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: MORENO SANTOYO MARIANA

Nº de Cuenta: 319170252

GRUPO DE LABORATORIO: 11

GRUPO DE TEORÍA: 04

SEMESTRE 2025-2

FECHA DE ENTREGA LÍMITE: 26/02/2025

CALIFICACIÓN: _____

REPORTE DE PRÁCTICA:

1. *Ejecución de los ejercicios que se dejaron, comentar cada uno y capturas de pantalla de bloques de código generados y de ejecución del programa.*
2. *Liste los problemas que tuvo a la hora de hacer estos ejercicios y si los resolvió explicar cómo fue, en caso de error adjuntar captura de pantalla*
3. *Conclusión:*
    a. *Los ejercicios del reporte: Complejidad, Explicación.*
    b. *Comentarios generales: Faltó explicar a detalle, ir más lento en alguna explicación, otros comentarios y sugerencias para mejorar desarrollo de la práctica*
    c. *Conclusión*
4. *Bibliografía en formato APA*

EJERCICIOS:

- ***Dibujar las iniciales de sus nombres, cada letra de un color diferente***

Para realizar las letras en diferentes colores, como lo especifica el ejercicio, y dado que en el enunciado no se menciona el uso obligatorio de algún tipo de índice para llevarlo a cabo, opté por buscar en línea los valores RGB de los colores que quería utilizar. Posteriormente, investigué cómo normalizarlos para que fueran compatibles con el formato de OpenGL.

Luego, retomé los vértices de las letras de mi archivo main de la práctica anterior y los incorporé, asignándoles sus respectivos valores RGB.

LETRA M :

```
GLfloat vertices_letras[] = {
//LETRA M
    -0.95f, -0.5f, 0.0f,1.0f, 0.713f, 0.757f,
    -0.85f, -0.5f, 0.0f,1.0f, 0.713f, 0.757f,
    -0.95f,  0.5f, 0.0f,1.0f, 0.713f, 0.757f,

    -0.95f, 0.5f, 0.0f,1.0f, 0.713f, 0.757f,
    -0.85f, 0.5f, 0.0f,1.0f, 0.713f, 0.757f,
    -0.85f,  -0.5f, 0.0f,1.0f, 0.713f, 0.757f,

     -0.85f, 0.25f, 0.0f,1.0f, 0.713f, 0.757f,
     -0.65f, 0.25f, 0.0f,1.0f, 0.713f, 0.757f,
     -0.85f, 0.5f, 0.0f,1.0f, 0.713f, 0.757f,

     -0.65, 0.25f, 0.0f,1.0f, 0.713f, 0.757f,
     -0.45, 0.25f, 0.0f,1.0f, 0.713f, 0.757f,
     -0.45f, 0.5f, 0.0f,1.0f, 0.713f, 0.757f,

     -0.85f, 0.25f, 0.0f,1.0f, 0.713f, 0.757f,
     -0.45f, 0.25f, 0.0f,1.0f, 0.713f, 0.757f,
     -0.65f, 0.0f, 0.0f,1.0f, 0.713f, 0.757f,

    -0.45f, -0.5f, 0.0f,1.0f, 0.713f, 0.757f,
    -0.35f, -0.5f, 0.0f,1.0f, 0.713f, 0.757f,
    -0.35f,  0.5f, 0.0f,1.0f, 0.713f, 0.757f,

    -0.45f, 0.5f, 0.0f,1.0f, 0.713f, 0.757f,
    -0.35f, 0.5f, 0.0f,1.0f, 0.713f, 0.757f,
    -0.45f,  -0.5f, 0.0f,1.0f, 0.713f, 0.757f

};
MeshColor* letras = new MeshColor();
letras->CreateMeshColor(vertices_letras,126);
meshColorList.push_back(letras);
```

LETRA A :

```
GLfloat vertices_letrasA[] = {
    //LETRA A

    -0.3f, -0.5f, 0.0f,0.73f, 0.57f, 0.87f,
    -0.2f, -0.5f, 0.0f,0.73f, 0.57f, 0.87f,
    -0.2f, -0.2f, 0.0f,0.73f, 0.57f, 0.87f,
    -0.2f, -0.2f, 0.0f,0.73f, 0.57f, 0.87f,
    -0.1f, -0.2f, 0.0f,0.73f, 0.57f, 0.87f,
    -0.2f, -0.5f, 0.0f,0.73f, 0.57f, 0.87f,

    -0.2f, -0.2f, 0.0f,0.73f, 0.57f, 0.87f,
    -0.1, -0.2f, 0.0f,0.73f, 0.57f, 0.87f,
    -0.1f,0.1f, 0.0f,0.73f, 0.57f, 0.87f,

    -0.1, 0.1f, 0.0f,0.73f, 0.57f, 0.87f,
    0.0, 0.1f, 0.0f,0.73f, 0.57f, 0.87f,
    -0.1f, -0.2f, 0.0f,0.73f, 0.57f, 0.87f,
    //pico
    -0.1f, 0.1f, 0.0f,0.73f, 0.57f, 0.87f,
    0.0f, 0.1f, 0.0f,0.73f, 0.57f, 0.87f,
    0.0f, 0.5f, 0.0f,0.73f, 0.57f, 0.87f,
    //pico
    0.0f, 0.1f, 0.0f,0.73f, 0.57f, 0.87f,
    0.1f, 0.1f, 0.0f,0.73f, 0.57f, 0.87f,
    0.0f,  0.5f, 0.0f,0.73f, 0.57f, 0.87f,

    0.1f, 0.1f, 0.0f,0.73f, 0.57f, 0.87f,
    0.0f, 0.1f, 0.0f,0.73f, 0.57f, 0.87f,
    0.1f,  -0.2f, 0.0f,0.73f, 0.57f, 0.87f,

    0.2f, -0.2f, 0.0f,0.73f, 0.57f, 0.87f,
    0.1, -0.2f, 0.0f, 0.73f, 0.57f, 0.87f,
    0.1f,0.1f, 0.0f,0.73f, 0.57f, 0.87f,

    0.2f, -0.2f, 0.0f,0.73f, 0.57f, 0.87f,
    0.1f, -0.2f, 0.0f,0.73f, 0.57f, 0.87f,
    0.2f, -0.5f, 0.0f,0.73f, 0.57f, 0.87f,

    0.3f, -0.5f, 0.0f,0.73f, 0.57f, 0.87f,
    0.2f, -0.5f, 0.0f,0.73f, 0.57f, 0.87f,
    0.2f, -0.2f, 0.0f,0.73f, 0.57f, 0.87f,

    0.0f, -0.3f, 0.0f,0.73f, 0.57f, 0.87f,
    0.15f, -0.3f, 0.0f,0.73f, 0.57f, 0.87f,
    0.0f, -0.2f, 0.0f,0.73f, 0.57f, 0.87f,

    0.0f, -0.2f, 0.0f,0.73f, 0.57f, 0.87f,
    0.15f, -0.2f, 0.0f,0.73f, 0.57f, 0.87f,
    0.15f, -0.3f, 0.0f,0.73f, 0.57f, 0.87f,

    0.0f, -0.3f, 0.0f,0.73f, 0.57f, 0.87f,
    -0.15f, -0.3f, 0.0f,0.73f, 0.57f, 0.87f,
    0.0f, -0.2f, 0.0f,0.73f, 0.57f, 0.87f,

    0.0f, -0.2f, 0.0f,0.73f, 0.57f, 0.87f,
    -0.15f, -0.2f, 0.0f,0.73f, 0.57f, 0.87f,
    -0.15f, -0.3f, 0.0f,0.73f, 0.57f, 0.87f,
};
MeshColor* letrasA = new MeshColor();
letrasA->CreateMeshColor(vertices_letrasA, 252);
meshColorList.push_back(letrasA);
```

LETRA S

```
GLfloat vertices_letrasS[] = {

    0.35f, -0.3f, 0.0f,0.68f, 0.85f, 0.90f,
    0.55f, -0.3f, 0.0f,0.68f, 0.85f, 0.90f,
    0.35f,  -0.1f, 0.0f,0.68f, 0.85f, 0.90f,

    0.35f, -0.3f, 0.0f,0.68f, 0.85f, 0.90f,
    0.55f, -0.3f, 0.0f,0.68f, 0.85f, 0.90f,
    0.55f,  -0.5f, 0.0f,0.68f, 0.85f, 0.90f,

    0.55f, -0.3f, 0.0f, 0.68f, 0.85f, 0.90f,
    0.75f, -0.3f, 0.0f, 0.68f, 0.85f, 0.90f,
    0.55f,  -0.5f, 0.0f, 0.68f, 0.85f, 0.90f,

    0.55f, -0.5f, 0.0f, 0.68f, 0.85f, 0.90f,
    0.75f, -0.5f, 0.0f, 0.68f, 0.85f, 0.90f,
    0.75f, -0.3f, 0.0f, 0.68f, 0.85f, 0.90f,

    0.75f, -0.3f, 0.0f, 0.68f, 0.85f, 0.90f,
    0.95f, -0.3f, 0.0f, 0.68f, 0.85f, 0.90f,
    0.75f, -0.5f, 0.0f, 0.68f, 0.85f, 0.90f,

    0.75f, -0.3f, 0.0f, 0.68f, 0.85f, 0.90f,
    0.95f, -0.3f, 0.0f, 0.68f, 0.85f, 0.90f,
    0.95f, -0.1f, 0.0f, 0.68f, 0.85f, 0.90f,

    0.75f, -0.3f, 0.0f, 0.68f, 0.85f, 0.90f,
    0.95f, -0.1f, 0.0f, 0.68f, 0.85f, 0.90f,
    0.75f, 0.1f, 0.0f, 0.68f, 0.85f, 0.90f,

    0.55f, -0.1f, 0.0f, 0.68f, 0.85f, 0.90f,
    0.75f, -0.3f, 0.0f, 0.68f, 0.85f, 0.90f,
    0.75f, 0.1f, 0.0f, 0.68f, 0.85f, 0.90f,

    0.55f, -0.1f, 0.0f, 0.68f, 0.85f, 0.90f,
    0.75f, 0.1f, 0.0f, 0.68f, 0.85f, 0.90f,
    0.55f, 0.3f, 0.0f, 0.68f, 0.85f, 0.90f,

    0.35f, 0.1f, 0.0f, 0.68f, 0.85f, 0.90f,
    0.55f, -0.1f, 0.0f, 0.68f, 0.85f, 0.90f,
    0.55f, 0.3f, 0.0f, 0.68f, 0.85f, 0.90f,

    0.35f, 0.3f, 0.0f, 0.68f, 0.85f, 0.90f,
    0.55f, 0.3f, 0.0f,0.68f, 0.85f, 0.90f,
    0.35f, 0.1f, 0.0f,0.68f, 0.85f, 0.90f,

    0.35f, 0.3f, 0.0f, 0.68f, 0.85f, 0.90f,
    0.55f, 0.3f, 0.0f, 0.68f, 0.85f, 0.90f,
    0.55f, 0.5f, 0.0f, 0.68f, 0.85f, 0.90f,

    0.55f, 0.3f, 0.0f, 0.68f, 0.85f, 0.90f,
    0.75f, 0.3f, 0.0f,  0.68f, 0.85f, 0.90f,
    0.55f, 0.5f, 0.0f,  0.68f, 0.85f, 0.90f,

    0.55f, 0.5f, 0.0f,  0.68f, 0.85f, 0.90f,
    0.75f, 0.5f, 0.0f, 0.68f, 0.85f, 0.90f,
    0.75f, 0.3f, 0.0f, 0.68f, 0.85f, 0.90f,

    0.75f, 0.3f, 0.0f, 0.68f, 0.85f, 0.90f,
    0.95f, 0.3f, 0.0f,  0.68f, 0.85f, 0.90f,
    0.75f, 0.5f, 0.0f,  0.68f, 0.85f, 0.90f,

    0.75f, 0.3f, 0.0f,  0.68f, 0.85f, 0.90f,
    0.95f, 0.3f, 0.0f,  0.68f, 0.85f, 0.90f,
    0.95f, 0.1f, 0.0f, 0.68f, 0.85f, 0.90f,

};
MeshColor* letrasS = new MeshColor();
letrasS->CreateMeshColor(vertices_letrasS,288);
meshColorList.push_back(letrasS);
```

Posteriormente, para no modificar en gran medida el archivo main, lo único que se hizo fue llamar a las funciones encargadas del renderizado de color tres veces, una por cada arreglo, ya que se optó por crear un arreglo independiente para cada letra. Además, se realizaron ligeras modificaciones en los valores de las transformaciones para evitar que la apariencia de las letras se viera afectada.

RESULTADO

- ***Generar el dibujo de la casa de la clase, pero en lugar de instanciar triangulos 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***

Durante la realización del segundo ejercicio de la práctica, surgió un problema debido a una interpretación errónea de la instrucción. En lugar de generar la vista desde una sola cara, se realizaron vistas desde las cuatro caras diferentes de la casa y los árboles.

Para llevar a cabo este ejercicio, fue necesario crear shaders específicos para cada color utilizado en la escena. En el caso de los árboles, se implementaron shaders con colores verde y café, mientras que para la casa se diseñaron shaders con colores azul, verde y rojo. La creación de estos shaders resultó sencilla, ya que se tomaron como base los previamente utilizados, modificando únicamente los valores RGB correspondientes a cada color.

Posteriormente, los shaders fueron agregados al archivo principal mediante la función crearShader(). A partir de ello, y utilizando transformaciones como escalado y rotación de las figuras, se logró obtener el resultado solicitado. Cabe destacar que las rotaciones pueden visualizarse activando la línea de código rotate, la cual actualmente se encuentra comentada.

SHADERS.FRAG:

rojoshader.frag

```
#version 330 core
out vec4 fragColor;

void main()
{
    fragColor = vec4(1.0, 0.0, 0.0, 1.0); // Color rojo
}

```

shadecafe.frag

```
#version 330 core
out vec4 fragColor;

void main()
{
    fragColor = vec4(0.478, 0.255, 0.067, 1.0); // Color CAFE

}

```

Estas son imágenes explicativas, pero en realidad, todos los shaders creados en los archivos .frag son exactamente iguales. Lo único que cambia es el vector de color, el cual varía dependiendo de los valores RGB asignados a cada elemento de la escena.

SHADERS.VERT:

rojoshader.vert

```
#version 330 core
layout (location = 0) in vec3 pos;

uniform mat4 model;
uniform mat4 projection;

void main()
{
    gl_Position = projection * model * vec4(pos, 1.0);
}

```

shadecafe.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(clamp(pos,0.0f,1.0f),1.0f);
}
```

verde.vert

```
#version 330 core
layout (location = 0) in vec3 pos;

uniform mat4 model;
uniform mat4 projection;

void main()
{
    gl_Position = projection * model * vec4(pos, 1.0);
}

```

De igual manera, estas son imágenes explicativas de los archivos .vert, los cuales son exactamente iguales en la mayoría de los casos.

CAMBIOS MAIN:

Lo primero fue agregar todos los shaders necesarios:

```
static const char* vShader = "shaders/shader.vert";
static const char* fShader = "shaders/shader.frag";
static const char* vShaderColor = "shaders/shadercolor.vert";
static const char* fShaderColor = "shaders/shadercolor.frag";
static const char* vContorno = "shaders/contorno.vert";
static const char* fContorno = "shaders/contorno.frag";
static const char* vRojo = "shaders/rojoshader.vert";
static const char* fRojo = "shaders/rojoshader.frag";
static const char* vAzul = "shaders/shaderazul.vert";
static const char* fAzul = "shaders/shaderazul.frag";
static const char* vCafe = "shaders/shadecafe.vert";
static const char* fCafe = "shaders/shadecafe.frag";
static const char* vVarbol = "shaders/verde.vert";
static const char* fVarbol = "shaders/verde.frag";
static const char* vverde = "shaders/verdefosfo.vert";
static const char* fverde = "shaders/verdefosfo.frag";
```

Los mismos shaders se pasan a la función de crear shader

```
void CreateShaders()
{

    Shader* shader1 = new Shader(); //shader para usar índices: objetos: cubo y  pirámide
    shader1->CreateFromFiles(vShader, fShader);
    shaderList.push_back(*shader1);

    Shader* shader2 = new Shader();//shader para usar color como parte del VAO: letras
    shader2->CreateFromFiles(vShaderColor, fShaderColor);
    shaderList.push_back(*shader2);

    Shader* shader3 = new Shader();//shader para usar color como parte del VAO: letras
    shader3->CreateFromFiles(vContorno, fContorno);
    shaderList.push_back(*shader3);

    Shader* shader4 = new Shader();//shader para usar color como parte del VAO: letras
    shader4->CreateFromFiles(vRojo, fRojo);
    shaderList.push_back(*shader4);

    Shader* shader5 = new Shader();//shader para usar color como parte del VAO: letras
    shader5->CreateFromFiles(vAzul, fAzul);
    shaderList.push_back(*shader5);

    Shader* shader6 = new Shader();//shader para usar color como parte del VAO: letras
    shader6->CreateFromFiles(vCafe, fCafe);
    shaderList.push_back(*shader6);

    Shader* shader7 = new Shader();//shader para usar color como parte del VAO: letras
    shader7->CreateFromFiles(vVarbol, fVarbol);
    shaderList.push_back(*shader7);

    Shader* shader8 = new Shader();//shader para usar color como parte del VAO: letras
    shader8->CreateFromFiles(vverde, fverde);
    shaderList.push_back(*shader8);
}
```

Se crea el cubo rojo

```
//Para el cubo y la pirámide se usa el primer set de shaders con índice 0 en ShaderList
shaderList[3].useShader();
uniformModel = shaderList[3].getModelLocation();
uniformProjection = shaderList[3].getProjectLocation();
angulo += 0.01;

//Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para almacenar las transformaciones geométricas
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.5f, -3.0f));
model = glm::scale(model, glm::vec3(1.75f, 1.75f, 1.75f));
//model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

Se crean los cubos verdes

```
shaderList[7].useShader();
uniformModel = shaderList[7].getModelLocation();
uniformProjection = shaderList[7].getProjectLocation();
angulo += 0.01;

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -1.0f, -2.0f));
model = glm::scale(model, glm::vec3(0.6f, 0.6f, 0.0f));
//model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.4f, -0.1f, -2.0f));
model = glm::scale(model, glm::vec3(0.6f, 0.6f, 0.0f));
//model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.4f, -0.1f, -2.0f));
model = glm::scale(model, glm::vec3(0.6f, 0.6f, 0.0f));
//model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

Se crean las pirámides azules para el techo

```
    shaderList[4].useShader();
    uniformModel = shaderList[4].getModelLocation();
    uniformProjection = shaderList[4].getProjectLocation();
    angulo += 0.01;


    model = glm::mat4(1.0); model = glm::translate(model, glm::vec3(0.0f, 0.9f, -3.0f));
    model = glm::scale(model, glm::vec3(2.0f, 1.0f, 2.0f));
    //model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
//  model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
    glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
    meshList[0]->RenderMesh();

    model = glm::mat4(1.0); model = glm::translate(model, glm::vec3(0.0f, 0.9f, -3.0f));
    model = glm::scale(model, glm::vec3(2.0f, 1.0f, 2.0f));
    model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
//  model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
    glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
    meshList[0]->RenderMesh();


    model = glm::mat4(1.0);
    model = glm::translate(model, glm::vec3(0.0f, 0.9f, -3.0f));
    model = glm::scale(model, glm::vec3(2.0f, 1.0f, 2.0f));
    model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
//  model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
    glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
    meshList[0]->RenderMesh();

    model = glm::mat4(1.0);
    model = glm::translate(model, glm::vec3(0.0f, 0.9f, -3.0f));
    model = glm::scale(model, glm::vec3(2.0f, 1.0f, 2.0f));
    model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
//  model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
    glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
    meshList[0]->RenderMesh();
```

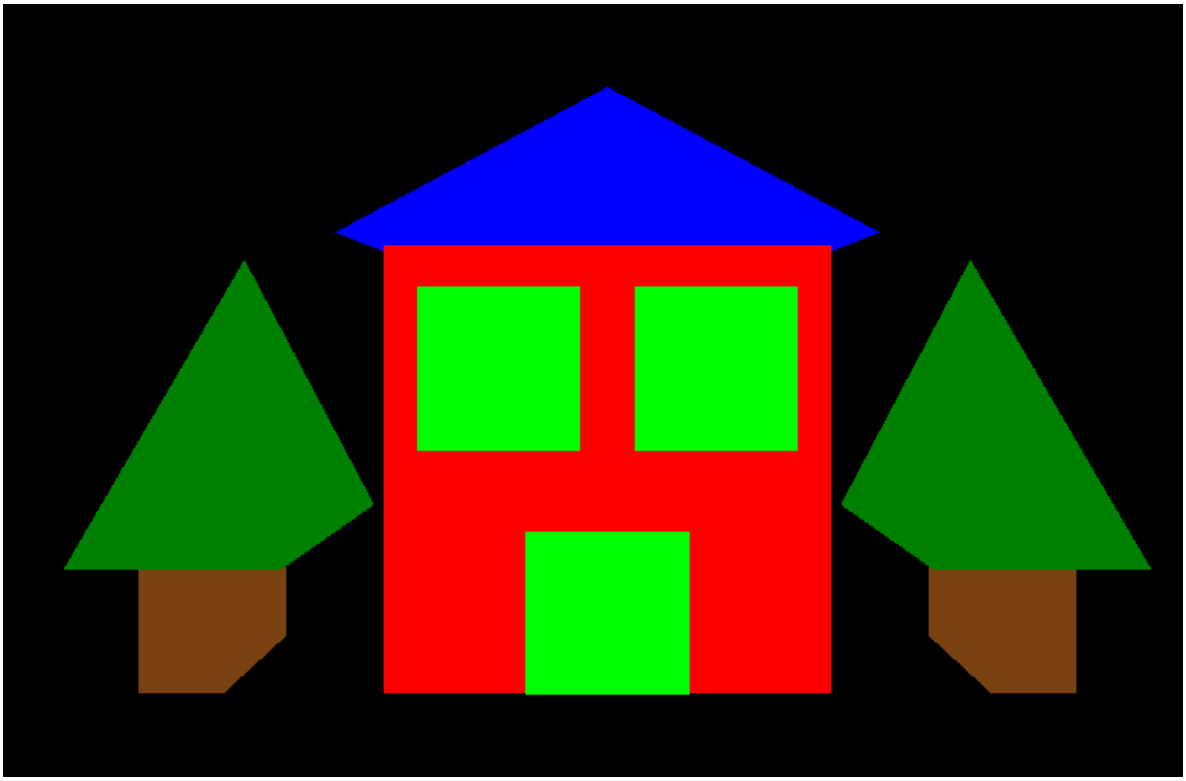
Se crean los cubos cafés para los troncos

```
shaderList[5].useShader();
uniformModel = shaderList[5].getModelLocation();
uniformProjection = shaderList[5].getProjectLocation();
angulo += 0.01;

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-2.0f, -1.3f, -2.8f));
model = glm::scale(model, glm::vec3(0.4f, 0.7f, 0.5f));
//model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(2.0f, -1.3f, -2.8f));
model = glm::scale(model, glm::vec3(0.4f, 0.7f, 0.5f));
//model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

Se crean las pirámides para las hojas de los arboles

```
shaderList[6].useShader();
uniformModel = shaderList[6].getModelLocation();
uniformProjection = shaderList[6].getProjectLocation();
angulo += 0.01;

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-2.0f, -0.3f, -3.0f));
model = glm::scale(model, glm::vec3(1.0f, 1.5f, 1.0f));
//model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-2.0f, -0.3f, -3.0f));
model = glm::scale(model, glm::vec3(1.0f, 1.5f, 1.0f));;
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
//model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();


model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-2.0f, -0.3f, -3.0f));
model = glm::scale(model, glm::vec3(1.0f, 1.5f, 1.0f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
//model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-2.0f, -0.3f, -3.0f));
model = glm::scale(model, glm::vec3(1.0f, 1.5f, 1.0f));
model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
//model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(2.0f, -0.3f, -3.0f));
model = glm::scale(model, glm::vec3(1.0f, 1.5f, 1.0f));
//model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(2.0f, -0.3f, -3.0f));
model = glm::scale(model, glm::vec3(1.0f, 1.5f, 1.0f));;
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
//model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();


model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(2.0f, -0.3f, -3.0f));
model = glm::scale(model, glm::vec3(1.0f, 1.5f, 1.0f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
//model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(2.0f, -0.3f, -3.0f));
model = glm::scale(model, glm::vec3(1.0f, 1.5f, 1.0f));
model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
//model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();
```

RESULTADO:



CONCLUSIONES:

A lo largo de la realización de la práctica, se lograron cumplir los objetivos planteados, ya que se comprendió de manera correcta el funcionamiento de los índices para la creación de figuras y la gestión de arreglos para asignar diferentes colores a cada una de ellas. Además, el trabajo con shaders se entendió de manera óptima debido a la repetitividad del proceso, lo que permitió reforzar su aplicación y uso dentro del desarrollo.

Por otro lado, el error de interpretación en la instrucción resultó ser una ventaja significativa, ya que permitió enfrentar y resolver diversos problemas, en especial aquellos relacionados con las rotaciones. Esto contribuyó a un mejor entendimiento de las transformaciones en OpenGL. Asimismo, la experiencia adquirida en el ejercicio previo a la práctica fue de gran ayuda, ya que permitió aprender el uso de una nueva función para definir los márgenes de las figuras, facilitando así la implementación y estructuración del código.

COMENTARIOS DE LA PRÁCTICA:

Considero que en esta práctica me faltó un poco de detalle en lo solicitado, ya que en ambos ejercicios realicé cosas que no eran requeridas, aunque esto no necesariamente fue algo negativo. A pesar de ello, me gustó mucho la forma en la que estuvo estructurada la práctica, ya que me ayudó a mejorar mi capacidad para dimensionarme en un espacio tridimensional.

El único problema significativo al que me enfrenté fue comprender de manera óptima la rotación, lo cual requirió un esfuerzo adicional. De igual manera, disfruté bastante el contenido y la manera en que estuvo organizada la práctica. Sin embargo, creo que una explicación más detallada de lo solicitado habría sido de gran ayuda para evitar confusiones.