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: Organista Alvarez Ricardo

Nº de Cuenta: 421018596

GRUPO DE LABORATORIO: 01

GRUPO DE TEORÍA: 04

SEMESTRE 2024-2

FECHA DE ENTREGA LÍMITE: 24/02/2024

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.

a. **Dibujar las iniciales de sus nombres, cada una con un color diferente.**

Se agregan los vértices de las letras previamente creadas en la practica anterior, ahora agregando también los valores de color a cada triangulo que se va a generar.

Vértices de la letra *R* con su respectivo color:

```
GLfloat vertices_R[] = {
        //X         Y           Z           R       G       B
        -0.9f,0.4f,0.0f,        0.5f,1.0f,0.0f,
        -0.8f,0.4f,0.0f,        0.5f,1.0f,0.0f,
        -0.9f,-0.4f,0.0f,       0.5f,1.0f,0.0f,
        -0.8f,0.4f,0.0f,        0.5f,1.0f,0.0f,
        -0.9f,-0.4f,0.0f,       0.5f,1.0f,0.0f,
        -0.8f,-0.4f,0.0f,       0.5f,1.0f,0.0f,
        -0.8f,0.4f,0.0f,        0.5f,1.0f,0.0f,
        -0.8f,0.3f,0.0f,        0.5f,1.0f,0.0f,
        -0.5f,0.4f,0.0f,        0.5f,1.0f,0.0f,
        -0.5f,0.4f,0.0f,        0.5f,1.0f,0.0f,
        -0.8f,0.3f,0.0f,        0.5f,1.0f,0.0f,
        -0.5f,0.3f,0.0f,        0.5f,1.0f,0.0f,
        -0.8f,0.2f,0.0f,        0.5f,1.0f,0.0f,
        -0.8f,0.1f,0.0f,        0.5f,1.0f,0.0f,
        -0.5f,0.2f,0.0f,        0.5f,1.0f,0.0f,
        -0.8f,0.1f,0.0f,        0.5f,1.0f,0.0f,
        -0.5f,0.2f,0.0f,        0.5f,1.0f,0.0f,
        -0.5f,0.1f,0.0f,        0.5f,1.0f,0.0f,
        -0.5f,0.4f,0.0f,        0.5f,1.0f,0.0f,
        -0.5f,0.1f,0.0f,        0.5f,1.0f,0.0f,
        -0.4f,0.4f,0.0f,        0.5f,1.0f,0.0f,
        -0.5f,0.1f,0.0f,        0.5f,1.0f,0.0f,
        -0.4f,0.4f,0.0f,        0.5f,1.0f,0.0f,
        -0.4f,0.1f,0.0f,        0.5f,1.0f,0.0f,
        -0.8f,0.1f,0.0f,        0.5f,1.0f,0.0f,
        -0.7f,0.1f,0.0f,        0.5f,1.0f,0.0f,
        -0.7f,-0.05f,0.0f,      0.5f,1.0f,0.0f,
        -0.7f,0.1f,0.0f,        0.5f,1.0f,0.0f,
        -0.7f,-0.05f,0.0f,      0.5f,1.0f,0.0f,
        -0.6f,-0.05f,0.0f,      0.5f,1.0f,0.0f,
        -0.7f,-0.05f,0.0f,      0.5f,1.0f,0.0f,
        -0.6f,-0.05f,0.0f,      0.5f,1.0f,0.0f,
        -0.6f,-0.2f,0.0f,       0.5f,1.0f,0.0f,
        -0.6f,-0.05f,0.0f,      0.5f,1.0f,0.0f,
        -0.6f,-0.2f,0.0f,       0.5f,1.0f,0.0f,
        -0.5f,-0.2f,0.0f,       0.5f,1.0f,0.0f,
```

```
        -0.6f,-0.2f,0.0f,       0.5f,1.0f,0.0f,
        -0.5f,-0.2f,0.0f,       0.5f,1.0f,0.0f,
        -0.5f,-0.35f,0.0f,      0.5f,1.0f,0.0f,
        -0.5f,-0.2f,0.0f,       0.5f,1.0f,0.0f,
        -0.5f,-0.35f,0.0f,      0.5f,1.0f,0.0f,
        -0.4f,-0.35f,0.0f,      0.5f,1.0f,0.0f,
        -0.5f,-0.35f,0.0f,      0.5f,1.0f,0.0f,
        -0.4f,-0.35f,0.0f,      0.5f,1.0f,0.0f,
        -0.4f,-0.4f,0.0f,       0.5f,1.0f,0.0f,
        -0.5f,-0.35f,0.0f,      0.5f,1.0f,0.0f,
        -0.4f,-0.4f,0.0f,       0.5f,1.0f,0.0f,
        -0.5f,-0.4f,0.0f,       0.5f,1.0f,0.0f,
        /*1.0f, 1.0f,       0.5f,           1.0f,   0.0f,   0.0f,
        -1.0f,  1.0f,       0.5f,           1.0f,   0.0f,   0.0f,
        -1.0f,  -1.0f,      0.5f,           1.0f,   0.0f,   0.0f,*/

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

Vértices de la letra *O*:

```
GLfloat vertices_O[] = {
        -0.2f,0.4f,0.0f,    0.0f,0.8f,0.6f,
        -0.2f,-0.4f,0.0f,   0.0f,0.8f,0.6f,
        -0.1f,0.4f,0.0f,    0.0f,0.8f,0.6f,
        -0.2f,-0.4f,0.0f,   0.0f,0.8f,0.6f,
        -0.1f,0.4f,0.0f,    0.0f,0.8f,0.6f,
        -0.1f,-0.4f,0.0f,   0.0f,0.8f,0.6f,
        -0.1f,0.4f,0.0f,    0.0f,0.8f,0.6f,
        -0.1f,0.3f,0.0f,    0.0f,0.8f,0.6f,
        0.1f,0.4f,0.0f,     0.0f,0.8f,0.6f,
        -0.1f,0.3f,0.0f,    0.0f,0.8f,0.6f,
        0.1f,0.4f,0.0f,     0.0f,0.8f,0.6f,
        0.1f,0.3f,0.0f,     0.0f,0.8f,0.6f,
        -0.1f,-0.4f,0.0f,   0.0f,0.8f,0.6f,
        -0.1f,-0.3f,0.0f,   0.0f,0.8f,0.6f,
        0.1f,-0.3f,0.0f,    0.0f,0.8f,0.6f,
        -0.1f,-0.4f,0.0f,   0.0f,0.8f,0.6f,
        0.1f,-0.3f,0.0f,    0.0f,0.8f,0.6f,
        0.1f,-0.4f,0.0f,    0.0f,0.8f,0.6f,
        0.1f,0.4f,0.0f,     0.0f,0.8f,0.6f,
        0.1f,-0.4f,0.0f,    0.0f,0.8f,0.6f,
        0.2f,0.4f,0.0f,     0.0f,0.8f,0.6f,
        0.1f,-0.4f,0.0f,    0.0f,0.8f,0.6f,
        0.2f,0.4f,0.0f,     0.0f,0.8f,0.6f,
        0.2f,-0.4f,0.0f,    0.0f,0.8f,0.6f,

};
MeshColor* letras_O = new MeshColor();
letras_O->CreateMeshColor(vertices_O, 144);
meshColorList.push_back(letras_O);
```

Vértices de la letra *A*:

```
GLfloat vertices_A[] = {
        0.4f,0.4f,0.0f,     0.7f,0.3f,0.1f,
        0.4f,-0.4f,0.0f,    0.7f,0.3f,0.1f,
        0.55f,0.4f,0.0f,    0.7f,0.3f,0.1f,
        0.4f,-0.4f,0.0f,    0.7f,0.3f,0.1f,
        0.55f,0.4f,0.0f,    0.7f,0.3f,0.1f,
        0.55f,-0.4f,0.0f,   0.7f,0.3f,0.1f,
        0.55f,0.4f,0.0f,    0.7f,0.3f,0.1f,
        0.55f,0.3f,0.0f,    0.7f,0.3f,0.1f,
        0.7f,0.4f,0.0f,     0.7f,0.3f,0.1f,
        0.55f,0.3f,0.0f,    0.7f,0.3f,0.1f,
        0.7f,0.4f,0.0f,     0.7f,0.3f,0.1f,
        0.7f,0.3f,0.0f,     0.7f,0.3f,0.1f,
        0.55f,0.1f,0.0f,    0.7f,0.3f,0.1f,
        0.55f,0.0f,0.0f,    0.7f,0.3f,0.1f,
        0.7f,0.1f,0.0f,     0.7f,0.3f,0.1f,
        0.55f,0.0f,0.0f,    0.7f,0.3f,0.1f,
        0.7f,0.1f,0.0f,     0.7f,0.3f,0.1f,
        0.7f,0.0f,0.0f,     0.7f,0.3f,0.1f,
        0.7f,0.4f,0.0f,     0.7f,0.3f,0.1f,
        0.85f,0.4f,0.0f,    0.7f,0.3f,0.1f,
        0.7f,-0.4f,0.0f,    0.7f,0.3f,0.1f,
        0.85f,0.4f,0.0f,    0.7f,0.3f,0.1f,
        0.7f,-0.4f,0.0f,    0.7f,0.3f,0.1f,
        0.85f,-0.4f,0.0f,   0.7f,0.3f,0.1f,
};
MeshColor* letras_A = new MeshColor();
letras_A->CreateMeshColor(vertices_A, 144);
meshColorList.push_back(letras_A);
```

Para mostrar las letras debemos indicar que elementos se van a renderizar, agregamos sus índices para que se muestren las tres a la vez.

Código para mostrar las letras:

```
//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, 0.0f, -4.0f));
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA y se envían al sha
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[0]->RenderMeshColor();
meshColorList[1]->RenderMeshColor();
meshColorList[2]->RenderMeshColor();
```

Resultado:


C:\Users\ricar\Documents\20:
se presiono la tecla 283'
se solto la tecla 283'
Practica 2: Proyecciones, transformaciones
R O A

b**. Generar el dibujo de la casa de clase, pero en lugar de instanciar triángulos 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 color clamp.**

Primero creamos los shaders para los diferentes colores que vamos a utilizar. Los creamos de la siguiente forma, cambiando los primeros tres valores del vector de color para obtener cada uno de los colores.

```
#version 330
layout (location =0) in vec3 pos;
layout (location =1) in vec3 color;
out vec4 vColor;
uniform mat4 model;
uniform mat4 projection;
void main()
{
    gl_Position=projection*model*vec4(pos.x,pos.y,pos.z,1.0f);
    vColor=vec4(0.0f,0.0f,1.0f,1.0f);
}
```

```
#version 330
layout (location =0) in vec3 pos;
layout (location =1) in vec3 color;
out vec4 vColor;
uniform mat4 model;
uniform mat4 projection;
void main()
{
    gl_Position=projection*model*vec4(pos.x,pos.y,pos.z,1.0f);
    vColor=vec4(0.478f,0.255f,0.067f,1.0f);
}
```

```
#version 330
layout (location =0) in vec3 pos;
layout (location =1) in vec3 color;
out vec4 vColor;
uniform mat4 model;
uniform mat4 projection;
void main()
{
    gl_Position=projection*model*vec4(pos.x,pos.y,pos.z,1.0f);
    vColor=vec4(0.0f,0.5f,0.0f,1.0f);
}
```

```
#version 330
layout (location =0) in vec3 pos;
layout (location =1) in vec3 color;
out vec4 vColor;
uniform mat4 model;
uniform mat4 projection;
void main()
{
    gl_Position=projection*model*vec4(pos.x,pos.y,pos.z,1.0f);
    vColor=vec4(0.0f,1.0f,0.0f,1.0f);
}
```

```
#version 330
layout (location =0) in vec3 pos;
layout (location =1) in vec3 color;
out vec4 vColor;
uniform mat4 model;
uniform mat4 projection;
void main()
{
    gl_Position=projection*model*vec4(pos.x,pos.y,pos.z,1.0f);
    vColor=vec4(1.0f,0.0f,0.0f,1.0f);
}
```

Después agregamos los shaders a nuestro programa para poder usarlos:

```
//shaders nuevos se crearían acá
static const char* vShaderRojo = "shaders/shaderRojo.vert";
static const char* vShaderVerde = "shaders/shaderVerde.vert";
static const char* vShaderAzul = "shaders/shaderAzul.vert";
static const char* vShaderCafe = "shaders/shaderCafe.vert";
static const char* vShaderVerdeOsc = "shaders/shaderVerdeOsc.vert";
```

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

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

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

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

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

Ahora colocamos las figuras en sus posiciones adecuadas para obtener el dibujo. Esto lo haremos mediante el escalado de las figuras para que tengan el tamaño adecuado y trasladándolas a sus posiciones correspondientes.

```
//Cubo Casa
shaderList[2].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
//Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para alma
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.25f, -3.0f));
model = glm::scale(model, glm::vec3(1.0f, 1.2f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES P
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();

// Cubo Puerta
shaderList[3].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.65f, -2.0f));
model = glm::scale(model, glm::vec3(0.3f, 0.4f, 0.35f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES P
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();

// Cubo Ventana Der
shaderList[3].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.3f, 0.1f, -2.0f));
model = glm::scale(model, glm::vec3(0.3f, 0.4f, 0.35f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES P
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

```
// Cubo Ventana Izq
shaderList[3].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.3f, 0.1f, -2.0f));
model = glm::scale(model, glm::vec3(0.3f, 0.4f, 0.35f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();

// Piramide Techo
shaderList[4].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.65f, -3.0f));
model = glm::scale(model, glm::vec3(1.35f, 0.6f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();

// Cubo Tronco Izq
shaderList[5].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.75f, -0.75f, -3.0f));
model = glm::scale(model, glm::vec3(0.1f, 0.2f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

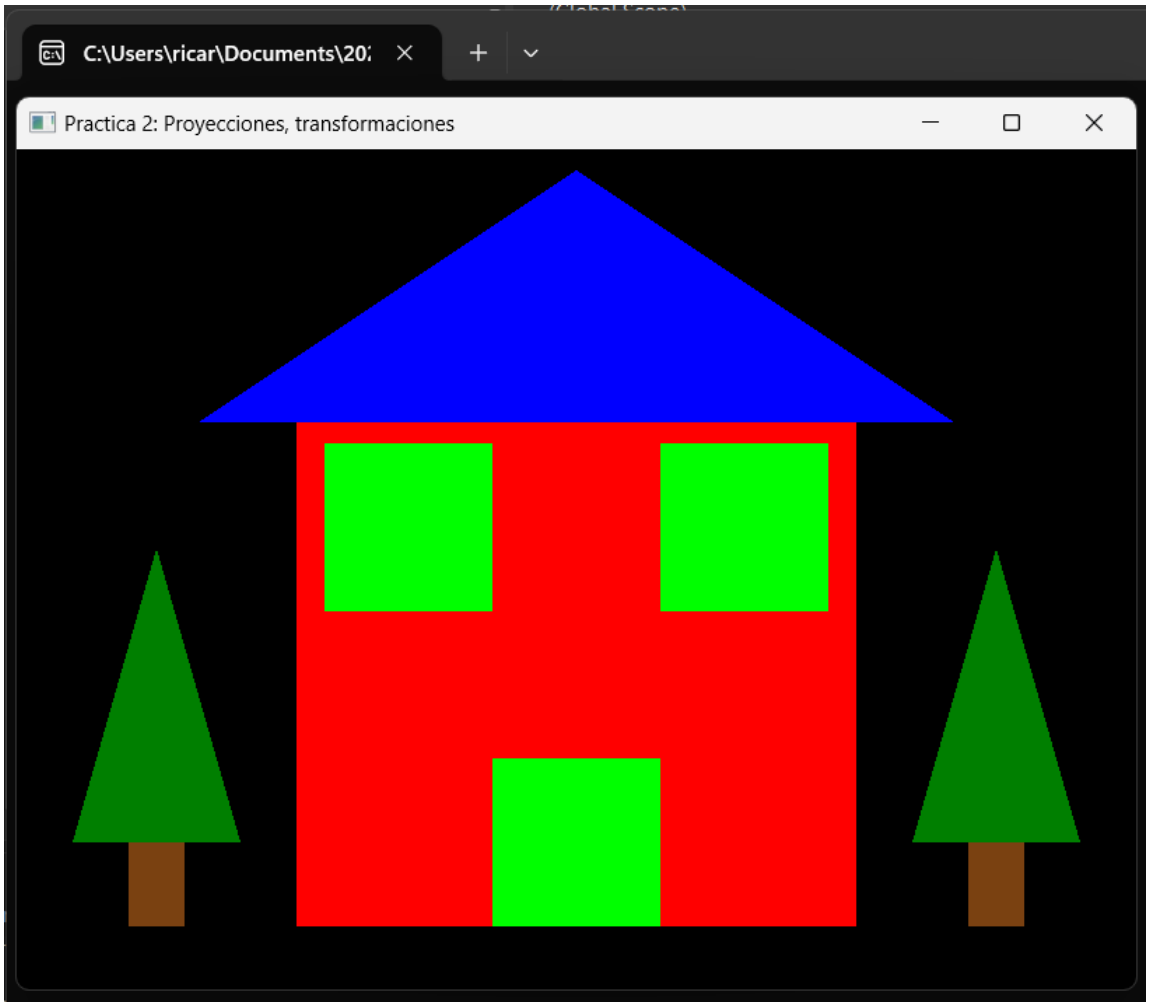
```
// Cubo Tronco Der
shaderList[5].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.75f, -0.75f, -3.0f));
model = glm::scale(model, glm::vec3(0.1f, 0.2f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();

// Piramide Arbol Izq
shaderList[6].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.75f, -0.3f, -3.0f));
model = glm::scale(model, glm::vec3(0.3f, 0.7f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();

// Piramide Arbol Der
shaderList[6].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.75f, -0.3f, -3.0f));
model = glm::scale(model, glm::vec3(0.3f, 0.7f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();

glUseProgram(0);
mainWindow.swapBuffers();
```

Resultado:



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

No hubo problemas al crear las figuras, y gracias a las explicaciones de la clase fue sencillo crear los shaders necesarios para cada color de la actividad dos.

3.- Conclusión:

a. Los ejercicios del reporte:
   Los ejercicios tienen buen nivel, gracias a las explicaciones de la clase es sencillo realizarlos correctamente, con esto pongo en práctica y aprendo más conceptos que serán útiles para otras prácticas y el proyecto final.
b. Comentarios generales:
   La explicación fue entendible y sirvió para resolver todo lo solicitado.
c. Conclusión:
   Gracias a esta practica pude poner en practica mas conceptos, como la perspectiva, los shaders para el color, así como ver por primera vez como generar objetos en tres dimensiones.