
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 2

**NOMBRE COMPLETO:** Gomez Enríquez Agustin

**Nº de Cuenta:** 317031405

**GRUPO DE LABORATORIO:** 3

**GRUPO DE TEORÍA: 5**

**SEMESTRE 2026-1**

**FECHA DE ENTREGA LÍMITE: 26 – agosto - 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**

```
62  int main() {
63      mainWindow = Window(800, 600);
64      mainWindow.Initialise();
65
66      // 2D: sin profundidad
67      glDisable(GL_DEPTH_TEST);
68
69      CreaPiramide();       // 0: triángulo
70      CrearCubo();          // 1: cuadrado
71      CreateShaders();
```

Vamos a comenzar modificando el main, para asegurarnos que todas nuestras figuras se van a trabajar en los planos Xy Z, es decir que no tendrán fondo y solo se visualizara una imagen en 2d

```
73      // Proyección ortográfica dependiente del aspecto
74      auto orthoProjection = [&]() -> glm::mat4 {
75          int w = mainWindow.getBufferWidth();
76          int h = mainWindow.getBufferHeight();
77          float aspect = (h > 0) ? (float)w / (float)h : 1.3333f;
78          return glm::ortho(-aspect, aspect, -1.0f, 1.0f, -1.0f, 1.0f);
79          };
80
```

Para asegurarnos de hacer una proyección ortogonal y hacer que nada se deforme y para poder encajar la casa y los árboles dentro de la ventana.

```
98          const glm::vec3 COLOR_ROOF_BLUE = { 0.00f, 0.25f, 1.00f };   // triángulo azul
99          const glm::vec3 COLOR_TRI_GREEN = { 0.00f, 0.50f, 0.00f };   // triángulo verde (0,0.5,0)
100         const glm::vec3 COLOR_BODY_RED = { 0.93f, 0.18f, 0.18f };   // cuadrado rojo
101         const glm::vec3 COLOR_SQUARE_GREEN = { 0.61f, 0.98f, 0.28f };   // cuadrado verde
102         const glm::vec3 COLOR_BROWN = { 0.478f, 0.255f, 0.067f };// cuadrado café
```

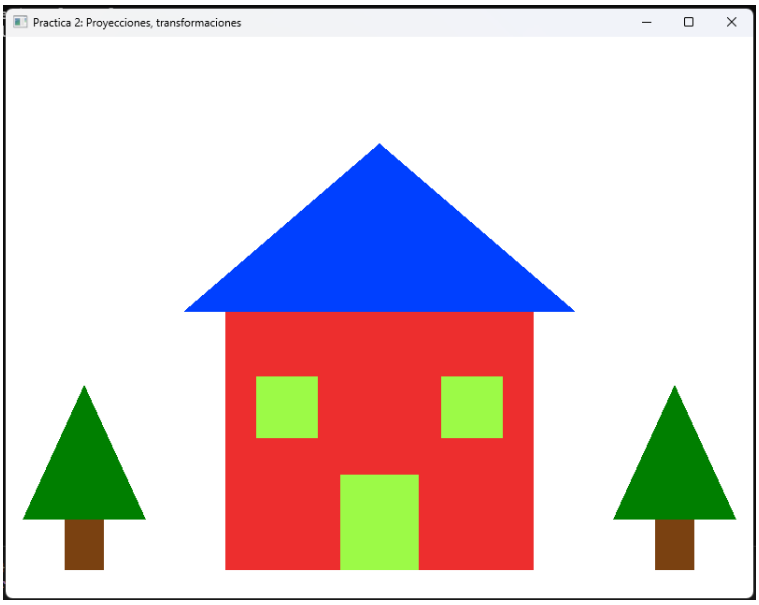
Ya dentro de nuestro loop, definimos los colores que nos dio el profesor para cada una de las figuras.

```
// ===== Casa =====
// Muro rojo (centrado en X). Altura y base alineada a GROUND_Y
const float BODY_W = 1.10f;
const float BODY_H = 0.92f;
const float bodyHalfH = BODY_H * 0.5f;
const float bodyCY = GROUND_Y + bodyHalfH;   // base del muro = GROUND_Y

{
    glm::mat4 M(1.0f);
    M = glm::translate(M, glm::vec3(0.0f, bodyCY, 0.0f));
    M = glm::scale(M, glm::vec3(BODY_W, BODY_H, 1.0f));
    glUniformMatrix4fv(uModel, 1, GL_FALSE, glm::value_ptr(M));
    glUniform3fv(uCol, 1, &COLOR_BODY_RED[0]);
    meshList[1]->RenderMesh();
}

// Techo azul (apoya justo sobre el muro)
const float ROOF_W = 1.40f;
const float ROOF_H = 0.60f;
const float roofCY = (bodyCY + bodyHalfH) + ROOF_H * 0.5f; // base del triángulo = top del muro
{
    glm::mat4 M(1.0f);
    M = glm::translate(M, glm::vec3(0.0f, roofCY, 0.0f));
    M = glm::scale(M, glm::vec3(ROOF_W, ROOF_H, 1.0f));
    glUniformMatrix4fv(uModel, 1, GL_FALSE, glm::value_ptr(M));
    glUniform3fv(uCol, 1, &COLOR_ROOF_BLUE[0]);
    meshList[0]->RenderMesh();
}

// Puerta (cuadrado verde) – dentro del muro, base al ras del muro (no afuera)
const float DOOR_W = 0.28f, DOOR_H = 0.34f;
{
    glm::mat4 M(1.0f);
    const float doorCY = GROUND_Y + DOOR_H * 0.5f; // base = GROUND_Y (igual que muro)
    M = glm::translate(M, glm::vec3(0.0f, doorCY, 0.0f));
    M = glm::scale(M, glm::vec3(DOOR_W, DOOR_H, 1.0f));
    glUniformMatrix4fv(uModel, 1, GL_FALSE, glm::value_ptr(M));
    glUniform3fv(uCol, 1, &COLOR_SQUARE_GREEN[0]);
    meshList[1]->RenderMesh();
}
```

Como ya teníamos la creación de figuras antes del main, ahora vamos a definir su tamaño y posición para asegurarnos que se visualiza como en la imagen de referencia. Recordando que cada una se debe renderizar.

```
// Ventanas (cuadrados verdes) – dentro del muro
const float WIN_W = 0.22f, WIN_H = 0.22f;
auto drawWindow = [&](float cx) {
    glm::mat4 W(1.0f);
    // un poco por debajo de la mitad superior del muro
    float wy = bodyCY + 0.12f;
    W = glm::translate(W, glm::vec3(cx, wy, 0.0f));
    W = glm::scale(W, glm::vec3(WIN_W, WIN_H, 1.0f));
    glUniformMatrix4fv(uModel, 1, GL_FALSE, glm::value_ptr(W));
    glUniform3fv(uCol, 1, &COLOR_SQUARE_GREEN[0]);
    meshList[1]->RenderMesh();
    };
drawWindow(-0.33f);
drawWindow(0.33f);

// ===== Árboles (tronco + copa centrada al tronco) =====
auto drawTree = [&](float side) {
    int w = mainWindow.getBufferWidth();
    int h = mainWindow.getBufferHeight();
    float aspect = (h > 0) ? (float)w / (float)h : 1.3333f;

    // Medias dimensiones
    const float trunkHalfX = 0.07f;
    const float trunkHalfY = 0.09f;
    const float canopyHalfX = 0.22f;   // triángulo verde más ancho que el tronco
    const float canopyHalfY = 0.24f;
    const float marginX = 0.06f;

    // CENTRO X COMÚN (misma X para tronco y copa, copa centrada sobre tronco)
    const float halfXmax = (canopyHalfX > trunkHalfX) ? canopyHalfX : trunkHalfX;
    const float centerX = side * (aspect - marginX - halfXmax);

    // Tronco (base al mismo GROUND_Y que la casa)
    {
        glm::mat4 T(1.0f);
        const float trunkCY = GROUND_Y + trunkHalfY;
        T = glm::translate(T, glm::vec3(centerX, trunkCY, 0.0f));
        T = glm::scale(T, glm::vec3(trunkHalfX * 2.0f, trunkHalfY * 2.0f, 1.0f));
        glUniformMatrix4fv(uModel, 1, GL_FALSE, glm::value_ptr(T));
        glUniform3fv(uCol, 1, &COLOR_BROWN[0]);
        meshList[1]->RenderMesh();
    }
```

## Ejecución




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

   Realmente no tuve ningún problema para crear la casa, ya que fue estar ajustando tamaños y posiciones de las figuras. Y la estructura del código se maneja por bloques, así que es relativamente rápida la modificación de estas.

## Conclusión

El día de hoy me ayudo mucho conocer la geometría que se trabaja en OpenGL, ya que podría considerar fundamental ese bloque de código:

```
glm::mat4 M(1.0f);

M = glm::translate(M, glm::vec3(XXXX, XXXX, XXXX));

M = glm::scale(M, glm::vec3(XXXX, XXXX, XXXX));

glUniformMatrix4fv(uModel, 1, GL_FALSE, glm::value_ptr(M));

glUniform3fv(uCol, 1, &COLOR_BODY_RED[0]);

meshList[1]->RenderMesh();
```

Igualmente conocer los fundamentos de trabajar proyecciones ortogonales nos permite definir parámetros de como trabajar con nuestros triángulos para proyectos mas complejos.