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: Hernández Vázquez Daniela

Nº de Cuenta: 318092867

GRUPO DE LABORATORIO: 01

GRUPO DE TEORÍA: 02

SEMESTRE 2024-2

FECHA DE ENTREGA LÍMITE: 24 de febrero de 2024

CALIFICACIÓN: _____

REPORTE DE PRÁCTICA:

Actividad 1.

Como primera actividad se dibujaron las iniciales de mi nombre (DHV) cada letra de un color diferente, para esto utilizamos como base los ejercicios realizados en la clase y el diagrama de las letras elaborado para la practica 1.

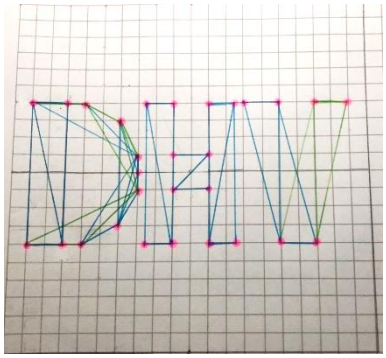

Figura 2.1 Diagrama de letras

Modificamos la función *CrearLetrasyFiguras()* y dentro de la sección de *vertices_letras[]* se declaran todos los datos correspondientes. Tomamos en cuenta que ahora cada primitiva de triangulo está firmado por 18 datos, 9 datos son utilizados para generar los vértices de triángulo, y otros 9 son utilizados para definir el color RGB del mismo.

Con estos triángulos procedemos a dibujar nuestras iniciales en la pantalla, usando para cada letra el número de vértices y triángulos que ya habíamos definido:

- Letra D: 27 vértices (9 triángulos) color naranja RGB (1.0,0.201,0.110)
- Letra H: 18 vértices (6 triángulos) color amarillo RGB (0.816,1.0,0.0)
- Letra V: 12 vértices (4 triángulos) color verde menta RGB (0.231,1.0,0.494)

Declaramos de igual modo que con las figuras de la practica anterior debemos de declararlo como un nuevo MeshColor con un índice de acuerdo al número de datos utilizados, en este caso 342 (Figura 2.2).

En el main principal no se modifica la proyección ortogonal y si se inicializa la matriz identidad con *glm::mat4 model(1.0);* una sola vez y posteriormente dentro del ciclo *while* es llamada para ser modificada. Las letras son instanciadas llamando al MeshColor[0]. (Figura 2.3).

```
void CrearLetrasyFiguras()
{
    GLfloat vertices_letras[] = {
        //X          Y          Z          R       G       B

        //Letra D   Rojo
        -0.9f,      0.4f,      0.5f,      1.0f,   0.201f, 0.110f,
        -0.9f,     -0.4f,      0.5f,      1.0f,   0.201f, 0.110f,
        -0.7f,     -0.4f,      0.5f,      1.0f,   0.201f, 0.110f,
        -0.9f,      0.4f,      0.5f,      1.0f,   0.201f, 0.110f,
        -0.7f,      0.4f,      0.5f,      1.0f,   0.201f, 0.110f,
        -0.7f,     -0.4f,      0.5f,      1.0f,   0.201f, 0.110f,

        -0.30f,     0.10f,     0.5f,      1.0f,   0.201f, 0.110f,
        -0.30f,    -0.10f,     0.5f,      1.0f,   0.201f, 0.110f,
        -0.40f,    -0.30f,     0.5f,      1.0f,   0.201f, 0.110f,
        -0.30f,     0.10f,     0.5f,      1.0f,   0.201f, 0.110f,
        -0.60f,    -0.40f,     0.5f,      1.0f,   0.201f, 0.110f,
        -0.40f,    -0.30f,     0.5f,      1.0f,   0.201f, 0.110f,
        -0.30f,    -0.10f,     0.5f,      1.0f,   0.201f, 0.110f,
        -0.90f,    -0.40f,     0.5f,      1.0f,   0.201f, 0.110f,
        -0.60f,    -0.40f,     0.5f,      1.0f,   0.201f, 0.110f,

        -0.30f,    0.00f,      0.5f,      1.0f,   0.201f, 0.110f,
        -0.40f,    0.30f,      0.5f,      1.0f,   0.201f, 0.110f,
        -0.40f,   -0.30f,      0.5f,      1.0f,   0.201f, 0.110f,

        //Letra H   Amarillo
        0.25f,      0.4f,      0.5f,      0.816f, 1.0f,   0.0f,
        0.25f,     -0.4f,      0.5f,      0.816f, 1.0f,   0.0f,
        0.10f,     -0.4f,      0.5f,      0.816f, 1.0f,   0.0f,
        0.25f,      0.4f,      0.5f,      0.816f, 1.0f,   0.0f,
        0.10f,      0.4f,      0.5f,      0.816f, 1.0f,   0.0f,
        0.10f,     -0.4f,      0.5f,      0.816f, 1.0f,   0.0f,

        0.10f,      0.10f,     0.5f,      0.816f, 1.0f,   0.0f,
        0.10f,     -0.10f,     0.5f,      0.816f, 1.0f,   0.0f,
       -0.10f,     -0.10f,     0.5f,      0.816f, 1.0f,   0.0f,
        0.10f,      0.10f,     0.5f,      0.816f, 1.0f,   0.0f,
       -0.10f,      0.10f,     0.5f,      0.816f, 1.0f,   0.0f,
       -0.10f,     -0.10f,     0.5f,      0.816f, 1.0f,   0.0f,

       -0.25f,      0.4f,      0.5f,      0.816f, 1.0f,   0.0f,
       -0.25f,     -0.4f,      0.5f,      0.816f, 1.0f,   0.0f,
       -0.10f,     -0.4f,      0.5f,      0.816f, 1.0f,   0.0f,
       -0.25f,      0.4f,      0.5f,      0.816f, 1.0f,   0.0f,
       -0.10f,      0.4f,      0.5f,      0.816f, 1.0f,   0.0f,
       -0.10f,     -0.4f,      0.5f,      0.816f, 1.0f,   0.0f,

      //Letra V    Verde menta
        0.30f,      0.4f,      0.5f,      0.231f, 1.0f,   0.494f,
        0.50f,      0.4f,      0.5f,      0.231f, 1.0f,   0.494f,
        0.50f,     -0.4f,      0.5f,      0.231f, 1.0f,   0.494f,
        0.70f,     -0.4f,      0.5f,      0.231f, 1.0f,   0.494f,
        0.50f,      0.4f,      0.5f,      0.231f, 1.0f,   0.494f,
        0.50f,     -0.4f,      0.5f,      0.231f, 1.0f,   0.494f,

        0.70f,     -0.4f,      0.5f,      0.231f, 1.0f,   0.494f,
        0.70f,      0.4f,      0.5f,      0.231f, 1.0f,   0.494f,
        0.50f,     -0.4f,      0.5f,      0.231f, 1.0f,   0.494f,
        0.70f,     -0.4f,      0.5f,      0.231f, 1.0f,   0.494f,
        0.70f,      0.4f,      0.5f,      0.231f, 1.0f,   0.494f,
        0.90f,      0.4f,      0.5f,      0.231f, 1.0f,   0.494f,

    };
    MeshColor* letras = new MeshColor();
    letras->CreateMeshColor(vertices_letras, 342);
    meshColorList.push_back(letras);
```

Figura 2.2 Vértices de letras

```
//Para las letras hay que usar el segundo set de shaders con índice 1 en ShaderList
shaderList[1].useShader();
uniformModel = shaderList[1].getModelLocation();
uniformProjection = shaderList[1].getProjectLocation();

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

Figura 2.3 Instanciar figuras

Figura 2.4 Ejecución del programa

Otra forma en la que se pudieron haber instanciado las letras era separándolas creando un GL float con los valores de cada uno, pero con esto se tendría que instanciar cada una por separado en lugar de llamar un solo color mesh.

Actividad 2.

Como segunda actividad se generó el dibujo de la casa de la clase, pero en lugar de instanciar las figuras de los triángulos y cuadrados de color creados, ahora la figura estaría formada por pirámides y cubos. Para esto se requiere crear *shaders* diferentes de los colores: rojo, verde, azul, café, verde y verde oscuro en lugar de usar el *shader* con el color *clamp*.

Tomamos los *shader* ya creados de la práctica 2 y con base a ellos creamos los nuevos *shaders* simplemente sustituyendo la salida de color *clamp* por el color RGB que le corresponde con la estructura de la figura 2.5. Agregándolos al proyecto dentro de la carpeta de *shaders*.

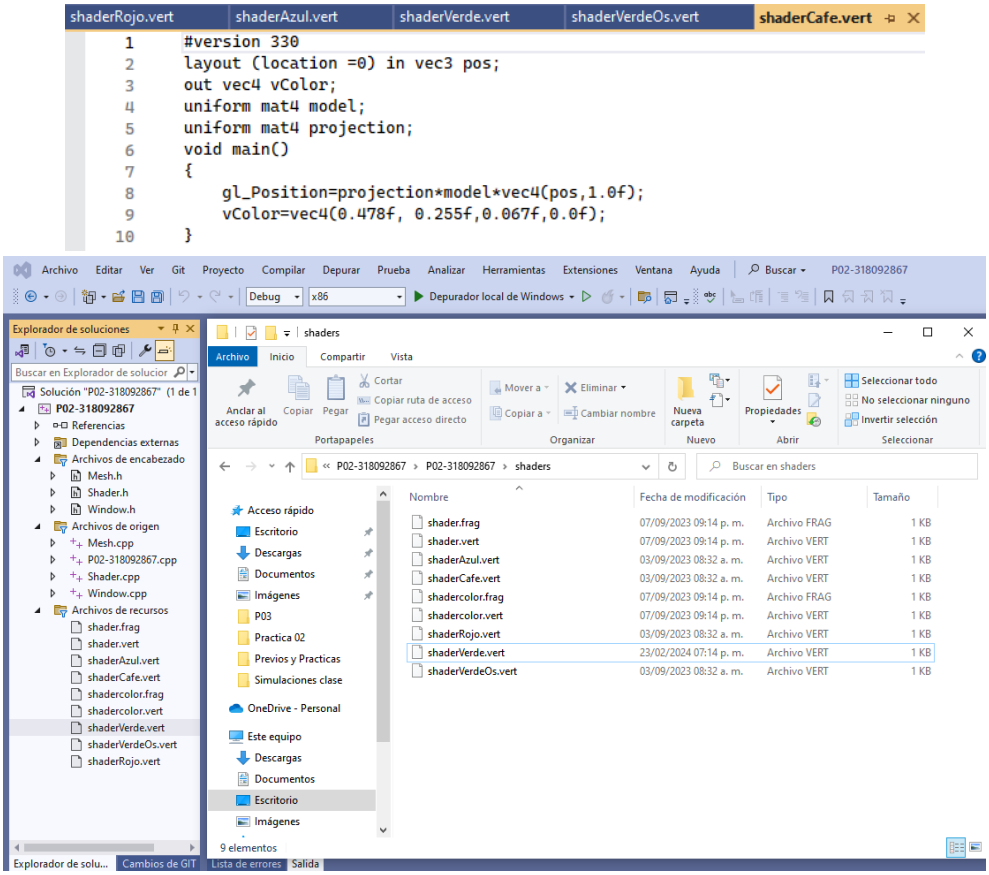

Figura 2.5 Shaders.vert de colores

En el código principal declaramos las variables que utilizaremos para trabajar con los shaders. Asi mismo dentro de la función de *CreateShaders()* se creen los nuevos shaders haciendo referencia a las variables del inicio, debemos de tomar en cuenta nuevamente que el orden en el que están siendo creados influye al momento de instanciarlos.

```
16    //Dimensiones de la ventana
17    const float toRadians = 3.14159265f/180.0; //grados a radianes
18    Window mainWindow;
19    std::vector<Mesh*> meshList;
20    std::vector<MeshColor*> meshColorList;
21    std::vector<Shader>shaderList;
22    //Vertex Shader
23    static const char* vShader = "shaders/shader.vert";
24    static const char* fShader = "shaders/shader.frag";
25    static const char* vShaderColor = "shaders/shadercolor.vert";
26    static const char* fShaderColor = "shaders/shadercolor.frag";
27    //shaders nuevos se crearían acá
28    static const char* vShaderAzul = "shaders/shaderAzul.vert";
29    static const char* vShaderCafe = "shaders/shaderCafe.vert";
30    static const char* vShaderVerde = "shaders/shaderVerde.vert";
31    static const char* vShaderVerdeOs = "shaders/shaderVerdeOs.vert";
32    static const char* vShaderRojo = "shaders/shaderRojo.vert";
```

```
264  void CreateShaders()// Se crean los shaders
265  {
266     Shader* shader1 = new Shader(); //shader para usar índices: objetos: cubo y  pirámide
267     shader1->CreateFromFiles(vShader, fShader);
268     shaderList.push_back(*shader1);
269
270     Shader* shader2 = new Shader();//shader para usar color como parte del VAO: letras
271     shader2->CreateFromFiles(vShaderColor, fShaderColor);
272     shaderList.push_back(*shader2);
273
274     Shader* shader3 = new Shader(); //shader para usar índices: objetos: cubo y  pirámide
275     shader3->CreateFromFiles(vShaderRojo, fShader);
276     shaderList.push_back(*shader3);
277
278     Shader* shader4 = new Shader(); //shader para usar índices: objetos: cubo y  pirámide
279     shader4->CreateFromFiles(vShaderAzul, fShader);
280     shaderList.push_back(*shader4);
281
282     Shader* shader5 = new Shader(); //shader para usar índices: objetos: cubo y  pirámide
283     shader5->CreateFromFiles(vShaderVerdeOs, fShader);
284     shaderList.push_back(*shader5);
285
286     Shader* shader6 = new Shader(); //shader para usar índices: objetos: cubo y  pirámide
287     shader6->CreateFromFiles(vShaderVerde, fShader);
288     shaderList.push_back(*shader6);
289
290     Shader* shader7 = new Shader(); //shader para usar índices: objetos: cubo y  pirámide
291     shader7->CreateFromFiles(vShaderCafe, fShader);
292     shaderList.push_back(*shader7);
293  }
```

Figura 2.6 Creación de shaders

Finalmente, dentro del main se instancian los objetos con su respectivo shader, debemos recordar que la matriz debe de inicializarse para cada modelo (tener cuidado con el número de índice de los shader), asi mismo se debe de tomar en cuenta para los mismos el escalamiento, traslaciones utilizadas y el tamaño de la ventana de proyección.

Figura 2.7 Ejecución del programa

Problemas:

En la primera actividad se corrigieron las anotaciones que se habían hecho con respecto a la inicialización de las matrices, se declararon los vértices de la practica 1 por lo que solo se agregaron los colores RGB para las letras utilizando la siguiente página para obtener los códigos de color: https://rgbcolorpicker.com/0-1

En cuanto a la segunda actividad lo más complejo fue el instanciar nuevamente las figuras para la construcción de la casa ya que se debía de tener cuidado con los índices, los escalamientos y las transiciones utilizadas.

Conclusión:

Los ejercicios resultan cada vez más complejos, pasamos de trabajar en 2D a 3D aunque siguiendo la guía de la práctica esta resultó más sencilla. Considero que lo ejercicios tienen un nivel adecuado. La explicación de clase me pareció suficiente para poder tener los elementos para modificar el programa, aunque por aluna razón en la clase no habia funcionado el archivo de la practica a pesar de ya estar modificados los parámetros. A pesar de ello las actividades se realizaron exitosamente en mi computadora.

Referencias:

Kessenich, J. M., Sellers, G., & Shreiner, D. (2016). *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V*. OpenGL.

Méndez Servín, M. (2022). *NOTAS PARA EL CURSO DE GRAFICACIÓN POR COMPUTADORA* [Libro electrónico]. https://prometeo.matem.unam.mx/recursos/VariosNiveles/iCartesiLibri/recursos/Notas_Graficacion_por_Computadora/index.html

RGB 0-1 Color Picker. (s. f.). https://rgbcolorpicker.com/0-1

Sellers, G., Wright, R. S., & Haemel, N. (2015). *OpenGL superbible: Comprehensive Tutorial and Reference*. Addison-Wesley Professional.