



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 N° 02

NOMBRE COMPLETO: García Soto Jean Carlo

N° de Cuenta: 319226304

GRUPO DE LABORATORIO: 03

GRUPO DE TEORÍA: 04

SEMESTRE 2025-2

FECHA DE ENTREGA LÍMITE: 22/02/2025

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.

Ejercicio 1:

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

Primero empezamos creando y pasando los vértices de las letras creadas en la anterior práctica (JCG), el cual definiremos un color cualquiera, en este caso, Verde Oscuro, Morado y Gris Oscuro. Dado que cada letra va a tener un color diferente es importante declararla por separado, ya que sino no tendrá una buena salida nuestras figuras. Y al final de cada declaración de vértices los metemos en nuestra lista **meshColorList**.

```
96 void CrearLetrasyFiguras()
97 {
98     //Letra J
99     GLfloat vertices_letra_J[] = {
100         //triangulo 1
101         //X      Y      Z      R      G      B
102         -0.8f,  0.5f,  0.5f,  0.0f,  0.3f,  0.0f,
103         -0.8f,  0.37f, 0.5f,  0.0f,  0.3f,  0.0f,
104         -0.4f,  0.5f,  0.5f,  0.0f,  0.3f,  0.0f,
105
106         //triangulo 2
107         //X      Y      Z      R      G      B
108         -0.8f,  0.37f, 0.5f,  0.0f,  0.3f,  0.0f,
109         -0.4f,  0.5f,  0.5f,  0.0f,  0.3f,  0.0f,
110         -0.4f,  0.37f, 0.5f,  0.0f,  0.3f,  0.0f,
111
112         //triangulo 3
113         //X      Y      Z      R      G      B
114         -0.67f, 0.37f, 0.5f,  0.0f,  0.3f,  0.0f,
115         -0.53f, 0.37f, 0.5f,  0.0f,  0.3f,  0.0f,
116         -0.67f, 0.0f,  0.5f,  0.0f,  0.3f,  0.0f,
117
118         //triangulo 4
119         //X      Y      Z      R      G      B
120         -0.53f, 0.37f, 0.5f,  0.0f,  0.3f,  0.0f,
121         -0.53f, 0.0f,  0.5f,  0.0f,  0.3f,  0.0f,
122         -0.67f, -0.0f, 0.5f,  0.0f,  0.3f,  0.0f,
123
124         //triangulo 5
125         //X      Y      Z      R      G      B
126         -0.67f, 0.1f,  0.5f,  0.0f,  0.3f,  0.0f,
127         -0.67f, 0.0f,  0.5f,  0.0f,  0.3f,  0.0f,
128         -0.85f, 0.0f,  0.5f,  0.0f,  0.3f,  0.0f,
129
130         //triangulo 6
131         //X      Y      Z      R      G      B
132         -0.67f, 0.1f,  0.5f,  0.0f,  0.3f,  0.0f,
133         -0.85f, 0.1f,  0.5f,  0.0f,  0.3f,  0.0f,
134         -0.85f, 0.0f,  0.5f,  0.0f,  0.3f,  0.0f,
135
136         //triangulo 7
137         //X      Y      Z      R      G      B
138         -0.75f, 0.1f,  0.5f,  0.0f,  0.3f,  0.0f,
139         -0.85f, 0.1f,  0.5f,  0.0f,  0.3f,  0.0f,
140         -0.85f, 0.15f, 0.5f,  0.0f,  0.3f,  0.0f,
141
142         //triangulo 8
143         //X      Y      Z      R      G      B
144         -0.75f, 0.1f,  0.5f,  0.0f,  0.3f,  0.0f,
145         -0.85f, 0.15f, 0.5f,  0.0f,  0.3f,  0.0f,
146         -0.75f, 0.15f, 0.5f,  0.0f,  0.3f,  0.0f,
147
148     };
149     MeshColor* letra_J = new MeshColor();
150     letra_J->CreateMeshColor(vertices_letra_J, 144); // 8*3*6=144
151     meshColorList.push_back(letra_J);
```

Asignando vértices y color a la letra C

```

153 //Letra C
154 GLfloat vertices_letra_C[] = {
155     //triangulo 1
156     //X      Y      Z      R      G      B
157     0.1f,    0.33f,  0.5f,    0.502f, 0.0f, 0.502f,
158     0.0f,    0.33f,  0.5f,    0.502f, 0.0f, 0.502f,
159     0.1f,    0.5f,   0.5f,    0.502f, 0.0f, 0.502f,
160
161     //triangulo 2
162     //X      Y      Z      R      G      B
163     0.0f,    0.33f,  0.5f,    0.502f, 0.0f, 0.502f,
164     0.1f,    0.5f,   0.5f,    0.502f, 0.0f, 0.502f,
165     0.0f,    0.5f,   0.5f,    0.502f, 0.0f, 0.502f,
166
167     //triangulo 3
168     //X      Y      Z      R      G      B
169     0.0f,    0.5f,   0.5f,    0.502f, 0.0f, 0.502f,
170     0.0f,    0.4f,   0.5f,    0.502f, 0.0f, 0.502f,
171     -0.2f,   0.4f,   0.5f,    0.502f, 0.0f, 0.502f,
172
173     //triangulo 4
174     //X      Y      Z      R      G      B
175     0.0f,    0.5f,   0.5f,    0.502f, 0.0f, 0.502f,
176     -0.2f,   0.4f,   0.5f,    0.502f, 0.0f, 0.502f,
177     -0.2f,   0.5f,   0.5f,    0.502f, 0.0f, 0.502f,
178
179     //triangulo 5
180     //X      Y      Z      R      G      B
181     -0.2f,   0.4f,   0.5f,    0.502f, 0.0f, 0.502f,
182     -0.09f,  0.4f,   0.5f,    0.502f, 0.0f, 0.502f,
183     -0.2f,   0.0f,   0.5f,    0.502f, 0.0f, 0.502f,
184
185     //triangulo 6
186     //X      Y      Z      R      G      B
187     -0.09f,  0.4f,   0.5f,    0.502f, 0.0f, 0.502f,
188     -0.2f,   0.0f,   0.5f,    0.502f, 0.0f, 0.502f,
189     -0.09f,  0.0f,   0.5f,    0.502f, 0.0f, 0.502f,
190
191     //triangulo 7
192     //X      Y      Z      R      G      B
193     -0.09f,  0.0f,   0.5f,    0.502f, 0.0f, 0.502f,
194     -0.09f,  0.1f,   0.5f,    0.502f, 0.0f, 0.502f,
195     0.1f,    0.1f,   0.5f,    0.502f, 0.0f, 0.502f,
196
197     //triangulo 8
198     //X      Y      Z      R      G      B
199     -0.09f,  0.0f,   0.5f,    0.502f, 0.0f, 0.502f,
200     0.1f,    0.1f,   0.5f,    0.502f, 0.0f, 0.502f,
201     0.1f,    0.0f,   0.5f,    0.502f, 0.0f, 0.502f,
202
203     //triangulo 9
204     //X      Y      Z      R      G      B
205     0.1f,    0.1f,   0.5f,    0.502f, 0.0f, 0.502f,
206     0.0f,    0.1f,   0.5f,    0.502f, 0.0f, 0.502f,
207     0.1f,    0.17f,  0.5f,    0.502f, 0.0f, 0.502f,
208
209     //triangulo 10
210     //X      Y      Z      R      G      B
211     0.0f,    0.1f,   0.5f,    0.502f, 0.0f, 0.502f,
212     0.1f,    0.17f,  0.5f,    0.502f, 0.0f, 0.502f,
213     0.0f,    0.17f,  0.5f,    0.502f, 0.0f, 0.502f
214
215 };
216 MeshColor* letra_C = new MeshColor();
217 letra_C->CreateMeshColor(vertices_letra_C, 180); // 10*3*6=180
218 meshColorList.push_back(letra_C);
219

```

Asignando vértices y color a la letra G

```
//Letra G
GLfloat vertices_letra_G[] = {

    //triangulo 1
    //X      Y      Z      R      G      B
    0.3f,    0.5f,    0.5f,    0.2f,    0.2f,    0.2f,
    0.7f,    0.5f,    0.5f,    0.2f,    0.2f,    0.2f,
    0.7f,    0.4f,    0.5f,    0.2f,    0.2f,    0.2f,

    //triangulo 2
    //X      Y      Z      R      G      B
    0.7f,    0.4f,    0.5f,    0.2f,    0.2f,    0.2f,
    0.3f,    0.5f,    0.5f,    0.2f,    0.2f,    0.2f,
    0.3f,    0.4f,    0.5f,    0.2f,    0.2f,    0.2f,

    //triangulo 3
    //X      Y      Z      R      G      B
    0.3f,    0.4f,    0.5f,    0.2f,    0.2f,    0.2f,
    0.4f,    0.4f,    0.5f,    0.2f,    0.2f,    0.2f,
    0.3f,    0.0f,    0.5f,    0.2f,    0.2f,    0.2f,

    //triangulo 4
    //X      Y      Z      R      G      B
    0.4f,    0.4f,    0.5f,    0.2f,    0.2f,    0.2f,
    0.3f,    0.0f,    0.5f,    0.2f,    0.2f,    0.2f,
    0.4f,    0.0f,    0.5f,    0.2f,    0.2f,    0.2f,

    //triangulo 5
    //X      Y      Z      R      G      B
    0.4f,    0.0f,    0.5f,    0.2f,    0.2f,    0.2f,
    0.4f,    0.1f,    0.5f,    0.2f,    0.2f,    0.2f,
    0.7f,    0.1f,    0.5f,    0.2f,    0.2f,    0.2f,

    //triangulo 6
    //X      Y      Z      R      G      B
    0.4f,    0.0f,    0.5f,    0.2f,    0.2f,    0.2f,
    0.7f,    0.0f,    0.5f,    0.2f,    0.2f,    0.2f,
    0.7f,    0.1f,    0.5f,    0.2f,    0.2f,    0.2f,

    //triangulo 7
    //X      Y      Z      R      G      B
    0.7f,    0.1f,    0.5f,    0.2f,    0.2f,    0.2f,
    0.6f,    0.1f,    0.5f,    0.2f,    0.2f,    0.2f,
    0.7f,    0.25f,    0.5f,    0.2f,    0.2f,    0.2f,

    //triangulo 8
    //X      Y      Z      R      G      B
    0.6f,    0.1f,    0.5f,    0.2f,    0.2f,    0.2f,
    0.6f,    0.25f,    0.5f,    0.2f,    0.2f,    0.2f,
    0.7f,    0.25f,    0.5f,    0.2f,    0.2f,    0.2f,

    //triangulo 9
    //X      Y      Z      R      G      B
    0.6f,    0.25f,    0.5f,    0.2f,    0.2f,    0.2f,
    0.6f,    0.19f,    0.5f,    0.2f,    0.2f,    0.2f,
    0.5f,    0.19f,    0.5f,    0.2f,    0.2f,    0.2f,

    //triangulo 10
    //X      Y      Z      R      G      B
    0.6f,    0.25f,    0.5f,    0.2f,    0.2f,    0.2f,
    0.5f,    0.25f,    0.5f,    0.2f,    0.2f,    0.2f,
    0.5f,    0.19f,    0.5f,    0.2f,    0.2f,    0.2f
};

MeshColor* letra_G = new MeshColor();
letra_G->CreateMeshColor(vertices_letra_G, 180); // 10*3*6=180
meshColorList.push_back(letra_G);
```

Dentro del **main()**, procederemos a renderizar nuestras letras, como estas fueron creadas en un orden, a la hora de llamar **meshColorList[]**, debemos de respetar el orden o índice para que podamos visualizar la letra junto el color escogido desde un inicio.

Ya solo le damos el la posición en las que queremos que aparezcan, y dado que ya estaban dimensionadas a la hora de crear las coordenadas de cada letra, le damos el mismo tamaño en las escalas tanto en x como en y, en z no es necesario dado que es en 2D.

```
421 //Renderizar letra J
422 model = glm::mat4(1.0);
423 model = glm::translate(model, glm::vec3(-2.0f, 0.0f, -4.0f));
424 model = glm::scale(model, glm::vec3(7.0f, 7.0f, 1.0f));
425 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
426 glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
427 meshColorList[0]->RenderMeshColor();
428
429 //Renderizar letra C
430 model = glm::mat4(1.0);
431 model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
432 model = glm::scale(model, glm::vec3(7.0f, 7.0f, 1.0f));
433 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
434 glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
435 meshColorList[1]->RenderMeshColor();
436
437 //Renderizar letra G
438 model = glm::mat4(1.0);
439 model = glm::translate(model, glm::vec3(2.0f, 0.0f, -4.0f));
440 model = glm::scale(model, glm::vec3(7.0f, 7.0f, 1.0f));
441 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
442 glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
443 meshColorList[2]->RenderMeshColor();
444
445
```

Y esta fue la salida, el cual fue la correcta, dado que solo queríamos agregarle un color diferente a cada letra.



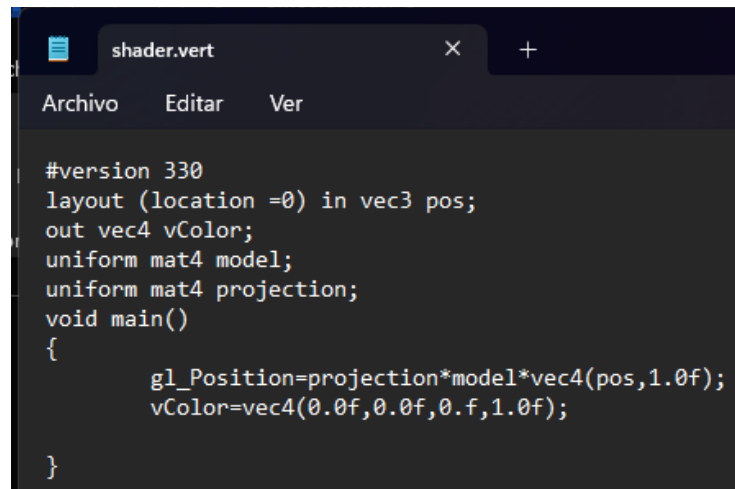
Ejercicio 2:

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

Se crean los nuevos shaders a utilizar, el cual será 1 por cada color (rojo, azul, verde, café, verde oscuro), y guardamos la dirección de en donde se encuentran cada uno de ellos.

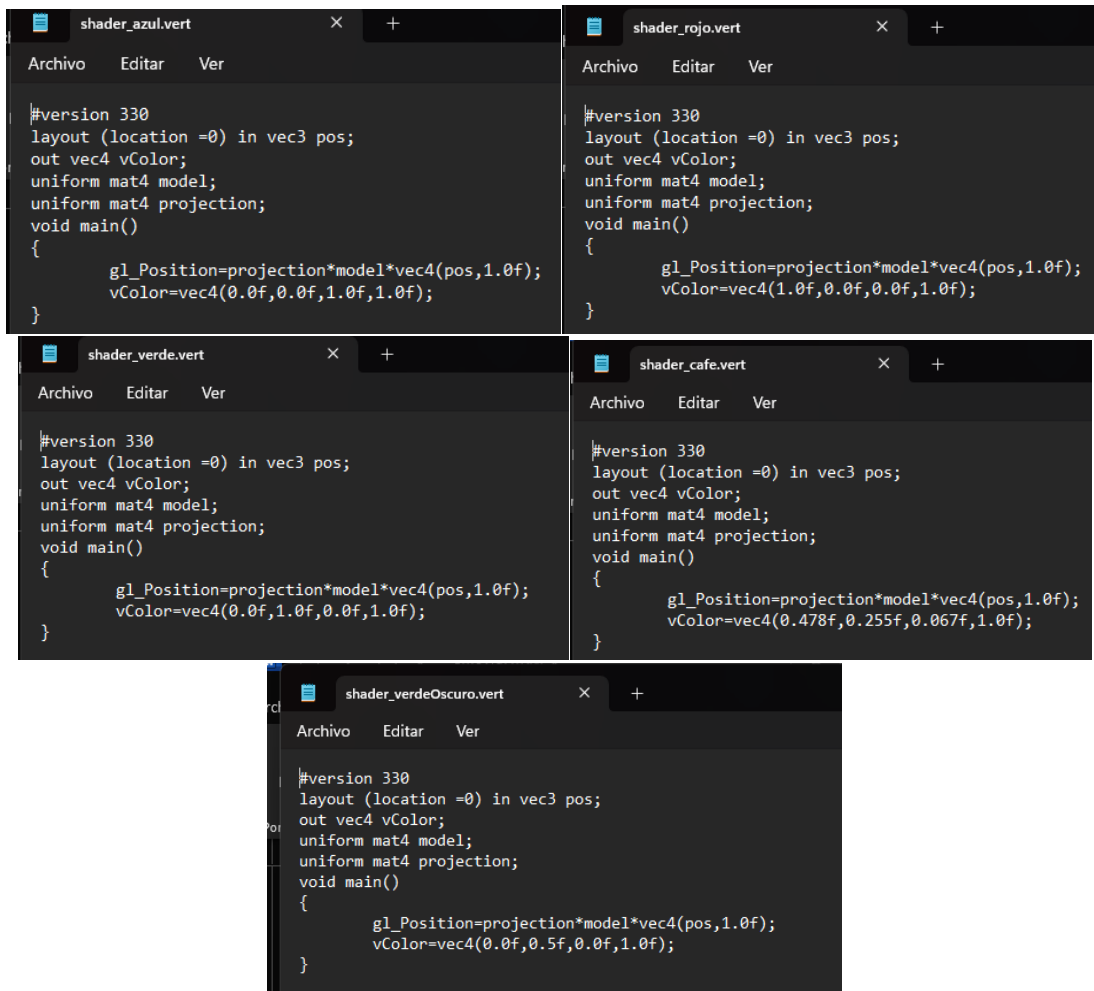
```
26 static const char* fShaderVerde = "shaders/shader_verde.frag";  
27 //shaders nuevos se crearían acá  
28 static const char* vShaderRojo = "shaders/shader_rojo.vert";  
29 static const char* fShaderRojo = "shaders/shader_rojo.frag";  
30 static const char* vShaderVerde = "shaders/shader_verde.vert";  
31 static const char* fShaderVerde = "shaders/shader_verde.frag";  
32 static const char* vShaderAzul = "shaders/shader_azul.vert";  
33 static const char* fShaderAzul = "shaders/shader_azul.frag";  
34 static const char* vShaderCafe = "shaders/shader_cafe.vert";  
35 static const char* fShaderCafe = "shaders/shader_cafe.frag";  
36 static const char* vShaderVerdeOscuro = "shaders/shader_verdeOscuro.vert";  
37 static const char* fShaderVerdeOscuro = "shaders/shader_verdeOscuro.frag";  
38
```

En el shader.vert borramos o comentamos el la instrucción con clamp(), y descomentamos la instrucción anterior.



```
shader.vert  
Archivo Editar Ver  
  
#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,0.0f,1.0f);  
}
```

Ahora empezamos a crear cada uno de los archivos.vert, que son 1 por cada color que necesitamos, el cual en el vector para el color, tendrá su respectivo numero que represente el color que vamos a mostrar.



```
shader_azul.vert
Archivo  Editar  Ver

#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);
}

shader_rojo.vert
Archivo  Editar  Ver

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

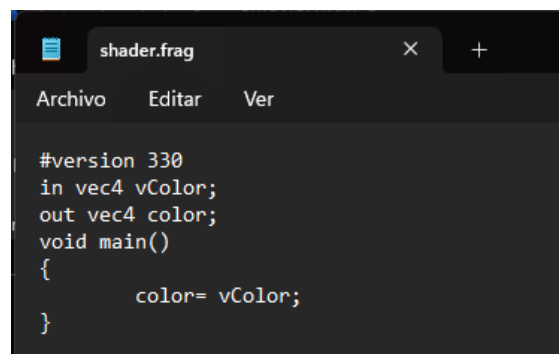
shader_verde.vert
Archivo  Editar  Ver

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

shader_verdeOscuro.vert
Archivo  Editar  Ver

#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,0.5f,0.0f,1.0f);
}
```

Ahora para los .frag copiamos el contenido del **shader.frag** y lo copiamos en cada uno de los .frag que tengamos, a estos se les cambia el nombre para poder identificarlos.



```
shader.frag
Archivo  Editar  Ver

#version 330
in vec4 vColor;
out vec4 color;
void main()
{
    color= vColor;
}
```

shader.frag	17/02/2025 07:25 p. m.	Archivo FRAG	1 KB
shader_azul.frag	17/02/2025 07:25 p. m.	Archivo FRAG	1 KB
shader_cafe.frag	17/02/2025 07:25 p. m.	Archivo FRAG	1 KB
shader_rojo.frag	17/02/2025 07:25 p. m.	Archivo FRAG	1 KB
shader_verde.frag	17/02/2025 07:25 p. m.	Archivo FRAG	1 KB
shader_verdeOscuro.frag	17/02/2025 07:25 p. m.	Archivo FRAG	1 KB

Después en nuestra función `CreateShaders()`, procedemos a crearlos y declararlos para después manipularlos con los índices, esto claramente hace que tengamos que fijarnos a quien insertamos primero para saber el valor del índice al cual pertenece.

```

386 void CreateShaders()
387 {
388
389     Shader *shader1 = new Shader(); //shader para usar índices: objetos: cubo y pirámide
390     shader1->CreateFromFiles(vShader, fShader);
391     shaderList.push_back(*shader1);
392
393     Shader *shader2 = new Shader(); //shader para usar color como parte del VAO: letras
394     shader2->CreateFromFiles(vShaderColor, fShaderColor);
395     shaderList.push_back(*shader2);
396
397     // Shader Cubo Rojo (2)
398     Shader* shaderRojo = new Shader();
399     shaderRojo->CreateFromFiles(vShaderRojo, fShaderRojo);
400     shaderList.push_back(*shaderRojo);
401
402     // Shader Cubo Verde (3)
403     Shader* shaderVerde = new Shader();
404     shaderVerde->CreateFromFiles(vShaderVerde, fShaderVerde);
405     shaderList.push_back(*shaderVerde);
406
407     // Shader Piramide Azul (4)
408     Shader* shaderAzul = new Shader();
409     shaderAzul->CreateFromFiles(vShaderAzul, fShaderAzul);
410     shaderList.push_back(*shaderAzul);
411
412     // Shader Cubo Cafe (5)
413     Shader* shaderCafe = new Shader();
414     shaderCafe->CreateFromFiles(vShaderCafe, fShaderCafe);
415     shaderList.push_back(*shaderCafe);
416
417     // Shader Piramide Verde Oscuro (6)
418     Shader* shaderVerdeOscuro = new Shader();
419     shaderVerdeOscuro->CreateFromFiles(vShaderVerdeOscuro, fShaderVerdeOscuro);
420     shaderList.push_back(*shaderVerdeOscuro);
421
422 }

```

Dentro del `main()`, comentamos el pojection que usa la función `ortho()` que es para 2D, y descomentamos la que usa `perspective()` que es para 3D

```

434 //GLM: MATRICES - 0,
435 //Projection: Matriz de Dimensión 4x4 para indicar si vemos en 2D( orthogonal) o en 3D) perspectiva
436 //glm::mat4 projection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, 0.1f, 100.0f);
437 glm::mat4 projection = glm::perspective(glm::radians(60.0f), mainWindow.getBufferWidth() / mainWindow.getBufferHeight(), 0.1f, 100.0f);
438

```


Ahora pasamos a la renderización de las figuras, el cuales son pirámides y cubos, el cual tenemos que respetar el índice del shader_list para que podamos mostrar el color deseado.

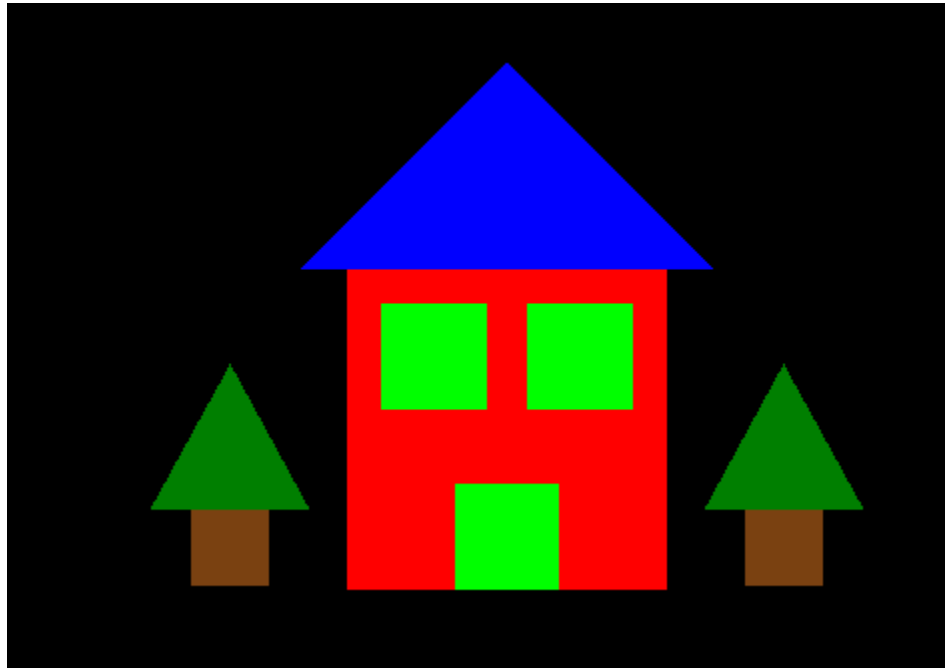
```
450
451 //Renderización de la casa (Cubo Rojo)
452 shaderList[2].useShader(); // shader_rojo
453 uniformModel = shaderList[2].getModelLocation();
454 uniformProjection = shaderList[2].getProjectLocation();
455 model = glm::mat4(1.0);
456 model = glm::translate(model, glm::vec3(0.0f, 0.0f, -15.0f));
457 model = glm::scale(model, glm::vec3(4.0f, 4.0f, 4.0f));
458 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
459 glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
460 meshList[1]->RenderMesh(); // Usar cubo para la casa
461
462 // Renderización del Techo (Piramide Azul)
463 shaderList[4].useShader(); // shader_azul
464 uniformModel = shaderList[4].getModelLocation();
465 uniformProjection = shaderList[4].getProjectLocation();
466 model = glm::mat4(1.0);
467 model = glm::translate(model, glm::vec3(0.0f, 3.8f, -15.0f));
468 model = glm::scale(model, glm::vec3(6.0f, 3.0f, 0.1f));
469 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
470 glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
471 meshList[0]->RenderMesh(); // Usar pirámide para el techo
472
473 // Renderización de la Ventana Izquierda (Cubo Verde)
474 shaderList[3].useShader(); // shader_verde
475 uniformModel = shaderList[3].getModelLocation();
476 uniformProjection = shaderList[3].getProjectLocation();
477 model = glm::mat4(1.0);
478 model = glm::translate(model, glm::vec3(-0.7f, 0.7f, -10.0f));
479 model = glm::scale(model, glm::vec3(1.0f, 1.0f, 0.1f));
480 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
481 glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
482 meshList[1]->RenderMesh(); // Usar cubo para ventanas
483
484 // Renderización de la Ventana Derecha (Cubo Verde)
485 shaderList[3].useShader(); // shader_verde
486 uniformModel = shaderList[3].getModelLocation();
487 uniformProjection = shaderList[3].getProjectLocation();
488 model = glm::mat4(1.0);
489 model = glm::translate(model, glm::vec3(0.7f, 0.7f, -10.0f));
490 model = glm::scale(model, glm::vec3(1.0f, 1.0f, 0.1f));
491 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
492 glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
493 meshList[1]->RenderMesh(); // Usar cubo para ventanas
494
495 // Renderización de la Puerta (Cubo Verde)
496 shaderList[3].useShader(); // shader_verde
497 uniformModel = shaderList[3].getModelLocation();
498 uniformProjection = shaderList[3].getProjectLocation();
499 model = glm::mat4(1.0);
500 model = glm::translate(model, glm::vec3(0.0f, -1.03f, -10.0f));
501 model = glm::scale(model, glm::vec3(1.0f, 1.0f, 0.1f));
502 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
503 glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
504 meshList[1]->RenderMesh(); // Usar cubo para puerta
505
```

```

506 // Renderización del tronco Izquierdo (Cubo Café)
507 shaderList[5].useShader(); // shader_cafe
508 uniformModel = shaderList[5].getModelLocation();
509 uniformProjection = shaderList[5].getProjectLocation();
510 model = glm::mat4(1.0);
511 model = glm::translate(model, glm::vec3(-4.0f, -1.7f, -15.0f));
512 model = glm::scale(model, glm::vec3(1.1f, 1.1f, 0.1f));
513 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
514 glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
515 meshList[1]->RenderMesh(); // Usar cubo para el tronco izquierdo
516
517 // Renderización del tronco Derecho (Cubo Café)
518 shaderList[5].useShader(); // shader_cafe
519 uniformModel = shaderList[5].getModelLocation();
520 uniformProjection = shaderList[5].getProjectLocation();
521 model = glm::mat4(1.0);
522 model = glm::translate(model, glm::vec3(4.0f, -1.7f, -15.0f));
523 model = glm::scale(model, glm::vec3(1.1f, 1.1f, 0.1f));
524 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
525 glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
526 meshList[1]->RenderMesh(); // Usar cubo para el tronco derecho
527
528 // Renderización de la Copa de árbol Izquierdo (Pirámide Verde Oscuro)
529 shaderList[6].useShader(); // shader_verdeOscuro
530 uniformModel = shaderList[6].getModelLocation();
531 uniformProjection = shaderList[6].getProjectLocation();
532 model = glm::mat4(1.0);
533 model = glm::translate(model, glm::vec3(-4.0f, -0.1f, -15.0f));
534 model = glm::scale(model, glm::vec3(2.3f, 2.1f, 0.1f));
535 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
536 glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
537 meshList[0]->RenderMesh(); // Usar pirámide para la copa del árbol
538
539 // Renderización de la Copa de árbol Derecho (Pirámide Verde Oscuro)
540 shaderList[6].useShader(); // shader_verdeOscuro
541 uniformModel = shaderList[6].getModelLocation();
542 uniformProjection = shaderList[6].getProjectLocation();
543 model = glm::mat4(1.0);
544 model = glm::translate(model, glm::vec3(4.0f, -0.1f, -15.0f));
545 model = glm::scale(model, glm::vec3(2.3f, 2.1f, 0.1f));
546 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
547 glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
548 meshList[0]->RenderMesh(); // Usar pirámide para la copa del árbol
549
550

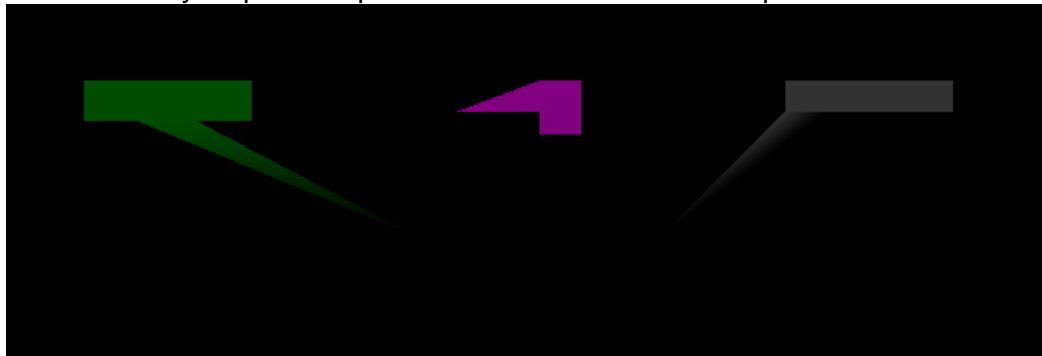
```

Y ya por último compilamos y ejecutamos



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

- A la hora de crear los vértices o las iniciales de mi nombre, pasaba las coordenadas, pero a la hora de ejecutarlo las figuras terminaban distorsionándose, por lo que no sabía el porqué no quedaban. Hasta que me di cuenta que era por la cantidad o el valor de vértices que estaba mandando, ya que no eran los suficientes para completar la figura y es por ello que no salía el ejercicio 1. Checando ya la función con detenimiento, vi que no me cuadraban unos valores y vi que eran por los valores de los vértices que se mandaban.



- En el ejercicio 2 se me complicó a la hora de renderizar ya las figuras, dado que son en 3D y estas no tienen sombra, hacen que la forma en la que se veían al mostrarse, la figura inicial no se viera o como se pedía, y era algo confuso de ver debido a la profundidad, por lo que decidí que al momento de modificar su escala, achicar la parte en el eje Z para que no se viera tan diferente a una figura en 2D.

3. Conclusión:

a. Los ejercicios del reporte: Complejidad, Explicación.

Considero que se vieron muchas cosas, ya que no tanto se nos enseña a como manipular las cosas, sino que un repaso de que hace que, por lo que siento que el nivel de los ejercicios es algo entre medio y alto, al resolverlas te despeja de muchas dudas.

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

Creo que falta alguna retroalimentación para poder mejorar el conocimiento de las funciones que usamos, así como la explicación de los códigos debería de ser más lineal, en vez de cambiar entre archivo y archivo, ya que uno se puede confundir por lo rápido que a veces se explica.

c. Conclusión

Con el desarrollo completo de esta práctica pudimos implementar el manejo y manipulación de los shaders, así como de las figuras 2D y 3D, aunque esto se irá mejorando a como avancemos en las prácticas, de momento hicimos ya uso de varias funciones que nos ayudan a la manipulación de nuestras figuras.

4. Bibliografía en formato APA

- c++ & OpenGL: How do you create a mesh object instance from within a class. (s. f.). Stack Overflow.
<https://stackoverflow.com/questions/37164634/c-opengl-how-do-you-create-a-mesh-object-instance-from-within-a-class>
- Vertex Shader - OpenGL Wiki. (s. f.).
https://www.khronos.org/opengl/wiki/Vertex_Shader
- G-Truc. (s. f.). GitHub - g-truc/glm: OpenGL Mathematics (GLM). GitHub.
<https://github.com/g-truc/glm>