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: Uriarte Ortiz Enrique Yahir

Nº de Cuenta: 318234757

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 04

SEMESTRE 2025-1

FECHA DE ENTREGA LÍMITE: Martes 26 de Agosto del 2024

CALIFICACIÓN: _____

EJERCICIOS DE SESIÓN:

1. *Actividades realizadas.*

   1) Instanciar primitivas geométricas para recrear el dibujo de la práctica pasada en 3D, se requiere que exista un piso; la casa tiene una ventana azul circular justo en medio de la pared trasera, 2 ventanas verdes en cada pared lateral iguales a las de la pared frontal y solo puerta en la pared frontal.

   Para comenzar con la actividad, agregue los archivos shader de los colores creados en la práctica anterior, además de modificar cada archivo *".vert"* para poder utilizar el movimiento de la cámara alrededor de los modelos.

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

Fig. 1. *Código de ".vert" para color de figuras, para mover los objetos.*

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

Fig. 2. *Código de ".vert" para color de figuras, para mover la cámara.*

Después los definí en "E03- 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.

```
//Vertex Shader
static const char* vShader = "shaders/shader.vert";
static const char* fShader = "shaders/shader.frag";
static const char* vShaderColor = "shaders/shadercolor.vert";

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_verde03 = "shaders/shader_verde03.vert";
static const char* fshader_verde03 = "shaders/shader_verde03.frag";
static const char* vshader_cafe = "shaders/shader_cafe.vert";
static const char* fshader_cafe = "shaders/shader_cafe.frag";
```

Fig. 3. *Shaders para los colores en él ".cpp".*

Como en esta actividad realizaremos básicamente la misma proyección de la casa en 3D, pero algunos detalles extras, emplee todas las figuras creadas en la práctica anterior y agregue las nuevas que se solicitan.

```
//Codigo de la practica anterior
void Casa() { ... }
void Techo() { ... }
void Puerta() { ... }
void Ventana01() { ... }
void Ventana02() { ... }
void Arbol01() { ... }
void Arbol02() { ... }
void Tronco01() { ... }
void Tronco02() { ... }

//Codigo nuevo para los elementos agregados
void Ventana_Lateral01() { ... }
void Ventana_Lateral02() { ... }
void Ventana_Lateral03() { ... }
void Ventana_Lateral04() { ... }
void CrearCilindro(int res, float R, float altura, float posX, float posY, float posZ) { ... }
void Terreno() { ... }
```

Fig. 4. *Funciones para las figuras de la casa.*

Entre las nuevas figuras que necesitaba definir primero hice las nuevas ventanas, estas fueron un tanto fáciles de hacer, simplemente analice la distancia a la que debían estar en las otras 2 caras, recorriéndose principalmente en los ejes X y Z, una vez realizadas las del lado derecho para hacer las del izquierdo solo cambie los signos en el eje X.

```
void Ventana_Lateral01() {
    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
        0.55f, 0.1f, 0.4f, 0.25f, 0.1f, 0.4f,
        0.25f, 0.4f, 0.4f, 0.55f, 0.4f, 0.4f,
        // Uniones traseras
        0.55f, 0.1f, 0.1f, 0.25f, 0.1f, 0.1f,
        0.25f, 0.4f, 0.1f, 0.55f, 0.4f, 0.1f };
    Mesh* ventana_l01 = new Mesh();
    ventana_l01->CreateMesh(cubo_vertices, cubo_indices, 24, 36);
    meshList.push_back(ventana_l01);}
```

```
void Ventana_Lateral02() {
    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
        0.55f, 0.1f, -0.4f, 0.25f, 0.1f, -0.4f,
        0.25f, 0.4f, -0.4f, 0.55f, 0.4f, -0.4f,
        // Uniones traseras
        0.55f, 0.1f, -0.1f, 0.25f, 0.1f, -0.1f,
        0.25f, 0.4f, -0.1f, 0.55f, 0.4f, -0.1f };
    Mesh* ventana_l02 = new Mesh();
    ventana_l02->CreateMesh(cubo_vertices, cubo_indices, 24, 36);
    meshList.push_back(ventana_l02);}
```

Fig. 5 y Fig. 6. *Funciones para las ventanas laterales, derecha.*

```
void Ventana_Lateral03() {
    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
        -0.55f, 0.1f, 0.4f, -0.25f, 0.1f, 0.4f,
        -0.25f, 0.4f, 0.4f, -0.55f, 0.4f, 0.4f,
        // Uniones traseras
        -0.55f, 0.1f, 0.1f, -0.25f, 0.1f, 0.1f,
        -0.25f, 0.4f, 0.1f, -0.55f, 0.4f, 0.1f };
    Mesh* ventana_l03 = new Mesh();
    ventana_l03->CreateMesh(cubo_vertices, cubo_indices, 24, 36);
    meshList.push_back(ventana_l03);}
```

```
void Ventana_Lateral04() {
    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
        -0.55f, 0.1f, -0.4f, -0.25f, 0.1f, -0.4f,
        -0.25f, 0.4f, -0.4f, -0.55f, 0.4f, -0.4f,
        // Uniones traseras
        -0.55f, 0.1f, -0.1f, -0.25f, 0.1f, -0.1f,
        -0.25f, 0.4f, -0.1f, -0.55f, 0.4f, -0.1f };
    Mesh* ventana_l04 = new Mesh();
    ventana_l04->CreateMesh(cubo_vertices, cubo_indices, 24, 36);
    meshList.push_back(ventana_l04);}
```

Fig. 7 y Fig. 8. *Funciones para las ventanas laterales, izquierda.*

Lo siguiente fue la ventana trasera, para esta tuve que definir en la función *“CrearCilindro”* nuevos parámetros para definir desde un inicio la altura y la posición en X, Y y Z, lo siguiente fue hacer que la impresión del cilindro no fuera paralela al eje Y, sino al eje Z. Ahora Z será igual desde un inicio a “-altura/2.0f” para que el cilindro este centrado en su posición Z, por lo mismo en el calculo de los valores de Z ahora son de Y, y cada vez que calculamos estos valores se le suma “posX” o “posY” para centrar la forma circular en el plano XY mientras se extiende a lo largo del eje Z. Además, como ahora el eje z es donde se pondrá la altura, para lo que corresponde a la cara superior se le agrega la altura en el eje Z.

```
void CrearCilindro(int res, float R, float altura, float posX, float posY, float posZ) {
    int n, i;
    GLfloat dt = 2 * PI / res, x, y, z = -altura / 2.0f + posZ;
    vector<GLfloat> vertices;
    vector<unsigned int> indices;

    for (n = 0; n <= res; n++) {
        if (n != res) { x = R * cos(n * dt) + posX;
                        y = R * sin(n * dt) + posY;}
        else{ x = R * cos(0) + posX;
              y = R * sin(0) + posY;}
        for (i = 0; i < 6; i++) {
            switch (i) {
            case 0: vertices.push_back(x); break;
            case 1: vertices.push_back(y); break;
            case 2: vertices.push_back(z); break;
            case 3: vertices.push_back(x); break;
            case 4: vertices.push_back(y); break;
            case 5: vertices.push_back(z + altura); break;}
        }
    }
```

Fig. 9. *Función para la ventana trasera.*

```
for (n = 0; n <= res; n++) {
    x = R * cos(n * dt) + posX;
    y = R * sin(n * dt) + posY;
    vertices.push_back(x);
    vertices.push_back(y);
    vertices.push_back(z);}

for (n = 0; n <= res; n++) {
    x = R * cos(n * dt) + posX;
    y = R * sin(n * dt) + posY;
    vertices.push_back(x);
    vertices.push_back(y);
    vertices.push_back(z + altura);}

for (i = 0; i < vertices.size(); i++) indices.push_back(i);

Mesh* cilindro = new Mesh();
cilindro->CreateMeshGeometry(vertices, indices, vertices.size(), indices.size());
meshList.push_back(cilindro);}
```

Fig. 10. *Función para la ventana trasera.*

Por último definí un primas rectangular para el piso, la modificación fue a partir de la función del cubo, solo que lo alargue hasta los 2.0f en el eje X, y le di una altura en el eje Y mínima de 0.05f.

```
void Terreno() {
    unsigned int terreno_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 terreno_vertices[] = {
        // Uniones frontales
            -2.0f, -0.5f,  2.0f,  2.0f, -0.5f, 2.0f,
             2.0f, -0.55f,  2.0f, -2.0f, -0.55f, 2.0f,
        // Uniones traseras
            -2.0f, -0.5f, -2.0f,  2.0f, -0.5f, -2.0f,
             2.0f, -0.55f, -2.0f, -2.0f, -0.55f, -2.0f };
    Mesh* terreno = new Mesh();
    terreno->CreateMesh(terreno_vertices, terreno_indices, 24, 36);
    meshList.push_back(terreno);}
```

Fig. 11. *Función para el piso.*

Después agregue a la función "CreateShaders" la asignación de shaders para cada color como en la práctica anterior.
En la función principal main agregue las referencias a las llamadas de las demás figuras en un inicio.

```
int main(){
    mainWindow = Window(800, 600);
    mainWindow.Initialise();

    Casa();
    Puerta();
    Ventana01();
    Ventana02();
    Ventana_Lateral01();
    Ventana_Lateral02();
    Ventana_Lateral03();
    Ventana_Lateral04();
    Techo();
    Arbol01();
    Arbol02();
    Tronco01();
    Tronco02();
    CrearCilindro(70, 0.2f,0.1f,0.0f,0.0f,-0.5f);
    Terreno();
```

Fig. 12. *Definición de funciones de figuras en main.*

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.

```
//Cubo Rojo.
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[0]->RenderMesh();
```

Fig. 13. *Selección de color para el cubo rojo.*

```
//Cubos de puerta y ventanas.
shaderList[2].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);// Puerta
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[1]->RenderMesh();
model = glm::mat4(1.0);// Ventana 01
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[2]->RenderMesh();
model = glm::mat4(1.0);// Ventana 02
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[3]->RenderMesh();
```

Fig. 15. *Selección de color para cubos verdes.*

```
model = glm::mat4(1.0);// Ventana 01L
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[4]->RenderMesh();
model = glm::mat4(1.0);// Ventana 02L
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[5]->RenderMesh();
model = glm::mat4(1.0);// Ventana 03L
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[6]->RenderMesh();
model = glm::mat4(1.0);// Ventana 04L
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[7]->RenderMesh();
```

Fig. 16. *Selección de color para cubos verdes, ventanas nuevas.*

```
//Techo.
shaderList[1].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[8]->RenderMesh();
```

Fig. 17. *Selección de color pirámide azul.*

```
//Arboles.
shaderList[3].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);//Arbol 01
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[9]->RenderMesh();
model = glm::mat4(1.0);//Arbol 02
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[10]->RenderMesh();
```

Fig. 18. *Selección de color para hojas de árboles.*

```
//Troncos.
shaderList[4].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);//Tronco 01
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[11]->RenderMesh();
model = glm::mat4(1.0);//Tronco 02
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[12]->RenderMesh();
```

Fig. 19. *Selección de color para troncos de árboles.*

*Para el caso dela ventana trasera, en el “**meshList[]**”* se cambió a *“renderMesh()”* para poder utilizar el cilindro.

```
//Ventana trasera
shaderList[1].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[13]->RenderMeshGeometry();
```

Fig. 20. *Selección de color para ventana trasera.*

```
//Terreno
shaderList[5].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[14]->RenderMesh();
```

Fig. 21. *Selección de color para piso de la casa y árboles.*

Por ultimo al ejecutar probe en los dos, cuando movemos el modelo y la cámara se queda en un lugar, y cuando la cámara es la que se mueve.

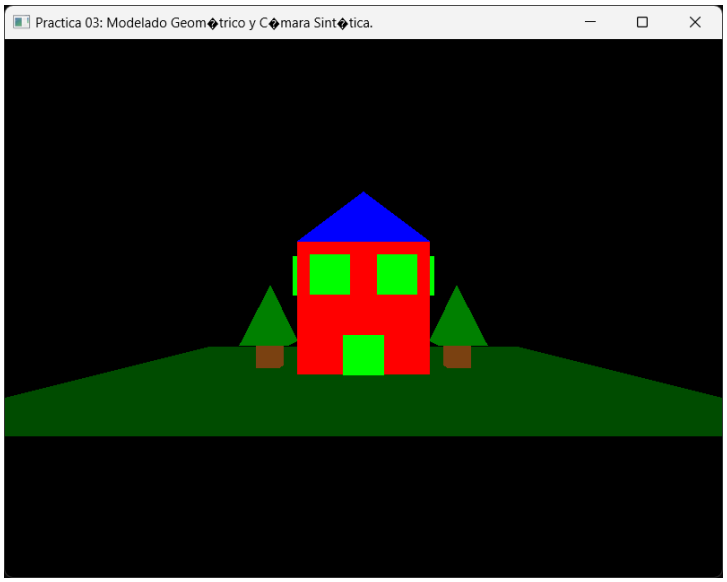

Fig. 22. *Ejecución de programa con cámara estática.*

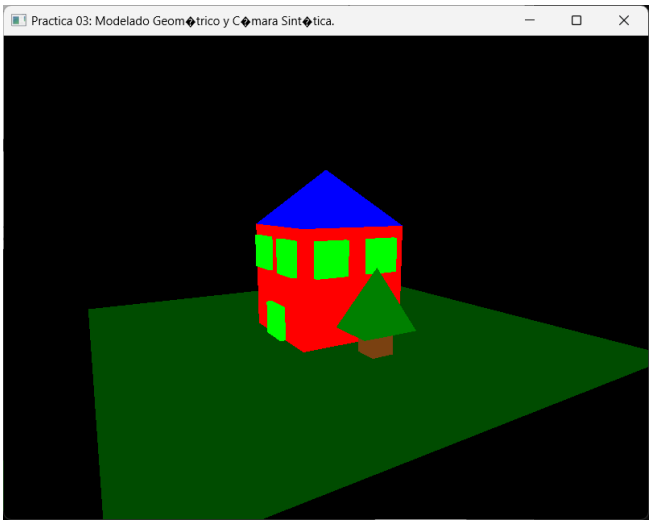

Fig. 23. *Ejecución de programa con cámara estática.*

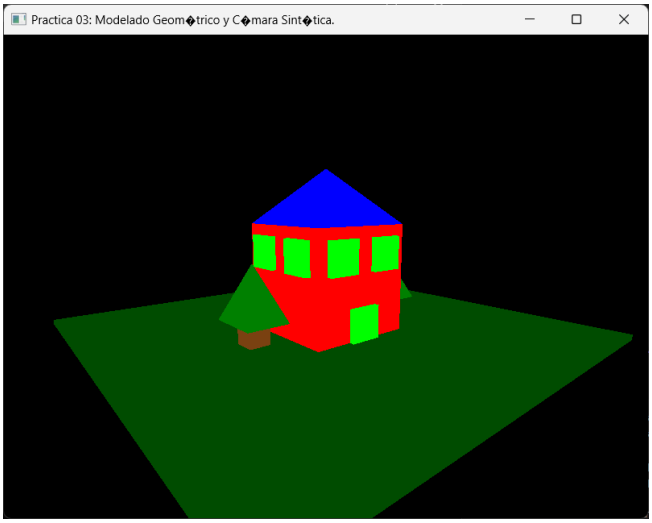

Fig. 24. *Ejecución de programa con cámara estática.*

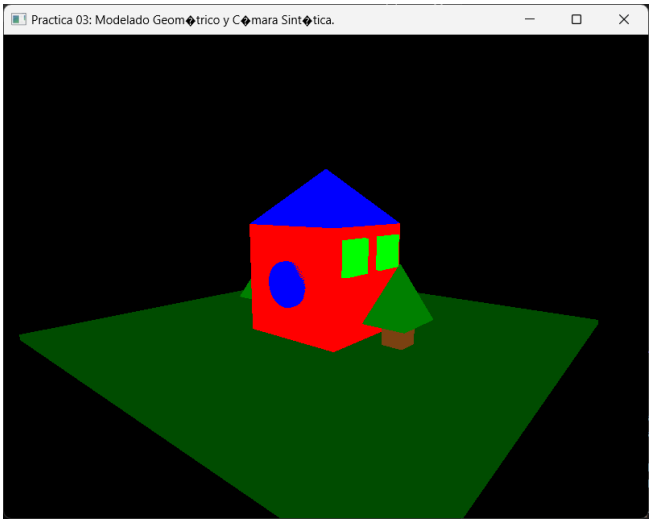

Fig. 25. *Ejecución de programa con cámara estática.*

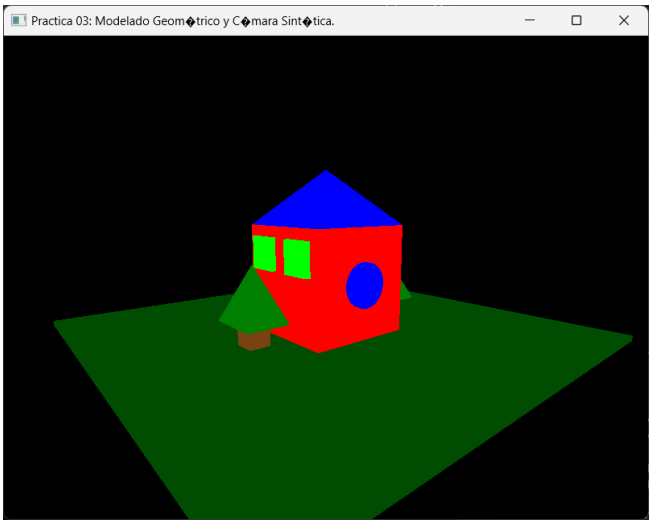

Fig. 26. *Ejecución de programa con cámara estática.*

Fig. 27. *Ejecución de programa con cámara estática.*

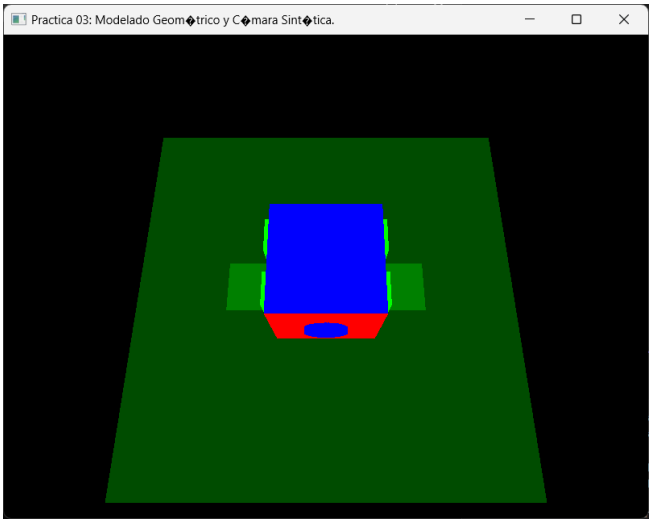

Fig. 28. *Ejecución de programa con cámara estática.*

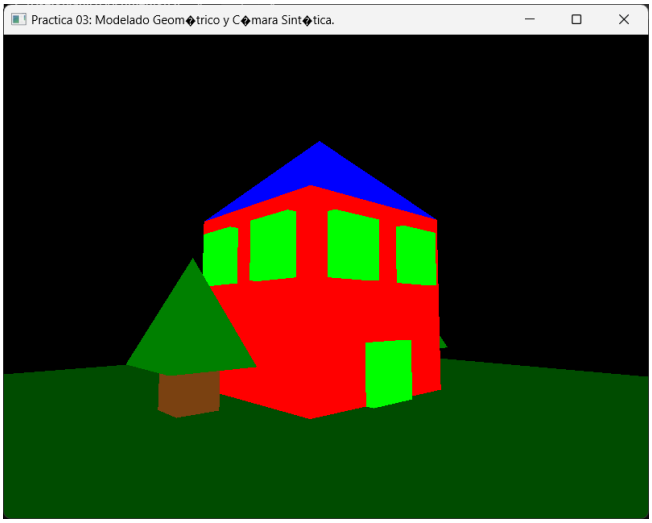

Fig. 29. *Ejecución de programa con cámara en movimiento.*

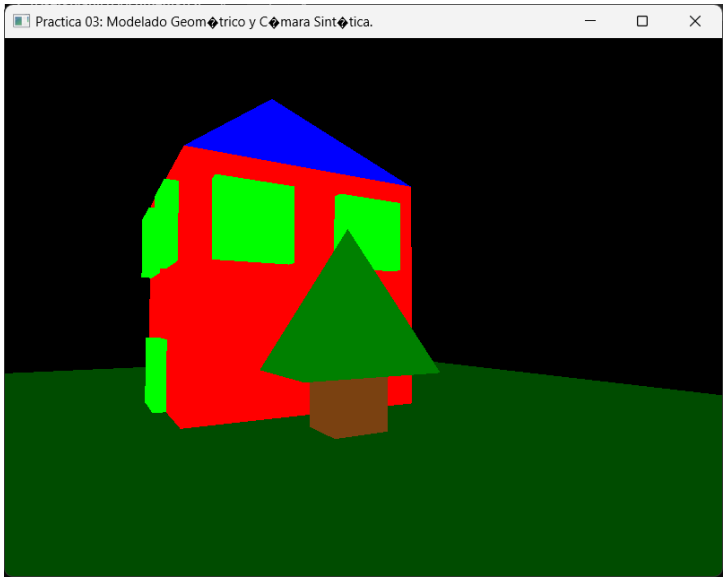

Fig. 30. *Ejecución de programa con cámara en movimiento.*

2. *Problemas presentados.*

En general en esta actividad no presente problemas, lo complicado fue hacer las modificaciones al cilindro, para que este dejara de estar en dirección paralela con el eje Y, ya que lo necesitaba en dirección paralela al eje Y para hacer la ventana y también escoger el tamaño de largo del cilindro, pero después de analizar el código imagine una solución dándole los parámetros para sus dimensiones y posición, por lo demás no tuve otros problemas.

3. *Conclusión:*

Después de realizar los ejercicios propuestos, considero que la complejidad que tenían era normal, realmente teniendo los códigos de las figuras de la practica anterior, pero eso no quita lo difícil que fue entender la creación del cilindro, esta vez considero que

no faltara explicar algún elemento de los ejercicios, todo se entendió. Esta practica fue un tanto complementaria con la anterior ya que abarcamos figuras que aún no habíamos visto y que eran necesario conocerlas para poder utilizarlas en el proyecto de la materia. Al final considero que la practica tuvo un buen nivel.