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: RENDÓN HERNÁNDEZ ROBERTO CARLOS

Nº de Cuenta: 420052603

GRUPO DE LABORATORIO: 01

GRUPO DE TEORÍA: 04

SEMESTRE 2024-2

FECHA DE ENTREGA LÍMITE: 24/02/2024

CALIFICACIÓN: _____

REPORTE DE PRÁCTICA:

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

Dentro de esta función se define un arreglo vertices_letras que contiene las coordenadas (X, Y, Z) de los vértices de varios triángulos que forman las letras R, C y H, así como los colores (R, G, B) asociados a cada vértice. Cada letra se dibuja con una serie de triángulos. Por ejemplo, la letra R se compone de 10 triángulos, la letra C de 8 triángulos y la letra H de 6 triángulos. Para cada letra, se especifican los vértices de los triángulos que la componen, junto con los colores correspondientes. Se crea un objeto MeshColor llamado letras utilizando el operador new. Este objeto se utiliza para almacenar la información de los vértices y colores de las letras. Se llama al método CreateMeshColor del objeto letras, pasando como argumentos el arreglo vertices_letras y el tamaño total del arreglo (432 elementos). Finalmente, el objeto letras se agrega a la lista meshColorList.

```
void CrearLetrasyFiguras()
{
    GLfloat vertices_letras[] = { ... }
    MeshColor *letras = new MeshColor();
    letras->CreateMeshColor(vertices_letras,432);
    meshColorList.push_back(letras);
```

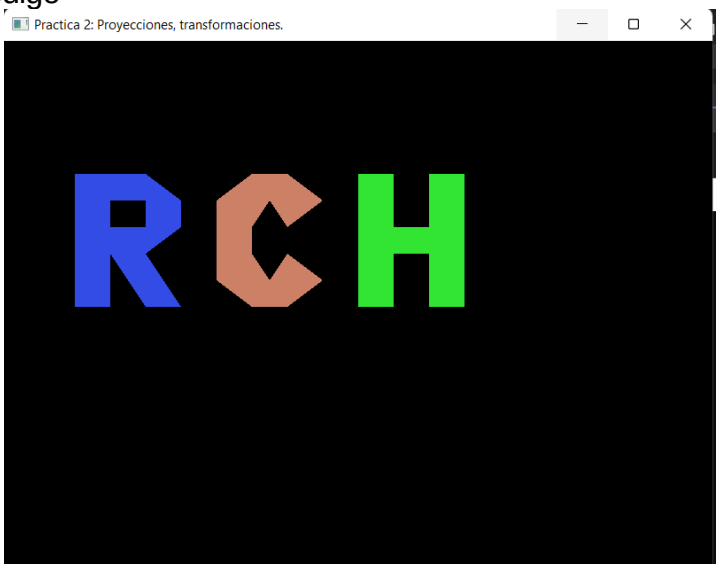
```
// Triangulos para la letra R
//triangulo 1
//X      Y      Z          R      G      B
-0.8f, 0.5f, 0.0f,      0.2f,   0.3f,   0.9f,
-0.8f, 0.4f, 0.0f,      0.2f,   0.3f,   0.9f,
-0.6f, 0.4f, 0.0f,      0.2f,   0.3f,   0.9f,
//triangulo 2
-0.8f, 0.5f, 0.0f,      0.2f,   0.3f,   0.9f,
-0.6f, 0.5f, 0.0f,      0.2f,   0.3f,   0.9f,
-0.6f, 0.4f, 0.0f,      0.2f,   0.3f,   0.9f,
//triangulo 3
-0.8f, 0.4f, 0.0f,      0.2f,   0.3f,   0.9f,
-0.8f, 0.0f, 0.0f,      0.2f,   0.3f,   0.9f,
-0.7f, 0.0f, 0.0f,      0.2f,   0.3f,   0.9f,
//triangulo 4
-0.8f, 0.4f, 0.0f,      0.2f,   0.3f,   0.9f,
-0.7f, 0.4f, 0.0f,      0.2f,   0.3f,   0.9f,
-0.7f, 0.0f, 0.0f,      0.2f,   0.3f,   0.9f,
//triangulo 5
-0.7f, 0.3f, 0.0f,      0.2f,   0.3f,   0.9f,
-0.7f, 0.2f, 0.0f,      0.2f,   0.3f,   0.9f,
-0.6f, 0.2f, 0.0f,      0.2f,   0.3f,   0.9f,
//triangulo 6
-0.7f, 0.3f, 0.0f,      0.2f,   0.3f,   0.9f,
-0.6f, 0.3f, 0.0f,      0.2f,   0.3f,   0.9f,
-0.6f, 0.2f, 0.0f,      0.2f,   0.3f,   0.9f,
//triangulo 7
-0.6f, 0.2f, 0.0f,      0.2f,   0.3f,   0.9f,
-0.6f, 0.5f, 0.0f,      0.2f,   0.3f,   0.9f,
-0.5f, 0.4f, 0.0f,      0.2f,   0.3f,   0.9f,
//triangulo 8
-0.6f, 0.2f, 0.0f,      0.2f,   0.3f,   0.9f,
-0.5f, 0.3f, 0.0f,      0.2f,   0.3f,   0.9f,
-0.5f, 0.4f, 0.0f,      0.2f,   0.3f,   0.9f,
//triangulo 9
-0.6f, 0.0f, 0.0f,      0.2f,   0.3f,   0.9f,
-0.7f, 0.2f, 0.0f,      0.2f,   0.3f,   0.9f,
-0.6f, 0.2f, 0.0f,      0.2f,   0.3f,   0.9f,
//triangulo 10
-0.6f, 0.0f, 0.0f,      0.2f,   0.3f,   0.9f,
-0.5f, 0.0f, 0.0f,      0.2f,   0.3f,   0.9f,
-0.6f, 0.2f, 0.0f,      0.2f,   0.3f,   0.9f,
```

Este bloque de código configura el uso de shaders específicos para renderizar letras o figuras en una escena de OpenGL.

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

//Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para alma
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 P
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[0]->RenderMeshColor();
```

Salida del código



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

1. **Techo**:
    - Se activa el shader correspondiente al techo.
    - Se aumenta el ángulo para alguna animación (posiblemente rotación).
    - Se configuran las matrices de modelo y proyección para el techo.
    - Se renderiza el modelo del techo almacenado en `meshList[0]`.

```
//Techo
shaderList[2].useShader();
angulo += 0.1;
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.50f, -3.50f));
model = glm::scale(model, glm::vec3(2.5f, 1.5f, 1.5f));
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();
```

2. **Casa**:
    - Se activa el shader principal.
    - Se configuran las matrices de modelo y proyección para la casa.
    - Se renderiza el modelo de la casa almacenado en `meshList[1]`.

```
//Casa
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].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, -1.0f, -4.0f));
model = glm::scale(model, glm::vec3(1.5f, 1.5f, 1.5f));
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[1]->RenderMesh();
```

3. **Ventanas y puerta**:
    - Se activa el shader correspondiente a las ventanas y puerta.
    - Se configuran las matrices de modelo y proyección para las ventanas derecha e izquierda.
    - Se renderizan los modelos de las ventanas almacenados en `meshList[1]`.

```
//Ventana Derecha
shaderList[4].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
//Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para almacenar las transformaciones geo
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.35f, -0.6f, -3.2f));
model = glm::scale(model, glm::vec3(0.4f, 0.4f, 0.4f));
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 Izquierda
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.35f, -0.6f, -3.2f));
model = glm::scale(model, glm::vec3(0.4f, 0.4f, 0.4f));
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
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -1.35f, -3.2f));
model = glm::scale(model, glm::vec3(0.4f, 0.5f, 0.4f));
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();
```

4. **Pinos**:
    - Se activa el shader correspondiente a los pinos.
    - Se configuran las matrices de modelo y proyección para los pinos derecho e izquierdo.
    - Se renderizan los modelos de los pinos almacenados en `meshList[1]`.
    - Se configuran las matrices de modelo y proyección para los troncos de los pinos derecho e izquierdo.

- Se renderizan los modelos de los troncos de los pinos almacenados en `meshList[0]`.

```
//Pino Derecho
shaderList[3].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(2.30f, -1.80f, -4.0f));
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES P
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();

//Tronco pino Derecho
shaderList[5].useShader();
uniformModel = shaderList[0].getModelLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(2.20f, -0.65f, -4.0f));
model = glm::scale(model, glm::vec3(1.5f, 2.0f, 1.5f));
model = glm::rotate(model, 135 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES P
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();
```

Salida del código:

Conclusión

Esta práctica muestra cómo configurar y renderizar diferentes elementos en una escena 3D utilizando OpenGL y la biblioteca glm para operaciones matriciales. El modularidad en el uso de shaders permite aplicar efectos visuales específicos a diferentes partes de la escena, mientras que las transformaciones geométricas proporcionan flexibilidad para manipular la posición y el tamaño de los objetos. En conjunto, el código ilustra los fundamentos de la representación y renderizado de escenas tridimensionales utilizando OpenGL.

Bibliografía

LearnTutorials.net. Recuperado el 18 de febrero de 2024, de https://learntutorials.net/es/opengl/topic/10680/vista-y-proyeccion-ogl

LearnOpenGL. (s.f.). Camera. Recuperado el 17 de febrero de 2024, de https://learnopengl.com/Getting-started/Camera

Stack Overflow. (2014). Understanding glm::lookAt(). Recuperado el 17 de febrero de 2024, de https://stackoverflow.com/questions/21830340/understanding-glmlookat

Stack Overflow. (2017, agosto 29). glGenVertexArrays and glGenBuffers arguments [Pregunta y respuestas]. Stack Overflow. https://stackoverflow.com/questions/45860198/glgenvertexarrays-and-glgenbuffers-arguments