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: EMILIANO GARCIA OLVERA

Nº de Cuenta: 314256009

GRUPO DE LABORATORIO: 13

GRUPO DE TEORÍA: 06

SEMESTRE 2026-2

FECHA DE ENTREGA LÍMITE: 26/FEBRERO/2026

CALIFICACIÓN: _____

Ejercicios:

1.- Dibujar las iniciales de sus nombres en diagonal de abajo hacia arriba, cada letra de un color diferente

Desarrollo y codigo

El primer paso para realizar el ejercicio fue utilizar los vertices utilizados en la anterior practica ya que estos ya estaban definidos en el espacio en tres dimensiones. Por lo que simplemente realizamos las adaptaciones adecuadas, en este caso agregar la informacion del color que nos pide el ejercicio como 3 flotantes extras por cada vertice y al final hacer push de estos vertices dentro de nuestra meshColorList.

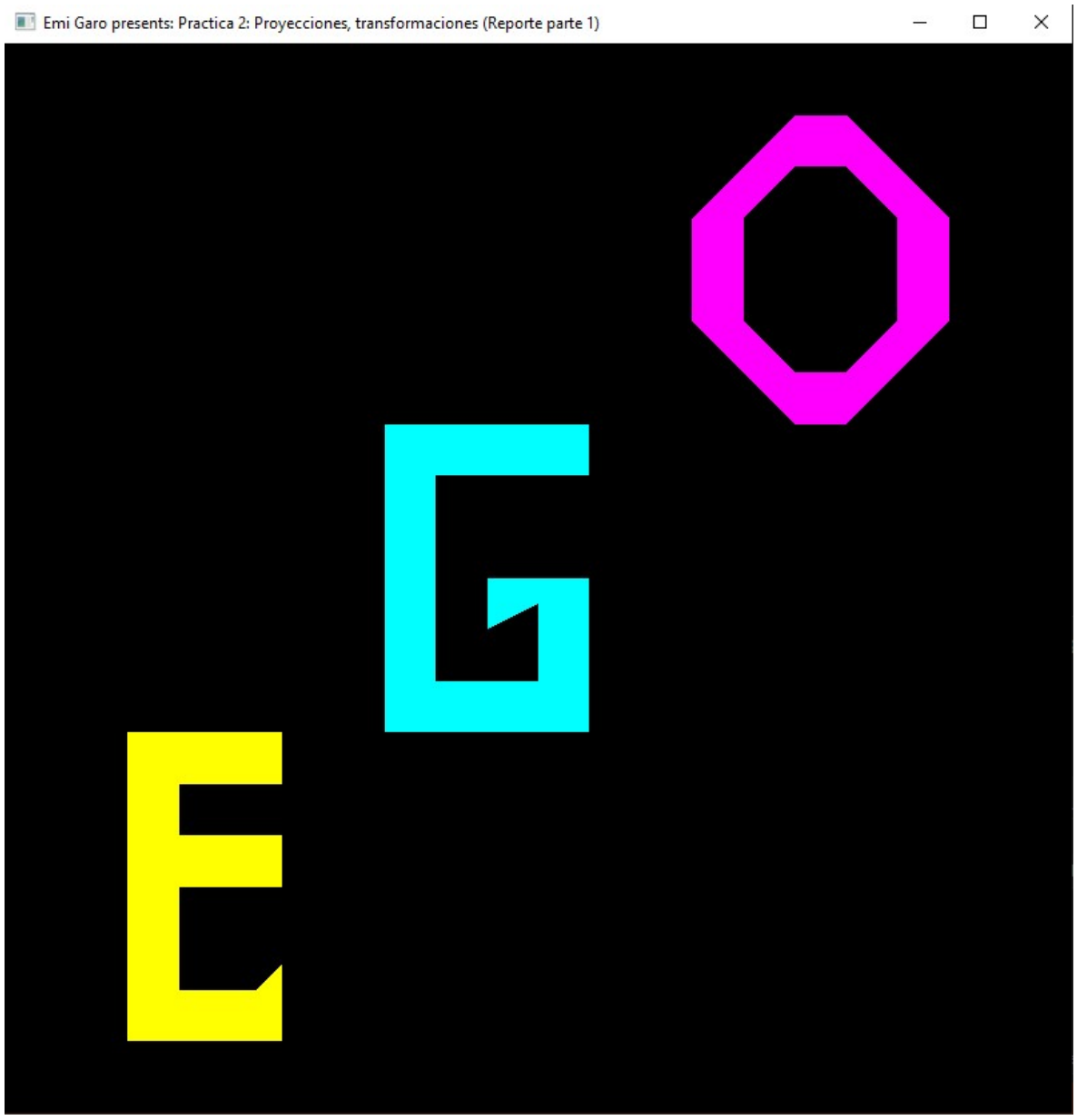
```
GLfloat vertices[] = {
	// E
	-0.9f, -0.3f - 0.6f, 0.0f, 1.0f, 1.0f, 0.0f,
	-0.8f, -0.3f - 0.6f, 0.0f, 1.0f, 1.0f, 0.0f,
	-0.9f,  0.3f - 0.6f, 0.0f, 1.0f, 1.0f, 0.0f,

	-0.8f, -0.3f - 0.6f, 0.0f, 1.0f, 1.0f, 0.0f,
	-0.8f,  0.3f - 0.6f, 0.0f, 1.0f, 1.0f, 0.0f,
	-0.9f,  0.3f - 0.6f, 0.0f, 1.0f, 1.0f, 0.0f,

	...
	// G
	-0.4f, -0.3f, 0.0f,0.0f, 1.0f, 1.0f,
	-0.3f, -0.3f, 0.0f,0.0f, 1.0f, 1.0f,
	-0.4f,  0.3f, 0.0f,0.0f, 1.0f, 1.0f,

	-0.3f, -0.3f, 0.0f,0.0f, 1.0f, 1.0f,
	-0.3f,  0.3f, 0.0f,0.0f, 1.0f, 1.0f,
	-0.4f,  0.3f, 0.0f,0.0f, 1.0f, 1.0f,

	...
	// O
	0.2f,  0.1f + 0.6f, 0.0,  1.0f, 0.0f, 1.0f,
	0.3f, -0.1f + 0.6f, 0.0,  1.0f, 0.0f, 1.0f,
	0.3f,  0.1f + 0.6f, 0.0,  1.0f, 0.0f, 1.0f,

	0.2f,  0.1f + 0.6f, 0.0,  1.0f, 0.0f, 1.0f,
	0.2f, -0.1f + 0.6f, 0.0,  1.0f, 0.0f, 1.0f,
	0.3f, -0.1f + 0.6f, 0.0,  1.0f, 0.0f, 1.0f,


};
MeshColor* triangulos_name = new MeshColor();
triangulos_name->CreateMeshColor(vertices, 6*102);
meshColorList.push_back(triangulos_name);
```

Finalmente renderizamos dicha meshColorList con los ultimos vertices enviados a la lista y listo. Nuestas iniciales ahora se muestran en la ventana, ahora a color.

```
	model = glm::mat4(1.0);
	model = glm::translate(model, glm::vec3(0.0f, 0.0f, -1.8f));
```

```
glUniformMatrix4fv(uniformModel, 1, GL_FALSE,
                   glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE,
                   glm::value_ptr(projection));
meshColorList[7]->RenderMeshColor();
```

Evidencia



2.- Generar el dibujo de la casa de la clase, pero en lugar de instanciar triangulos y cuadrados será instanciando piramides 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. Se debe de entregar los archivos de shader diferentes dentro de un zip adicional al main y documento escrito.

Desarrollo y Codigo

Para el desarrollo de este ejercicio primero creamos los shaders pedidos para poder colorear nuestras figuras. En este caso creamos un archivo de shader diferente por cada color. Por ejemplo este es el shader de color azul.

```
#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(0.0f, 0.0f, 1.0f, 1.0f);
}
```

Remarcamos como la parte importante en el codigo anterior es el vec4 que contiene un vector RGBA donde podemos especificar nuestros color. De esta forma creamos nuestros shaders y a continuacion declaramos la direccion de cada uno.

```
//Vertex Shader
static const char* vShader = "shaders/shader.vert";
static const char* fShader = "shaders/shader.frag";
//shaders nuevos se crearían acá
static const char* vShader_blue = "shaders/shader_blue.vert";
static const char* vShader_half_green = "shaders/shader_half_green.vert";
static const char* vShader_red = "shaders/shader_red.vert";
static const char* vShader_green = "shaders/shader_green.vert";
static const char* vShader_brown = "shaders/shader_brown.vert";
```

A continuacion en nuestra funcion CreateShaders cargamos cada uno de los shaders creados a nuestro programa.

```
void CreateShaders()
{

	Shader* shader = new Shader();
	shader->CreateFromFiles(vShader, fShader);
	shaderList.push_back(*shader);

	Shader* shader_blue = new Shader();
	shader_blue->CreateFromFiles(vShader_blue, fShader);
	shaderList.push_back(*shader_blue);

	Shader* shader_half_green = new Shader();
	shader_half_green->CreateFromFiles(vShader_half_green, fShader);
	shaderList.push_back(*shader_half_green);

	Shader* shader_red = new Shader();
	shader_red->CreateFromFiles(vShader_red, fShader);
	shaderList.push_back(*shader_red);

	Shader* shader_green = new Shader();
	shader_green->CreateFromFiles(vShader_green, fShader);
```

```
	shaderList.push_back(*shader_green);

	Shader* shader_brown = new Shader();
	shader_brown->CreateFromFiles(vShader_brown, fShader);
	shaderList.push_back(*shader_brown);
}
```

Enseguida de nuestro main corremos un par de funciones que cargan nuestras figuras a utilizar y los shaders que pintaran cada una de estas.

```
CrearPiramide(); //índice 0 en MeshList
CrearCubo();//índice 1 en MeshList
CrearTronco();//índice 2 en MeshList
CrearHojas();//índice 3 en MeshList
CrearVentana();//índice 4 en MeshList
CreateShaders();
```

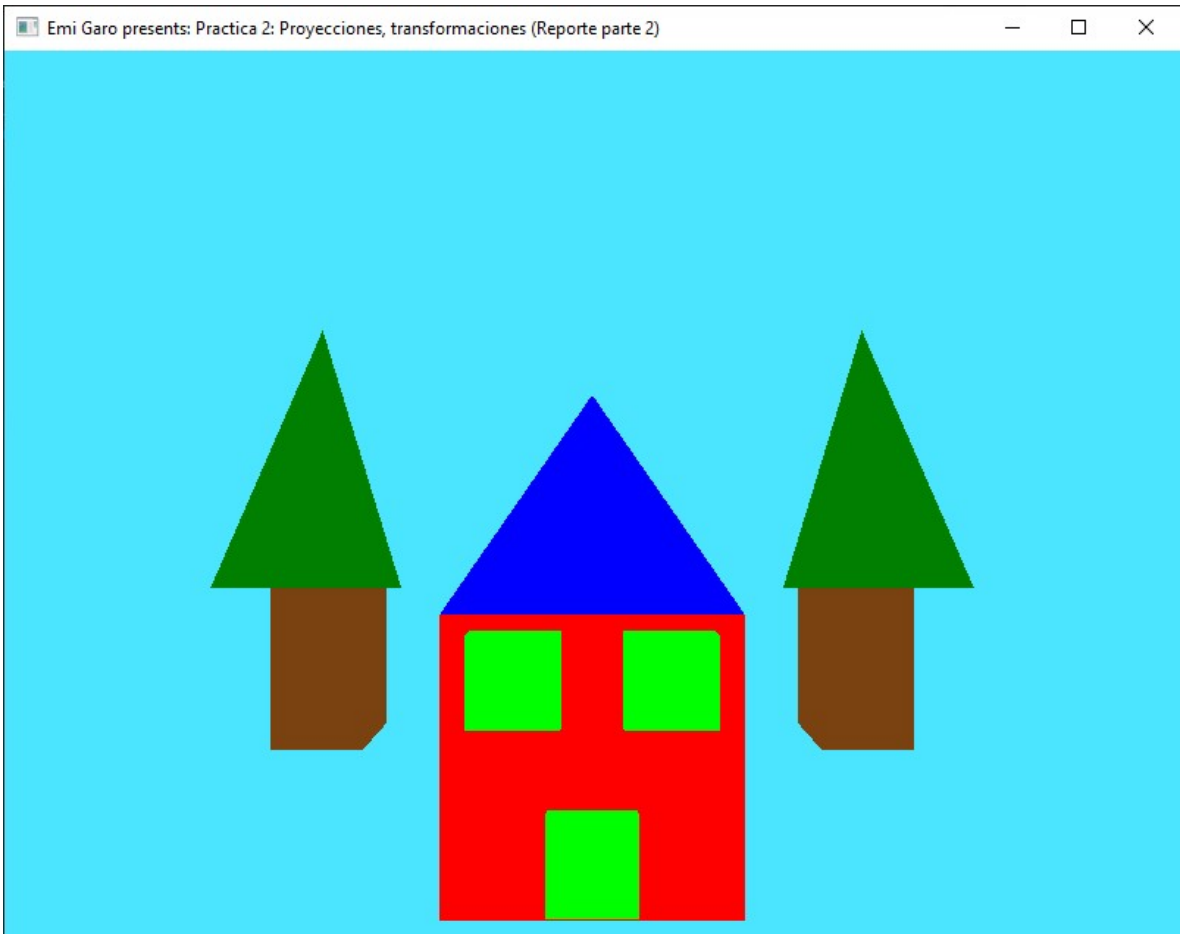
Finalmente pintamos, creamos, y posicionamos cada una de las figuras para poder construir nuestra casa en tres dimensiones.

```
		// Shader colors: 1.blue, 2.half-green, 3.red, 4.green, 5.brown

		// Shader Blue
		shaderList[1].useShader();
		uniformModel = shaderList[1].getModelLocation();
		uniformProjection = shaderList[1].getProjectLocation();
		glUniformMatrix4fv(uniformProjection, 1, GL_FALSE,
glm::value_ptr(projection));
		// Draw Piramid
		model = glm::mat4(1.0);
		model = glm::translate(model, glm::vec3(0.0f, 0.1f, -2.5f));
		glUniformMatrix4fv(uniformModel, 1, GL_FALSE,
glm::value_ptr(model));
		meshList[0]->RenderMesh();

		// Shader Red
		shaderList[3].useShader();
		uniformModel = shaderList[3].getModelLocation();
		uniformProjection = shaderList[3].getProjectLocation();
		glUniformMatrix4fv(uniformProjection, 1, GL_FALSE,
glm::value_ptr(projection));
		// Draw Cube
		model = glm::mat4(1.0);
		model = glm::translate(model, glm::vec3(0.0f, -0.9f, -3.0f));
		glUniformMatrix4fv(uniformModel, 1, GL_FALSE,
glm::value_ptr(model));
		meshList[1]->RenderMesh();

		// Shader Green
		shaderList[4].useShader();
		uniformModel = shaderList[4].getModelLocation();
		uniformProjection = shaderList[4].getProjectLocation();
		glUniformMatrix4fv(uniformProjection, 1, GL_FALSE,
glm::value_ptr(projection));
		// Draw Ventana
		model = glm::mat4(1.0);
		model = glm::translate(model, glm::vec3(-0.25f, -0.6f, -2.5f));
```

```
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE,
glm::value_ptr(model));
            meshList[4]->RenderMesh();

            // Shader Green
            shaderList[4].useShader();
            uniformModel = shaderList[4].getModelLocation();
            uniformProjection = shaderList[4].getProjectLocation();
            glUniformMatrix4fv(uniformProjection, 1, GL_FALSE,
glm::value_ptr(projection));
            // Draw Ventana
            model = glm::mat4(1.0);
            model = glm::translate(model, glm::vec3(0.25f, -0.6f, -2.5f));
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE,
glm::value_ptr(model));
            meshList[4]->RenderMesh();

            // Shader Green
            shaderList[4].useShader();
            uniformModel = shaderList[4].getModelLocation();
            uniformProjection = shaderList[4].getProjectLocation();
            glUniformMatrix4fv(uniformProjection, 1, GL_FALSE,
glm::value_ptr(projection));
            // Draw Ventana
            model = glm::mat4(1.0);
            model = glm::translate(model, glm::vec3(0.0f, -1.19f, -2.5f));
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE,
glm::value_ptr(model));
            meshList[4]->RenderMesh();

            // Shader Brown
            shaderList[5].useShader();
            uniformModel = shaderList[5].getModelLocation();
            uniformProjection = shaderList[5].getProjectLocation();
            glUniformMatrix4fv(uniformProjection, 1, GL_FALSE,
glm::value_ptr(projection));
            // Draw Tronco
            model = glm::mat4(1.0);
            model = glm::translate(model, glm::vec3(-1.5f, -0.9f, -4.4f));
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE,
glm::value_ptr(model));
            meshList[2]->RenderMesh();

            // Shader Brown
            shaderList[5].useShader();
            uniformModel = shaderList[5].getModelLocation();
            uniformProjection = shaderList[5].getProjectLocation();
            glUniformMatrix4fv(uniformProjection, 1, GL_FALSE,
glm::value_ptr(projection));
            // Draw Tronco
            model = glm::mat4(1.0);
            model = glm::translate(model, glm::vec3(1.5f, -0.9f, -4.4f));
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE,
glm::value_ptr(model));
            meshList[2]->RenderMesh();

            // Shader Half Green
            shaderList[2].useShader();
```

```
            uniformModel = shaderList[2].getModelLocation();
            uniformProjection = shaderList[2].getProjectLocation();
            glUniformMatrix4fv(uniformProjection, 1, GL_FALSE,
glm::value_ptr(projection));
            // Draw Hojas
            model = glm::mat4(1.0);
            model = glm::translate(model, glm::vec3(-1.5f, -0.f, -4.0f));
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE,
glm::value_ptr(model));
            meshList[3]->RenderMesh();

            // Shader Half Green
            shaderList[2].useShader();
            uniformModel = shaderList[2].getModelLocation();
            uniformProjection = shaderList[2].getProjectLocation();
            glUniformMatrix4fv(uniformProjection, 1, GL_FALSE,
glm::value_ptr(projection));
            // Draw Hojas
            model = glm::mat4(1.0);
            model = glm::translate(model, glm::vec3(1.5f, -0.f, -4.0f));
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE,
glm::value_ptr(model));
            meshList[3]->RenderMesh();
```

Evidencia



Conclusión

Esta semana avanzamos en nuestro aprendizaje de OpenGL logrando avanzar en el codigo creado anteriormente para poder dibujar nuestro nombre, ya que anteriormente solo utilizamos la informacion de la pocision de nuestros vectores sin preocuparnos por colores. Esta vez, mediante triangulos y shaders trabajamos de manera que esta vez cada uno de los triangulos tenia un color diferente. Esto sirve a grandes rasgos como una introduccion a shaders y fue utilizado en nuestro ejercicio en clase para poder dibujar con triangulos de colores una pequeña casa.

A continuacion en nuestros ejercicios de la practica utilizamos la misma forma de trabajo para recrear nuestro nombre esta vez pintando cada una de las letras, lo cual si entendimos bien hasta este punto fue trivial realizarlo pues la parte dificil (ubicar cada vertice de cada triangulo) ya estaba resuelta para nosotros.

Finalmente logramos llegar a el uso de las tres dimensiones, pintando la casa anterior esta vez con piramides y cubos. Esto fue mas dificil pues teniamos que ocupar todo lo anterior y realmente entender que estabamos trabajando en un espacio y no solo un plano, ademas de crear shaders de colores para cada figura en vez de introducir la informacion en cada triangulo por lo que represento un reto en nuestro conocimiento que cumplimos con éxito.

Referencias:

- Apuntes de la clase de teoria y laboratorio de “Computacion Grafica e Interaccion Humano Computadora”