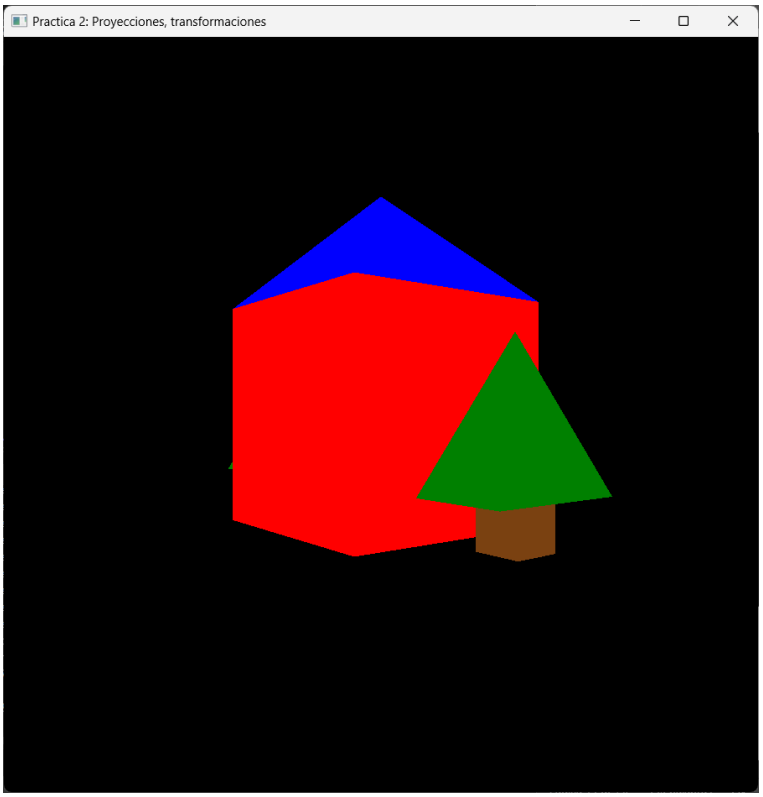

Fig. 36. *Ejecución del código.*

Por default ambas actividades de ejecutaron separadas pero en una ejecución al mismo tiempo se vería así.

Fig. 37. *Ejecución de ambos ejercicios.*

2. *Problemas presentados.*

Para el primer ejercicio de la práctica no tuve ningún problema, pero para la segunda actividad si presenté algunas complicaciones.
Fue complicado entender la definición de las coordenadas en las 2 figuras originales, además de no entender en un inicio hacía que dirección se encontraba el eje Z en la proyección de perspectiva, pero con ayuda de geogebra 3D fue más fácil entender hacia donde se encontraban cada punto y como se unían. Para la definición de los shaders no entendía exactamente como se implementaban, pero después de realizar una análisis aproximado de 1 hora por completo a los códigos, el cpp principal, los .vert y .frag, entendí las especificaciones en cada uno para modificarlos y generar los demás colores.
Como dato extra, un error que se me presento en la practica fue cuando intente volver a ejecutar el codigo cuando este ya estaba corriendo, no entendía muy bien por que marcaba error al compilar cuando las modificaciones que realice no eran más que cambios en la posición de los vértices, después de unos 20 minutos analizando el codigo modificado me di cuenta de eso, fue un error mínimo pero me pareció curioso observar como el programa no permite ejecutar múltiples compilaciones, supongo es una forma de proteger el equipo y no exigirle tanto trabajo al mismo tiempo.

3. *Conclusión:*
   Tras realizar las actividades de esta practica debo comentar que me gusto mucho implementar otra forma de ejecución en el curso, me refiero a la implementación de proyección en perspectiva para las figuras en el espacio 3D, en cuanto a la complejidad de los ejercicios creo que si existió un cambio entra la actividad 1 y 2, ya que la primera aun se relacionaba o asemejaba a las actividades de la practica anterior de permanecer en 2D, mientras que la segunda ya era realizarlas en el espacio 3D, algo nuevo para el curso, en general la explicación de la clase creo que fue suficiente para entender en general el codigo, excepto por la parte de los shaders que no me quedo del todo clara. A final considero que esta práctica cumplió su objetivo de presentarnos y analizar las dimensiones que tienen las figuras en 3D como apoyo en la perspectiva del proyecto en el futuro.

4. *Bibliografía.*

   - *No empleada*