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: Vázquez Reyes Sebastián

Nº de Cuenta: 318150923

GRUPO DE LABORATORIO: 11

GRUPO DE TEORÍA: 6

SEMESTRE 2024-2

FECHA DE ENTREGA LÍMITE: miércoles 21 de febrero de 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.

- Dibujar las iniciales de sus nombres, cada letra de un color diferente

Para esta sección del código solamente tuve que copiar los vértices generados en la práctica anterior para mis iniciales, y agregar unos cuantos vértices mas para indicar el color de las letras.

Tenemos los vertices para la letra S

```
GLfloat vertices_S[] = {
    //X         Y           Z           R       G       B
    -0.8f,      -0.5f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.8f,      -0.3f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.5f,      -0.5f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.8f,      -0.3f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.5f,      -0.5f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.5f,      -0.3f,      0.0f,       1.0f,   0.0f,   0.0f,

    -0.5f,      -0.5f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.5f,      +0.1f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.65f,     -0.5f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.5f,      +0.1f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.65f,     -0.5f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.65f,     +0.1f,      0.0f,       1.0f,   0.0f,   0.0f,

    -0.8f,      -0.1f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.8f,      +0.1f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.5f,      -0.1f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.8f,      +0.1f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.5f,      -0.1f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.5f,      +0.1f,      0.0f,       1.0f,   0.0f,   0.0f,

    -0.8f,      -0.1f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.8f,      +0.3f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.65f,     -0.1f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.8f,      +0.3f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.65f,     -0.1f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.65f,     +0.3f,      0.0f,       1.0f,   0.0f,   0.0f,

    -0.8f,      +0.5f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.8f,      +0.3f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.5f,      +0.5f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.8f,      +0.3f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.5f,      +0.5f,      0.0f,       1.0f,   0.0f,   0.0f,
    -0.5f,      +0.3f,      0.0f,       1.0f,   0.0f,   0.0f,

};
MeshColor* letraS = new MeshColor();
```

Para la letra V

```
GLfloat vertices_V[] = {
    //X        Y          Z          R       G       B
    -0.2f,     -0.5f,     0.0f,      0.0f,   1.0f,   0.0f,
    -0.2f,     -0.3f,     0.0f,      0.0f,   1.0f,   0.0f,
    +0.1f,     -0.5f,     0.0f,      0.0f,   1.0f,   0.0f,
    -0.2f,     -0.3f,     0.0f,      0.0f,   1.0f,   0.0f,
    +0.1f,     -0.5f,     0.0f,      0.0f,   1.0f,   0.0f,
    +0.1f,     -0.3f,     0.0f,      0.0f,   1.0f,   0.0f,

    -0.35f,    +0.5f,     0.0f,      0.0f,   1.0f,   0.0f,
    -0.2f,     -0.5f,     0.0f,      0.0f,   1.0f,   0.0f,
    -0.05f,    -0.3f,     0.0f,      0.0f,   1.0f,   0.0f,
    -0.35f,    +0.5f,     0.0f,      0.0f,   1.0f,   0.0f,
    -0.05f,    -0.3f,     0.0f,      0.0f,   1.0f,   0.0f,
    -0.25f,    +0.5f,     0.0f,      0.0f,   1.0f,   0.0f,

    +0.25f,    +0.5f,     0.0f,      0.0f,   1.0f,   0.0f,
    +0.1f,     -0.5f,     0.0f,      0.0f,   1.0f,   0.0f,
    -0.05f,    -0.3f,     0.0f,      0.0f,   1.0f,   0.0f,
    +0.25f,    +0.5f,     0.0f,      0.0f,   1.0f,   0.0f,
    -0.05f,    -0.3f,     0.0f,      0.0f,   1.0f,   0.0f,
    +0.15f,    +0.5f,     0.0f,      0.0f,   1.0f,   0.0f,


};
MeshColor* letraV = new MeshColor();
letraV->CreateMeshColor(vertices_V, 108);
meshColorList.push_back(letraV);
```

Y para la letra R

```
GLfloat vertices_R[] = {
    //X        Y          Z          R       G       B
    +0.8f,     +0.3f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.8f,     +0.5f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.5f,     +0.3f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.8f,     +0.5f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.5f,     +0.3f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.5f,     +0.5f,     0.0f,      0.0f,   0.0f,   1.0f,

    +0.4f,     +0.5f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.4f,     -0.5f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.55f,    -0.5f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.4f,     +0.5f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.55f,    -0.5f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.55f,    +0.5f,     0.0f,      0.0f,   0.0f,   1.0f,

    +0.8f,     +0.1f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.8f,     -0.1f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.5f,     +0.1f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.8f,     -0.1f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.5f,     +0.1f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.5f,     -0.1f,     0.0f,      0.0f,   0.0f,   1.0f,

    +0.8f,     +0.5f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.8f,     -0.1f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.65f,    -0.1f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.8f,     +0.5f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.65f,    -0.1f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.65f,    +0.5f,     0.0f,      0.0f,   0.0f,   1.0f,

    +0.8f,     -0.5f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.65f,    -0.5f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.5f,     -0.1f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.8f,     -0.5f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.65f,    -0.1f,     0.0f,      0.0f,   0.0f,   1.0f,
    +0.5f,     -0.1f,     0.0f,      0.0f,   0.0f,   1.0f,
```

Y para mostrarlas en la ventana utilizamos el mismo código del ejercicio anterior, solo que esta vez usando índices diferentes, los correspondientes a las letras

```
shaderList[6].useShader();
uniformModel = shaderList[6].getModelLocation();
uniformProjection = shaderList[6].getProjectLocation();

//Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para alm
//S
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.3f, +2.0f, -4.0f));
model = glm::scale(model, glm::vec3(0.9f, 0.7f, 0.7f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[0]->RenderMeshColor();

//V
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(+0.0f, +2.0f, -4.0f));
model = glm::scale(model, glm::vec3(0.9f, 0.7f, 0.7f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshColorList[1]->RenderMeshColor();

//R
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(+0.3f, +2.0f, -4.0f));
model = glm::scale(model, glm::vec3(0.9f, 0.7f, 0.7f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshColorList[2]->RenderMeshColor();

angulo += 0.1f;
```

- Generar el dibujo de la casa de la 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 shader con el color clamp

Para crear cada shader necesita crear 2 archivos por color, el .vert y el .frag. El .frag es el mismo código para todos los shader, lo único que cambia es el nombre del archivo. Para el .vert, la única línea que cambia es la última, en donde ahora colocamos los valores del color que queremos, por ejemplo, para el .vert del shader café utilizamos el siguiente código

dercafe.vert | shadercafe.frag | shadercolor.vert | P02-318150923.

```
1   #version 330
2   layout (location =0) in vec3 pos;
3   out vec4 vColor;
4   uniform mat4 model;
5   uniform mat4 projection;
6   void main()
7   {
8       gl_Position=projection*model*vec4(pos,1.0f);
9       //vColor=vec4(color,1.0f);
10      vColor=vec4(0.478, 0.255, 0.067, 0.8f);
11  }
```

Lo único que cambia son los valores del vector de 4 dimensiones, ahora toman los valores de los colores que queremos.

Luego, todos los nuevos shaders son creados en el código main, primero obteniendo su dirección del directorio:

```
//Solo manda como salida el dato que mando shader.vert
static const char* vShaderColor = "shaders/shadercolor.vert";
static const char* fShaderColor = "shaders/shadercolor.frag";
static const char* vShaderAzul = "shaders/shaderazul.vert";
static const char* fShaderAzul = "shaders/shaderazul.frag";
static const char* vShaderVerde = "shaders/shaderverde.vert";
static const char* fShaderVerde = "shaders/shaderverde.frag";
static const char* vShaderRojo = "shaders/shaderrojo.vert";
static const char* fShaderRojo = "shaders/shaderrojo.frag";
static const char* vShaderCafe = "shaders/shadercafe.vert";
static const char* fShaderCafe = "shaders/shadercafe.frag";
static const char* vShaderVerdeOsc = "shaders/shaderverdeoscuro.vert";
static const char* fShaderVerdeOsc = "shaders/shaderverdeoscuro.frag";
//shaders nuevos se crearian acá
```

Y luego los creamos con la función correspondiente:

```
Shader* shader1 = new Shader(); //shader para usar índices: objetos: cubo y  pirámide. Shader arcoiris
shader1->CreateFromFiles(vShader, fShader);
shaderList.push_back(*shader1);

Shader* shader2 = new Shader(); //shader verde
shader2->CreateFromFiles(vShaderVerde, fShaderVerde);
shaderList.push_back(*shader2);

Shader* shader3 = new Shader(); //shader azul
shader3->CreateFromFiles(vShaderAzul, fShaderAzul);
shaderList.push_back(*shader3);

Shader* shader4 = new Shader(); //shader rojo
shader4->CreateFromFiles(vShaderRojo, fShaderRojo);
shaderList.push_back(*shader4);

Shader* shader5 = new Shader(); //shader verde oscuro
shader5->CreateFromFiles(vShaderCafe, fShaderCafe);
shaderList.push_back(*shader5);

Shader* shader6 = new Shader(); //shader cafe
shader6->CreateFromFiles(vShaderVerdeOsc, fShaderVerdeOsc);
shaderList.push_back(*shader6);

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

Ademas, cree una nueva funcion llamada CrearPiramideInversa(), que crea una pirámide de la misma forma y tamaño que la funcion original, solo que esta cambia un vertice para que uno de sus vértices apunte a la dirección contraria que la otra pirámide lo hace. De esta manera, al crearlas en el mismo espacio, forman una pirámide cuadrangular:

```
//Nueva funcion, similar a la anterior, solo para
void CreaPiramideinversa()
{
    unsigned int indices[] = {
        0,1,2,
        1,3,2,
        3,0,2,
        1,0,3

    };
    GLfloat vertices[] = {
        -0.5f, -0.5f,0.0f,  //0
        0.5f,-0.5f,0.0f,    //1
        0.0f,0.5f, 0.0f,    //2
        0.0f,-0.5f, +0.5f,  //3

    };
    Mesh* obj2 = new Mesh();
    obj2->CreateMesh(vertices, indices, 12, 12);
    meshList.push_back(obj2);
}
```

Por último, para formar la casa se tuvo que crear cada objeto con un respectivo shader para darle un color distinto a cada uno.

Para el cuerpo de la casa:

```
// CUBO ROJO
shaderList[3].useShader();
uniformModel = shaderList[3].getModelLocation();
uniformProjection = shaderList[3].getProjectLocation();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(+0.0f, -0.6f, -4.0f));
model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[2]->RenderMesh();
```

Para el techo de la casa:

```
// PIRAMIDE AZUL
shaderList[2].useShader();
uniformModel = shaderList[2].getModelLocation();
uniformProjection = shaderList[2].getProjectLocation();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(+0.0f, +0.9f, -4.0f));
model = glm::scale(model, glm::vec3(4.0f, 1.0f, 5.0f));
model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(+0.0f, +0.9f, -4.0f));
model = glm::scale(model, glm::vec3(3.5f, 1.0f, 5.0f));
model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[1]->RenderMesh();
```

Para la puerta y ventanas:

```
//CUBOS VERDES
shaderList[1].useShader();
uniformModel = shaderList[1].getModelLocation();
uniformProjection = shaderList[1].getProjectLocation();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(+0.0f, -1.2f, -3.0f));
model = glm::scale(model, glm::vec3(0.35f, 0.35f, 0.35f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[2]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(+0.55f, -0.1f, -3.0f));
model = glm::scale(model, glm::vec3(0.35f, 0.35f, 0.35f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[2]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.55f, -0.1f, -3.0f));
model = glm::scale(model, glm::vec3(0.35f, 0.35f, 0.35f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[2]->RenderMesh();
```

Para los árboles a los costados:

```
//PIRAMIDES VERDES OSCURO
shaderList[5].useShader();
uniformModel = shaderList[5].getModelLocation();
uniformProjection = shaderList[5].getProjectLocation();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-2.0f, -0.5f, -3.8f));
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-2.0f, -0.5f, -3.8f));
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[1]->RenderMesh(); //Las piramides con indice 1 estan pegadas a las de ind

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(+2.0f, -0.5f, -3.8f));
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[0]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(+2.0f, -0.5f, -3.8f));
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[1]->RenderMesh();
```

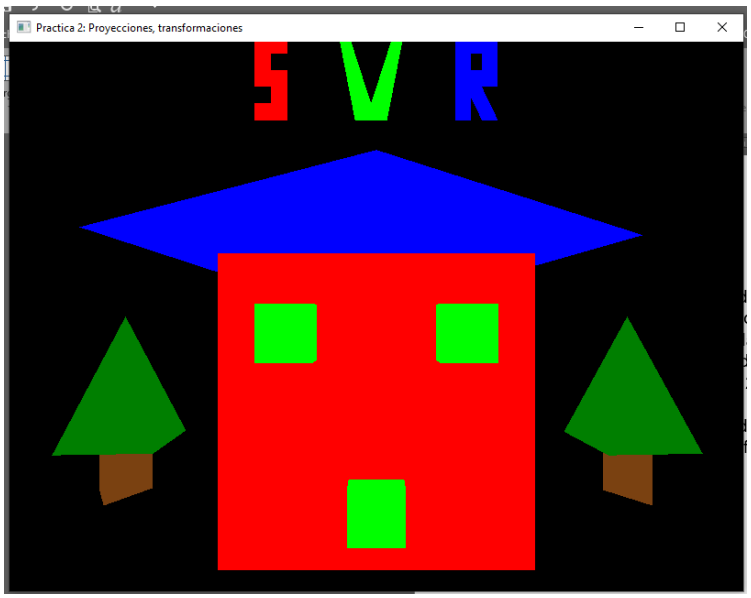
```
//CUBOS CAFES
shaderList[4].useShader();
uniformModel = shaderList[4].getModelLocation();
uniformProjection = shaderList[4].getProjectLocation();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-2.1f, -1.2f, -4.0f));
model = glm::scale(model, glm::vec3(0.3f, 0.5f, 0.5f));
model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[2]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(+2.1f, -1.2f, -4.0f));
model = glm::scale(model, glm::vec3(0.3f, 0.5f, 0.5f));
model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[2]->RenderMesh();

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

Y la ejecución se ve así:



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

El único problema que presente fue en la rotación de la casa, no averigüe una forma de hacerla rotar junto con sus puertas y ventanas sin que se viera encimado el cubo rojo enorme. Debido a esto, decidí dejarlos estáticos, mientras que el resto de elementos geométricos si están rotando, para que se note que son figuras 3D.

3.- Conclusión:

a. Los ejercicios del reporte: Complejidad, Explicación.

El primer ejercicio realmente fue sencillo, únicamente fue copiar y pegar. Para el segundo ejercicio solo me atore unos cuantos minutos intentando averiguar como crear los shader. Pero una vez que entendí como crear shaders de colores distintos, el resto fue fácil. Sin embargo, reitero, no averigüe una forma de "incrustar" los cubos verdes en el cubo rojo, para que cuando rotaran, los cubos verdes solo aparecieran en la cara frontal del cubo rojo como en una casa normal, así que decidí dejarlos sin rotar, aunque esto significa que ambos tipos de figuras parezcan más una figura de tipo 2D en vez de 3D.

b. Comentarios generales: Faltó explicar a detalle, ir más lento en alguna explicación, otros comentarios y sugerencias para mejorar desarrollo de la práctica

Explicar el código durante la sesión me ahorro muchísimo tiempo para la realización de la práctica, gracias a eso no me tomo mucho tiempo entender de nuevo la construcción de los shader, y como realizar las transformaciones para las figuras 2D y 3D sin atorarme mucho.

c. Conclusión

Al finalizar la práctica aprendí a cómo manejar las transformaciones de rotación, traslación y escalado para crear y trabajar figuras geométricas 3D y 2D, además, también aprendí a como crear shaders para dar color a mis figuras. También aprendí a cómo manejar vértices utilizando índices para mis próximas construcciones.

Bibliografía

- Dpto. de Ciencias e Ingeniería de la Computación, "Computación Gráfica". Universidad Nacional del Sur. Recuperado el 20 de febrero de 2024 de http://www.cs.uns.edu.ar/cg/clasespdf/3-Pipe3D.pdf