UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

INGENIERÍA EN COMPUTACIÓN

LABORATORIO DE COMPUTACIÓN GRÁFICA e INTERACCIÓN HUMANO COMPUTADORA

EJERCICIOS DE CLASE Nº 02

NOMBRE COMPLETO: Diego Aldair García Hernández

Nº de Cuenta: 319024045

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 04

SEMESTRE 2025-2

FECHA DE ENTREGA LÍMITE: 19 de febrero de 2025

CALIFICACIÓN: _____

Introducción

Este ejercicio tiene como objetivo la generación y manipulación de figuras geométricas en OpenGL, utilizando índices y transformaciones geométricas dentro de un espacio definido por una proyección ortogonal. Se buscó construir una escena con figuras básicas, como triángulos y cuadrados, para representar una casa con su respectivo techo, puertas, ventanas y árboles.

Desarrollo

Se generaron figuras básicas como triángulos y cuadrados para representar una casa con su respectivo techo, puertas, ventanas y árboles. Para ello, se definieron los vértices de las figuras reutilizando estructuras previamente definidas:

- Triángulo Azul: Se utilizó como techo de la casa.

```
// Techo
GLfloat vertices_triangulorojo[] = {
    //X         Y           Z           R       G       B
    -1.0f,  -1.0f,      0.5f,           0.0f,   0.0f,   1.0f,
    1.0f,   -1.0f,      0.5f,           0.0f,   0.0f,   1.0f,
    0.0f,   1.0f,       0.5f,           0.0f,   0.0f,   1.0f,

};

MeshColor* triangulorojo = new MeshColor();
triangulorojo->CreateMeshColor(vertices_triangulorojo, 18);
meshColorList.push_back(triangulorojo);
```

- Triángulo Verde: Representó la copa de los árboles.

```
// Pino
GLfloat vertices_pino[] = {
    //X         Y           Z           R       G       B
    -1.0f,  -1.0f,      0.5f,           0.0f,   0.5f,   0.0f,
    1.0f,   -1.0f,      0.5f,           0.0f,   0.5f,   0.0f,
    0.0f,   1.0f,       0.5f,           0.0f,   0.5f,   0.0f,

};

MeshColor* pino = new MeshColor();
pino->CreateMeshColor(vertices_pino, 18);
meshColorList.push_back(pino);
```

- Cuadrado Rojo: Usado como base de la casa.

```
// Casa (cuadrado grande)
GLfloat vertices_cuadradoverde[] = {
    //X         Y           Z           R       G       B
    -0.5f,  -0.5f,      0.5f,           1.0f,   0.0f,   0.0f,
    0.5f,   -0.5f,      0.5f,           1.0f,   0.0f,   0.0f,
    0.5f,   0.5f,       0.5f,           1.0f,   0.0f,   0.0f,
    -0.5f,  -0.5f,      0.5f,           1.0f,   0.0f,   0.0f,
    0.5f,   0.5f,       0.5f,           1.0f,   0.0f,   0.0f,
    -0.5f,  0.5f,       0.5f,           1.0f,   0.0f,   0.0f,

};
MeshColor* cuadradoverde = new MeshColor();
cuadradoverde->CreateMeshColor(vertices_cuadradoverde, 36);
meshColorList.push_back(cuadradoverde);
```

- Cuadrado Verde: Empleado para las ventanas y la puerta.

```
// Puerta y ventanas
GLfloat vertices_puerta[] = {
    //X         Y           Z           R       G       B
    -0.5f,  -0.5f,      0.5f,           0.0f,   1.0f,   0.0f,
    0.5f,   -0.5f,      0.5f,           0.0f,   1.0f,   0.0f,
    0.5f,   0.5f,       0.5f,           0.0f,   1.0f,   0.0f,
    -0.5f,  -0.5f,      0.5f,           0.0f,   1.0f,   0.0f,
    0.5f,   0.5f,       0.5f,           0.0f,   1.0f,   0.0f,
    -0.5f,  0.5f,       0.5f,           0.0f,   1.0f,   0.0f,

};
MeshColor* puerta = new MeshColor();
puerta->CreateMeshColor(vertices_puerta, 36);
meshColorList.push_back(puerta);
```

- Cuadrado café: Utilizado para los troncos de los árboles.

```
// Troncos
GLfloat vertices_tronco[] = {
    //X         Y           Z           R       G       B
    -0.5f,  -0.5f,      0.5f,           0.478f, 0.255f, 0.0f,
    0.5f,   -0.5f,      0.5f,           0.478f, 0.255f, 0.0f,
    0.5f,   0.5f,       0.5f,           0.478f, 0.255f, 0.0f,
    -0.5f,  -0.5f,      0.5f,           0.478f, 0.255f, 0.0f,
    0.5f,   0.5f,       0.5f,           0.478f, 0.255f, 0.0f,
    -0.5f,  0.5f,       0.5f,           0.478f, 0.255f, 0.0f,

};
MeshColor* tronco = new MeshColor();
tronco->CreateMeshColor(vertices_tronco, 36);
meshColorList.push_back(tronco);
```

Cada figura se almacenó en un MeshColor y se agregó a la lista meshColorList, lo que permitió su posterior instancia y transformación en la escena. Se empleó la proyección ortogonal definida por glm::ortho(-1.0f, 1.0f, -1.0f, 1.0f, 0.1f, 100.0f), lo que facilitó el trabajo en un entorno 2D sin distorsiones de perspectiva.

A continuación, se presenta un pequeño fragment de código que se utilizó para dibujar la casa con sus respectivas transformaciones geométricas de escalamiento y traslación.

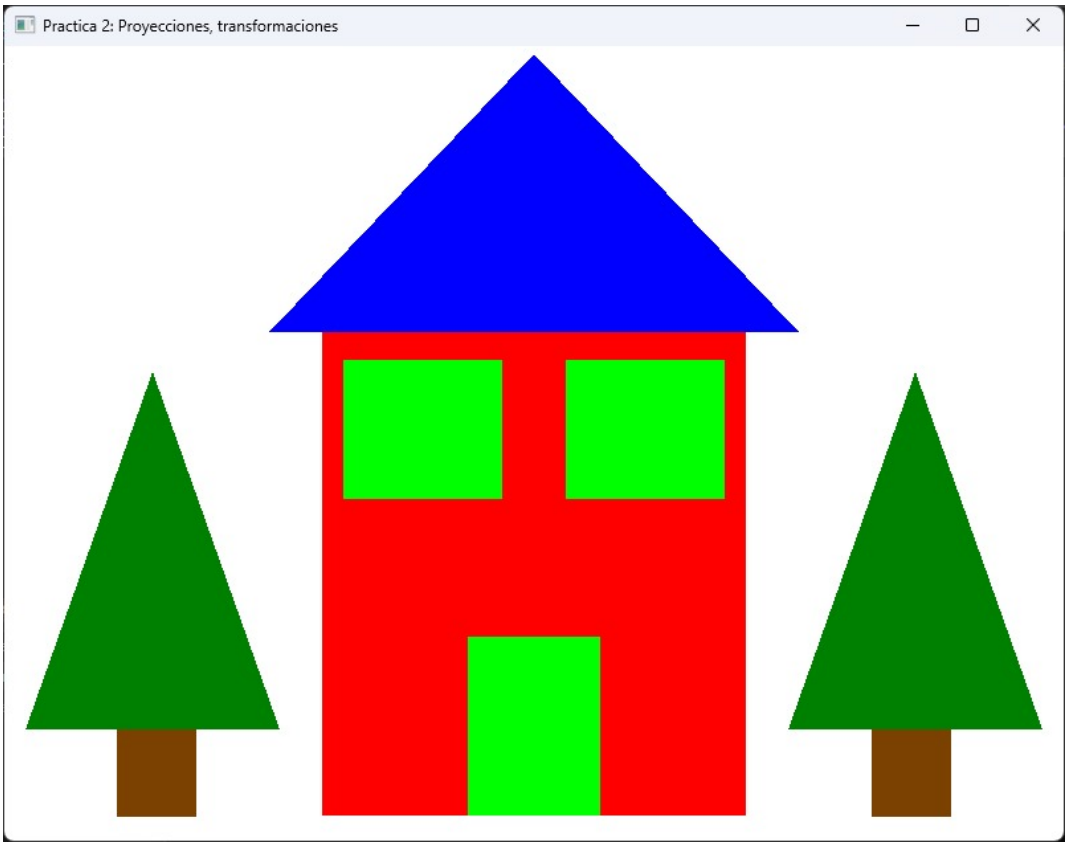
```
// Dibujar tronco2
model = glm::mat4(1.0);
model = glm::scale(model, glm::vec3(0.15f, 0.25f, 1.0f));
model = glm::translate(model, glm::vec3(4.75f, -3.26f, -1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[3]->RenderMeshColor();

// Dibujar puerta
model = glm::mat4(1.0);
model = glm::scale(model, glm::vec3(0.25f, 0.45f, 1.0f));
model = glm::translate(model, glm::vec3(0.0f, -1.58f, -1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[2]->RenderMeshColor();

// Dibujar ventana1
model = glm::mat4(1.0);
model = glm::scale(model, glm::vec3(0.3f, 0.35f, 1.0f));
model = glm::translate(model, glm::vec3(-0.7f, 0.1f, -1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[2]->RenderMeshColor();
```

Problemas presentados.

La puerta no se dibujaba correctamente, ya que se estaba renderizando después de la casa, lo que causaba que quedara oculta detrás de ella. La solución fue cambiar el orden de renderizado, dibujando la puerta antes que la casa.

Por otro lado, algunos objetos aparecían en ubicaciones inesperadas porque glm::scale() se aplicaba antes de glm::translate(), lo que afectaba el desplazamiento. Para corregirlo, se invirtió el orden de las transformaciones, aplicando primero glm::translate() y luego glm::scale().

Salida



Conclusión

Este ejercicio permitió comprender cómo manejar proyecciones ortogonales y transformaciones geométricas en OpenGL para construir una escena 2D compleja a partir de figuras básicas. Se trabajó con la creación de mallas, su almacenamiento en estructuras dinámicas y la aplicación de traslaciones y escalados para posicionarlas correctamente. A lo largo del desarrollo, se presentaron desafíos como el orden de renderizado, errores de acceso a vectores y la correcta aplicación de transformaciones, los cuales se resolvieron mediante depuración y ajustes en el código. El ejercicio fortaleció los conocimientos en el uso de matrices de transformación y la representación de escenas en OpenGL.