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 Gómez Carlos Iván

Nº de Cuenta: 4200551851

GRUPO DE LABORATORIO: 03

GRUPO DE TEORÍA: 04

SEMESTRE 2025-1

FECHA DE ENTREGA LÍMITE: 24/08/2024

CALIFICACIÓN: _____

REPORTE DE PRÁCTICA:

1.- Ejercicios planteados

Primera actividad

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

Para realizar este ejercicio fue necesario agregar las letras creadas en la práctica anterior, la única diferencia fue que ahora pondríamos los parámetros de color en cada una de las letras, como se muestra a continuación.

Para la letra C

```
GLfloat vertices_letras[] = {

    //LETRA C
        // Primer Triangulo
    //X         Y           Z           R       G       B
    -0.9f,      0.6f,       0.0f,       0.05f, 0.0f, 0.0f,
    -0.9f,      0.3f,       0.0f,       0.05f, 0.0f, 0.0f,
    -0.45f,     0.6f,       0.0f,       0.05f, 0.0f, 0.0f,

    // Segundo Triangulo
    //X         Y           Z           R       G       B
    -0.9f,      0.3f,       0.0f,       0.05f, 0.0f, 0.0f,
    -0.45f,     0.6f,       0.0f,       0.05f, 0.0f, 0.0f,
    -0.45f,     0.3f,       0.0f,       0.05f, 0.0f, 0.0f,

    // Tercer Triangulo
    //X         Y           Z           R       G       B
    -0.9f,      0.3f,       0.0f,       0.05f, 0.0f, 0.0f,
    -0.9f,      -0.4f,      0.0f,       0.05f, 0.0f, 0.0f,
    -0.7f,      0.3f,       0.0f,       0.05f, 0.0f, 0.0f,

    // Cuarto Triangulo
    //X         Y           Z           R       G       B
    -0.9f,      -0.4f,      0.0f,       0.05f, 0.0f, 0.0f,
    -0.7f,      -0.4f,      0.0f,       0.05f, 0.0f, 0.0f,
    -0.7f,      0.3f,       0.0f,       0.05f, 0.0f, 0.0f,

    // Quinto Triangulo
    //X         Y           Z           R       G       B
    -0.9f,      -0.6f,      0.0f,       0.05f, 0.0f, 0.0f,
    -0.9f,      -0.3f,      0.0f,       0.05f, 0.0f, 0.0f,
    -0.45f,     -0.6f,      0.0f,       0.05f, 0.0f, 0.0f,

    // Sexto Triangulo
    //X         Y           Z           R       G       B
    -0.9f,      -0.3f,      0.0f,       0.05f, 0.0f, 0.0f,
    -0.45f,     -0.6f,      0.0f,       0.05f, 0.0f, 0.0f,
    -0.45f,     -0.3f,      0.0f,       0.05f, 0.0f, 0.0f,
```

Para la letra i mayúscula

```
//LETRA I

// Primer Triangulo COLOR TURQUESA
//X          Y          Z          R       G       B
-0.4f,      0.6f,      0.0f,      0.25f, 0.88f, 0.82f,
-0.4f,      0.3f,      0.0f,      0.25f, 0.88f, 0.82f,
0.1f,       0.6f,      0.0f,      0.25f, 0.88f, 0.82f,

// Segundo Triangulo
-0.4f,      0.3f,      0.0f,      0.25f, 0.88f, 0.82f,
0.1f,       0.6f,      0.0f,      0.25f, 0.88f, 0.82f,
0.1f,       0.3f,      0.0f,      0.25f, 0.88f, 0.82f,

// Tercer Triangulo
-0.25f,     0.3f,      0.0f,      0.25f, 0.88f, 0.82f,
-0.05f,     0.3f,      0.0f,      0.25f, 0.88f, 0.82f,
-0.05f,     -0.3f,     0.0f,      0.25f, 0.88f, 0.82f,

// Cuarto Triangulo
-0.25f,     0.3f,      0.0f,      0.25f, 0.88f, 0.82f,
-0.25f,     -0.3f,     0.0f,      0.25f, 0.88f, 0.82f,
-0.05f,     -0.3f,     0.0f,      0.25f, 0.88f, 0.82f,

// Quinto Triangulo
-0.4f,      -0.6f,     0.0f,      0.25f, 0.88f, 0.82f,
-0.4f,      -0.3f,     0.0f,      0.25f, 0.88f, 0.82f,
0.1f,       -0.6f,     0.0f,      0.25f, 0.88f, 0.82f,

// Sexto Triangulo
-0.4f,      -0.3f,     0.0f,      0.25f, 0.88f, 0.82f,
0.1f,       -0.6f,     0.0f,      0.25f, 0.88f, 0.82f,
0.1f,       -0.3f,     0.0f,      0.25f, 0.88f, 0.82f,
```

Para la letra G

```
//LETRA G
// Primer Triangulo COLOR Plateado: (0.75, 0.75, 0.75)
//X          Y           Z           R       G       B
0.15f,      0.6f,       0.0f,       0.75f, 0.75f, 0.75f,
0.15f,      0.3f,       0.0f,       0.75f, 0.75f, 0.75f,
0.85f,      0.6f,       0.0f,       0.75f, 0.75f, 0.75f,

// Segundo Triangulo
0.15f,      0.3f,       0.0f,       0.75f, 0.75f, 0.75f,
0.85f,      0.6f,       0.0f,       0.75f, 0.75f, 0.75f,
0.85f,      0.3f,       0.0f,       0.75f, 0.75f, 0.75f,

// Tercer Triangulo
0.15f,      0.3f,       0.0f,       0.75f, 0.75f, 0.75f,
0.15f,      -0.4f,      0.0f,       0.75f, 0.75f, 0.75f,
0.35f,      0.3f,       0.0f,       0.75f, 0.75f, 0.75f,

// Cuarto Triangulo
0.15f,      -0.4f,      0.0f,       0.75f, 0.75f, 0.75f,
0.35f,      -0.4f,      0.0f,       0.75f, 0.75f, 0.75f,
0.35f,      0.3f,       0.0f,       0.75f, 0.75f, 0.75f,

// Quinto Triangulo
0.15f,      -0.6f,      0.0f,       0.75f, 0.75f, 0.75f,
0.15f,      -0.3f,      0.0f,       0.75f, 0.75f, 0.75f,
0.85f,      -0.6f,      0.0f,       0.75f, 0.75f, 0.75f,

// Sexto Triangulo
0.15f,      -0.3f,      0.0f,       0.75f, 0.75f, 0.75f,
0.85f,      -0.6f,      0.0f,       0.75f, 0.75f, 0.75f,
0.85f,      -0.3f,      0.0f,       0.75f, 0.75f, 0.75f,

// Septimo Triangulo
0.65f,      -0.3f,      0.0f,       0.75f, 0.75f, 0.75f,
0.85f,      -0.1f,      0.0f,       0.75f, 0.75f, 0.75f,
0.85f,      -0.3f,      0.0f,       0.75f, 0.75f, 0.75f,
```

```
    // Octavo Triangulo
    0.65f,      -0.1f,      0.0f,       0.75f, 0.75f, 0.75f,
    0.85f,      -0.1f,      0.0f,       0.75f, 0.75f, 0.75f,
    0.65f,      -0.3f,      0.0f,       0.75f, 0.75f, 0.75f,

    // Noveno Triangulo
    0.45f,      -0.1f,      0.0f,       0.75f, 0.75f, 0.75f,
    0.85f,      0.1f,       0.0f,       0.75f, 0.75f, 0.75f,
    0.85f,      -0.1f,      0.0f,       0.75f, 0.75f, 0.75f,

    // Decimo Triangulo
    0.45f,      -0.1f,      0.0f,       0.75f, 0.75f, 0.75f,
    0.45f,      0.1f,       0.0f,       0.75f, 0.75f, 0.75f,
    0.85f,      0.1f,       0.0f,       0.75f, 0.75f, 0.75f,

};
MeshColor *letras = new MeshColor();
letras->CreateMeshColor(vertices_letras,600);
meshColorList.push_back(letras);
```

Para poder verlas en pantalla, se dejo como defecto el shaderList[1] que fue indicado dentro del código proporcionado.

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

A demás, se necesitó de esta sentencia de código para poder verlo en pantalla, y escoger el shader seleccionado para observar los colores.

```
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 shader como variables de tipo uniform
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[0]->RenderMeshColor();
```

Una vez en pantalla, observe que tenia por default el color negro, y yo quería ver mi letra de color negro, por lo tanto, le cambie el color a la pantalla a blanco cambiando la siguiente sentencia.

```
glClearColor(0.0f,0.0f,0.0f,1.0f);
```

Finalmente, se pueden observar las 3 letras de diferentes colores con un fondo blanco escogido por mí. Los colores turquesa y gris fueron investigados y se colocara su bibliografía en el área correspondiente.

Segunda actividad

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 resolver este ejercicio fue necesario crear 5 shaders para cada color solicitado, esto se crearon de la siguiente manera.

shader_rojo.vert: Bloc de notas

Archivo Edición Formato Ver Ayuda

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

Para esto se siguió el código base del archivo “shader.vert”

shader.vert: Bloc de notas

Archivo Edición Formato Ver Ayuda

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

Solo se modificó la última sentencia de “vColor” colocando los parámetros para los colores necesarios, y así fue para cada shader solicitado.

Al observar el código, me di cuenta que no era necesario crear un shader.frag para cada color, ya que compartía la misma lógica para cada uno, por eso solo reutilice el mismo shader frag.

Una vez creados nuestros “shader.vert” de cada color es necesario definir la variable constante que contiene la ruta a nuestros archivos shader, en este caso se encuentran en la carpeta “shaders”.

```
//shaders nuevos se crearían acá
    //Color Verde
 static const char* vShaderColor_Verde = "shaders/shader_verde.vert";

    // Color Rojo
 static const char* vShaderColor_Rojo = "shaders/shader_rojo.vert";

    // Color Azul
 static const char* vShaderColor_Azul = "shaders/shader_azul.vert";

    // Color cafe
 static const char* vShaderColor_Cafe = "shaders/shader_cafe.vert";

    // Color verde Obscuro
 static const char* vShaderColor_VerdeOs = "shaders/shader_verdeos.vert";
```

Ahora, por cada constante creada es necesario crear los shaders en nuestra “CreateShaders” en donde se asigna el numero de shader y los parámetros que agarra, en este caso sería nuestro “.vert” y “.frag” de cada uno de los colores solicitados.

Es importante respetar el orden en el que los fuimos creando para un futuro hacer uso de ellos.

```
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);

    //Shader color verde
    Shader* shader3 = new Shader();
    shader3->CreateFromFiles(vShaderColor_Verde, fShaderColor);
    shaderList.push_back(*shader3);

    //Shader color rojo
    Shader* shader4 = new Shader();
    shader4->CreateFromFiles(vShaderColor_Rojo, fShaderColor);
    shaderList.push_back(*shader4);

    //Shader color azul
    Shader* shader5 = new Shader();
    shader5->CreateFromFiles(vShaderColor_Azul, fShaderColor);
    shaderList.push_back(*shader5);

    //Shader color cafe
    Shader* shader6 = new Shader();
    shader6->CreateFromFiles(vShaderColor_Cafe, fShaderColor);
    shaderList.push_back(*shader6);

    //Shader color verde oscuro
    Shader* shader7 = new Shader();
    shader7->CreateFromFiles(vShaderColor_VerdeOs, fShaderColor);
    shaderList.push_back(*shader7);
};
```

Una vez creados estos shaders ahora podemos hacer uso de esos colores, y realizar la casa con las figuras geométricas colocadas que son pirámide y cubo.

Para el cubo se coloca el shaderList[3], porque el color rojo fue el cuarto en crearse y recordemos que el índice inicia desde 0. Así mismo, el colocado de meshList[1] fue porque el cubo es el segundo en declararse y la pirámide fue la primera.

En cuanto las proyecciones, estas fueron reutilizadas del ejercicio realizado en clase.

```
//Para el cubo rojo
shaderList[3].useShader();
uniformModel = shaderList[3].getModelLocation();
uniformProjection = shaderList[3].getProjectLocation();
angulo += 0.01;
//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, -0.5f, -3.0f));
model = glm::scale(model, glm::vec3(1.0f, -1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

En este caso para el techo nos indica que es el color azul, y en meshList[0] hace referencia a la pirámide. En este caso de las pirámides si tuve que modificar la proyección en escalado, pero no fue mucha la diferencia a comparación del ejercicio hecho en clase.

```
//Para el techo
shaderList[4].useShader();
uniformModel = shaderList[4].getModelLocation();
uniformProjection = shaderList[4].getProjectLocation();
angulo += 0.01;
//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, 0.25f, -2.0f));
model = glm::scale(model, glm::vec3(1.2f, 0.5f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();
```

Al entender el funcionamiento del “shaderList” y “meshList” las demás figuras fue sencillo realizarla, me guie en base al ejercicio realizado en clase.

```
//Ventana izquierda
shaderList[2].useShader();
uniformModel = shaderList[2].getModelLocation();
uniformProjection = shaderList[2].getProjectLocation();
angulo += 0.01;
//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(-0.25f, -0.3f, -2.0f));
model = glm::scale(model, glm::vec3(0.3f, -0.3f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

Ventana derecha

```
//Ventana derecha
shaderList[2].useShader();
uniformModel = shaderList[2].getModelLocation();
uniformProjection = shaderList[2].getProjectLocation();
angulo += 0.01;
//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(0.25f, -0.3f, -2.0f));
model = glm::scale(model, glm::vec3(0.3f, -0.3f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

Puerta

```
//Puerta
shaderList[2].useShader();
uniformModel = shaderList[2].getModelLocation();
uniformProjection = shaderList[2].getProjectLocation();
angulo += 0.01;
//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(0.0, -0.85f, -2.0f));
model = glm::scale(model, glm::vec3(0.3f, -0.3f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

Tronco Izquierdo

```
//Tronco izquierdo
shaderList[5].useShader();
uniformModel = shaderList[5].getModelLocation();
uniformProjection = shaderList[5].getProjectLocation();
angulo += 0.01;
//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(-0.75, -0.9f, -2.0f));
model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

Tronco derecho

```
//Tronco derecho
shaderList[5].useShader();
uniformModel = shaderList[5].getModelLocation();
uniformProjection = shaderList[5].getProjectLocation();
angulo += 0.01;
//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(0.75, -0.9f, -2.0f));
model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

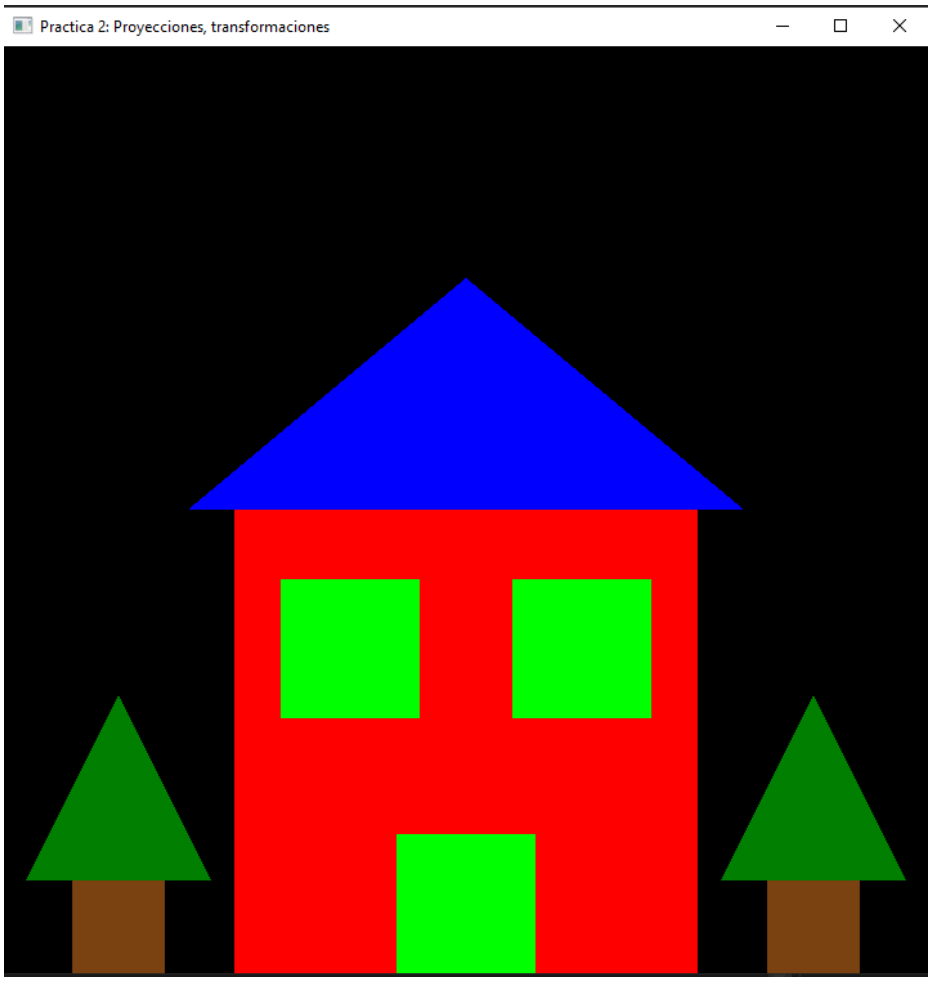
Pino Izquierdo

```
//Pino izquierdo
shaderList[6].useShader();
uniformModel = shaderList[6].getModelLocation();
uniformProjection = shaderList[6].getProjectLocation();
angulo += 0.01;
//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(-0.75f, -0.6f, -2.0f));
model = glm::scale(model, glm::vec3(0.4f, 0.40f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();
```

Pino derecho

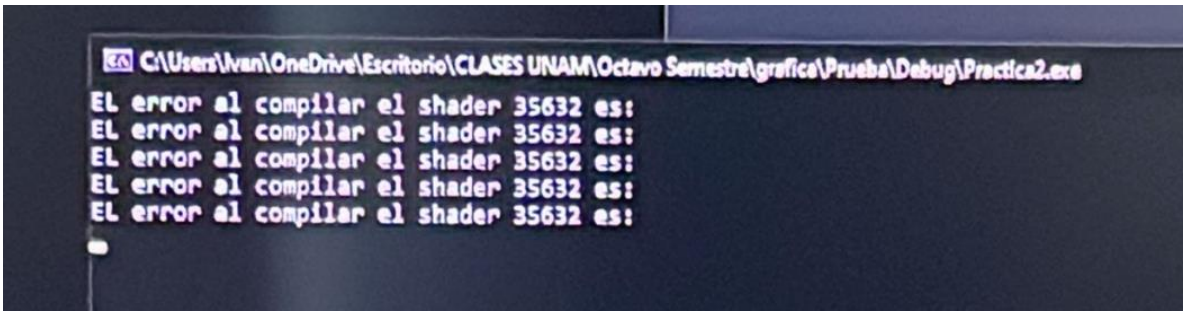
```
//Pino derecho
shaderList[6].useShader();
uniformModel = shaderList[6].getModelLocation();
uniformProjection = shaderList[6].getProjectLocation();
angulo += 0.01;
//Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para almacenar las transformaciones geométricas
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.75f, -0.6f, -2.0f));
model = glm::scale(model, glm::vec3(0.4f, 0.40f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();
```

Obteniendo como resultado la siguiente figura realizada con cubos y pirámides



2.- Problemas

Mi mayor desafío en esta practica fue crear los shaders de una manera correcta, no entendía como crearlos, y por ende me salía este error en pantalla. Al darme cuenta de este error, consulte con el profesor como se creaban los shaders, y con su explicación me quedo más claro. Ya que no estaba usando la sentencia de código correcta.



En la cuestión de observar las letras en pantalla no fue ningún problema, ni tampoco el uso del cubo o pirámide.

3.- Conclusión:

En esta practica se noto un poco más de nivel, y esto se debe a la creación de shaders, ya que no solo era escribir código en visual studio, si no que tendríamos que crear archivos y hacer que funcionaran dentro de nuestro código. Fuera de eso, creo que los ejercicios planteados fueron de una complejidad razonable, porque son ejercicios que ya habíamos realizado, pero ahora complementando las nuevas funciones que vamos aprendiendo en las clases de laboratorio. De esta práctica, me llevo como crear shaders que fue lo que mas se me complico, y por ende fue lo que más aprendí para poder resolver mis ejercicios solicitados.

Bibliografía

- *Consulta la tabla de colores HTML - cdmon*. (s/f). Cdmon.com. Recuperado el 24 de agosto de 2024, de https://www.cdmon.com/es/apps/tabla-colores