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 3

NOMBRE COMPLETO: Gomez Enríquez Agustin

Nº de Cuenta: 317031405

GRUPO DE LABORATORIO: 3

GRUPO DE TEORÍA: 5

SEMESTRE 2026-1

FECHA DE ENTREGA LÍMITE: 02 – septiembre - 2025

CALIFICACIÓN: _____

EJERCICIOS DE SESIÓN:

1. **Actividades realizadas. Una descripción de los ejercicios y capturas de pantalla de bloques de código generados y de ejecución del programa**

```
// -------------------- ESCENA CASA -------------------- ESCENA CASA -------------------- ESCENA CAS
int main()
{
    mainWindow = Window(800, 600);
    mainWindow.Initialise();

    CrearCubo();                  // idx 0
    CrearPiramideTriangular();    // idx 1 (no se usa, pero lo conservo)
    CrearCilindro(24, 0.5f);      // idx 2
    CrearCono(24, 1.0f);          // idx 3
    CrearPiramideCuadrangular();  // idx 4 (esta si se usa) XD
    CreateShaders();

    // Ajustes de camara
    camera = Camera(
        glm::vec3(0.0f, 4.0f, 10.0f),   // posición
        glm::vec3(0.0f, 1.0f, 0.0f),
        -60.0f, 0.0f,
        2.0f, 2.0f                      // moveSpeed, turnSpeed
    );


    glm::mat4 projection = glm::perspective(glm::radians(45.0f),
        (float)mainWindow.getBufferWidth() / (float)mainWindow.getBufferHeight(), 0.1f, 100.0f);

```

Dentro del main se trabajó con los índices a los cuales modifique el cilindro y cono para que se ajustaran a la creación de los árboles. Además, a esto hice correcciones a la cámara y la perspectiva puesto que la posición inicial era en el interior de la casa y solo podía ver colores sólidos. así que se coloco una vista inicial alejada en los tres vértices para tener una vista superior y frontal de la casa. así mismo arregle la velocidad de la cámara puesto que al interactuar con las teclas se perdía el origen del mapa.

Shader.cpp | shader.frag | **shader.vert** | practica3.cpp

```
#version 430 core
layout (location = 0) in vec3 pos;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

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

```

Shader.cpp | **shader.frag** | shader.vert | practica3.cpp

```
#version 430 core
out vec4 FragColor;
uniform vec3 Color;

void main() {
    FragColor = vec4(Color, 1.0);
}

```

También modificamos las versiones de los shaders para poder correr correctamente el proyecto y ajustamos la proyección de la cámara para poder movernos libremente en la ejecución.

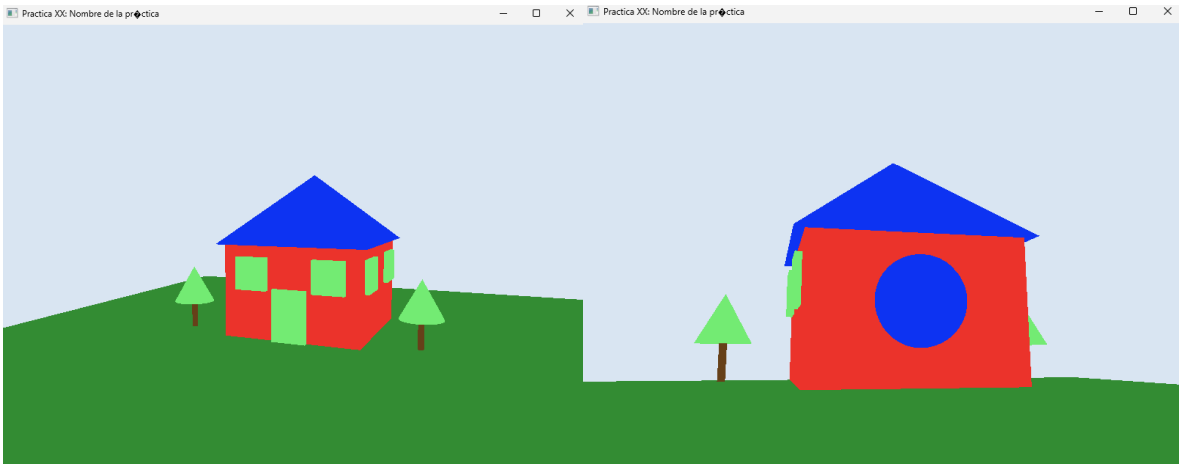
```
practica3                                                    (Ámbito global)
204     while (!mainWindow.getShouldClose())
205     {
206         // Timing + eventos
207         GLfloat now = glfwGetTime();
208         deltaTime = now - lastTime; deltaTime += (now - lastTime) / limitFPS; lastTime = now;
209         glfwPollEvents();
210         camera.keyControl(mainWindow.getsKeys(), deltaTime);
211         camera.mouseControl(mainWindow.getXChange(), mainWindow.getYChange());
212
213         // Clear
214         glClearColor(0.85f, 0.90f, 0.95f, 1.0f);
215         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
216
217         // Shader
218         shaderList[0].useShader();
219         uModel = shaderList[0].getModelLocation();
220         uProj = shaderList[0].getProjectLocation();
221         uView = shaderList[0].getViewLocation();
222         //uColor = shaderList[0].getColorLocation();
223
224         // Proyección y vista (una vez por frame)
225         glUniformMatrix4fv(uProj, 1, GL_FALSE, glm::value_ptr(projection));
226         glm::mat4 view = camera.calculateViewMatrix();
227         glUniformMatrix4fv(uView, 1, GL_FALSE, glm::value_ptr(view));
228
229         glm::mat4 model(1.0f);
230
231         // PISO
232         model = glm::mat4(1.0f);
233         model = glm::translate(model, glm::vec3(0.0f, -0.51f, -4.0f));
234         model = glm::scale(model, glm::vec3(30.0f, 0.02f, 30.0f));
235         glUniformMatrix4fv(uModel, 1, GL_FALSE, glm::value_ptr(model));
236         glUniform3fv(uColor, 1, glm::value_ptr(COLOR_GRASS));
237         meshList[0]->RenderMesh();
238
239         // CUERPO DE LA CASA
240         model = glm::mat4(1.0f);
241         model = glm::translate(model, glm::vec3(0.0f, houseH / 2.0f, -4.0f));
242         model = glm::scale(model, glm::vec3(houseW, houseH, houseD));
243         glUniformMatrix4fv(uModel, 1, GL_FALSE, glm::value_ptr(model));
244         glUniform3fv(uColor, 1, glm::value_ptr(COLOR_RED));
245         meshList[0]->RenderMesh();
```

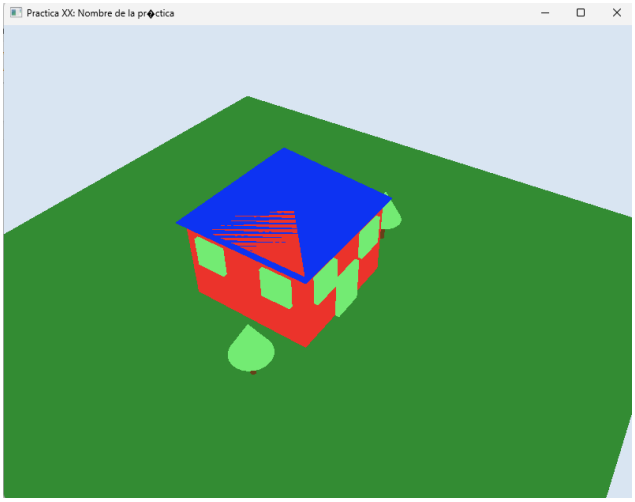
```
// TECHO
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, houseH + roofH / 2.0f, -4.0f));
model = glm::scale(model, glm::vec3(houseW * 1.1f, roofH, houseD * 1.1f));
glUniformMatrix4fv(uModel, 1, GL_FALSE, glm::value_ptr(model));
glUniform3fv(uColor, 1, glm::value_ptr(COLOR_BLUE));
meshList[4]->RenderMeshGeometry();

// PUERTA
float doorW = 1.5f, doorH = 2.2f, doorT = 0.2f;
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, doorH / 2.0f, -4.0f + houseD / 2.0f + doorT / 2.0f + 0.01f));
model = glm::scale(model, glm::vec3(doorW, doorH, doorT));
glUniformMatrix4fv(uModel, 1, GL_FALSE, glm::value_ptr(model));
glUniform3fv(uColor, 1, glm::value_ptr(COLOR_GREEN));
meshList[0]->RenderMesh();

// VENTANAS
float winW = 1.4f, winH = 1.4f, winT = 0.2f, winY = 2.8f, winX = 1.7f;
for (int s : {-1, +1}) {
    model = glm::mat4(1.0f);
    model = glm::translate(model, glm::vec3(s * winX, winY, -4.0f + houseD / 2.0f + winT / 2.0f + 0.01f));
    model = glm::scale(model, glm::vec3(winW, winH, winT));
    glUniformMatrix4fv(uModel, 1, GL_FALSE, glm::value_ptr(model));
    glUniform3fv(uColor, 1, glm::value_ptr(COLOR_GREEN));
    meshList[0]->RenderMesh();
}

// VENTANAS
float sideZ = 1.7f;
// Lado izquierdo
for (int sZ : {-1, +1}) {
    model = glm::mat4(1.0f);
    model = glm::translate(model, glm::vec3(-houseW / 2.0f - 0.01f - winT / 2.0f, winY, -4.0f + sZ * sideZ));
    model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0, 1, 0)); // pegar al muro lateral
    model = glm::scale(model, glm::vec3(winW, winH, winT));
    glUniformMatrix4fv(uModel, 1, GL_FALSE, glm::value_ptr(model));
    glUniform3fv(uColor, 1, glm::value_ptr(COLOR_GREEN));
    meshList[0]->RenderMesh();
}
// Lado derecho
```

```cpp
// Lado derecho
for (int sZ : {-1, +1}) {
    model = glm::mat4(1.0f);
    model = glm::translate(model, glm::vec3(+houseW / 2.0f + 0.01f + winT / 2.0f, winY, -4.0f + sZ * sideZ));
    model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0, 1, 0));
    model = glm::scale(model, glm::vec3(winW, winH, winT));
    glUniformMatrix4fv(uModel, 1, GL_FALSE, glm::value_ptr(model));
    glUniform3fv(uColor, 1, glm::value_ptr(COLOR_GREEN));
    meshList[0]->RenderMesh();
}

// VENTANA CIRCULAR TRASERA
float circR = 1.2f;
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, 2.2f, -4.0f - houseD / 2.0f - 0.02f));
model = glm::scale(model, glm::vec3(circR, circR, circR));
glUniformMatrix4fv(uModel, 1, GL_FALSE, glm::value_ptr(model));
glUniform3fv(uColor, 1, glm::value_ptr(COLOR_BLUE));
sp.render(); // usa el mismo shader activo

// ÁRBOLES
struct Tree { glm::vec3 p; };
std::vector<Tree> trees = {
    { glm::vec3(-houseW / 2.0f - 2.2f, 0.0f, -4.0f + 2.0f) },
    { glm::vec3(+houseW / 2.0f + 2.2f, 0.0f, -4.0f + 2.0f) }
};
for (auto& t : trees) {
    // Tronco
    float trunkH = 1.2f, trunkR = 0.25f;
    model = glm::mat4(1.0f);
    model = glm::translate(model, glm::vec3(t.p.x, trunkH / 2.0f, t.p.z));
    model = glm::scale(model, glm::vec3(trunkR, trunkH, trunkR));
    glUniformMatrix4fv(uModel, 1, GL_FALSE, glm::value_ptr(model));
    glUniform3fv(uColor, 1, glm::value_ptr(COLOR_BROWN));
    meshList[2]->RenderMeshGeometry();

    // Copa
    float coneH = 1.6f, coneR = 0.9f;
    model = glm::mat4(1.0f);
    model = glm::translate(model, glm::vec3(t.p.x, trunkH + coneH / 2.0f, t.p.z));
    model = glm::scale(model, glm::vec3(coneR, coneH, coneR));
    glUniformMatrix4fv(uModel, 1, GL_FALSE, glm::value_ptr(model));
    glUniform3fv(uColor, 1, glm::value_ptr(COLOR_GREEN));
    meshList[3]->RenderMeshGeometry();
```

Trabajando con las primitivas establecemos las ubicaciones, tamaño, color y forma de cada parte de la casa. Y repetimos el proceso para cada una de las figuras.

Ejecución:





2. **Problemas presentados. Listar si surgieron problemas a la hora de ejecutar el código**

- Tuve problemas en la computadora del laboratorio pues no me dejaba compilar el programa a pesar de seguir el mismo protocolo para cada proyecto. Al realizar el ejercicio en cas no tuve ese problema, pero lo asocio a la versión de los shaders con los que se trabajó.
- Un problema que note al finalizar la practica y no se resolver es el que algunos colores no son del todo sólidos, pues se enciman algunas figuras como el caso de la estructura de la casa con el techo, me gustaría conocer como se optimiza esta cuestión.

3. **Conclusión:**

Esta practica me ayudo mucho a como interpretar el pipeline gráfico, de manera que se trabaja por bloques de código. Es decir que se establecen parámetros con los que se van a trabajar, establecemos nuestras figuras primitivas y jugamos con los vértices de cada una de estas primitivas para acoplarlas a la imagen que buscamos. Además, a esto fue mi primer acercamiento con la interacción de un entorno en 3d y poder moverme libremente con el uso de teclado. Para concluir esta práctica fue difícil de entender pues sigo atrasado en conocimiento, pero que el código venga con comentarios explicando el uso de cada función me ayudo mucho a saber interpretar cada función.