FACULTAD DE INGENIERÍA - Course 2019/ 2019

SECRETARÍA/DIVISIÓN: DIVISIÓN DE INGENIERÍA ELÉCTRICA
ÁREA/DEPARTAMENTO: INGENIERÍA EN COMPUTACIÓN

LABORATORIO DE COMPUTACIÓN GRÁFICA E INTERACCIÓN HUMANO COMPUTADORA:

Proyecciones y puertos de vista. Transformaciones Geométricas

Reynaldo Martell Avila

PRÁCTICA 3

Contents

1 Objetivos de aprendizaje

1.1 Objetivos generales:

El alumno repasará como crear buffers de OpenGL, leer archivos, comprenderá los diferentes tipos de proyección y las funciones de la librería glm para crear éstas, así como comprenderá los diferentes sistemas de referencia de OpenGL. Del mismo modo practicará como colocar en la escena diferentes geometrías.

1.2 Objetivos específicos:

El alumno practicará crear geometrías con índices, revisará los sistemas de referencia que se aplican en OpenGL, comprenderá la utilización de la matriz de modelo, vista, proyección y la zona de dibujo.

2 Recursos a emplear

2.1 Software

Sistema Operativo: Windows 7 Ambiente de Desarrollo: Visual Studio 2017.

2.2 Equipos

Los equipos de cómputo con los que cuenta el laboratorio de Computación Gráfica

2.3 Instrumentos

3 Fundamento Teórico

- **Presentación de conceptos.**
  Se mostrará la utilización de índices, se revisará como crear los shaders de fragmento y vértices desde un archivo, se presentará el funcionamiento de variables globales que son tomadas de los shaders, utilizará los diferentes tipos de proyecciones y funciones para crearlas, así como colocar diferentes figuras en pantalla.
- **Datos necesarios.** Librería OpenGL 3.3, librería de creación de ventanas, IDE de desarrollo (Visual Studio 2017.

3.1 Desarrollo de actividades

1. Ejecutar el código base de la práctica **03-SistemasCoordenados**, observar la ejecución.
2. Explicar el código de la Clase **Shader.h** y su implementación **Shader.cpp**.
3. Utilice la Clase Shader para instanciar un objeto de éste tipo, para utilizarlo es necesario incluir la cabecera **#include "Headers/Shader.h"**, para instanciar el objeto utilice: **Shader shader;** y por último agregar la inicialización de los shaders en el método initialize: **shader.initialize("../Shaders/tra "../Shaders/transformaciones.fs");**

4. Localizar y abrir los archivo que contienen el código de los Shaders **../Shaders/transformaciones.vs** y **../Shaders/transformaciones.fs**.

5. Se explican los código de los Shaders.

6. Crear una matriz de proyección en perspectiva en el loop principal:
glm::mat4 projection = glm::perspective(glm::radians(45.0f), (float) screenWidth / (float) screenHeight, 0.01f, 100.0f);

7. Agregar la matriz de modelo vista y proyección, realizar una translación a la matriz de vista en dirección del vector $\bar{v} = (0.0, 0.0, -3.0)$,y por ulimo enviar las matrices a los shaders. Tomar como ejemplo el código de la sección 1.

Ejemplo 1: Ejemplo para envíar las matrices del modelo, vista y proyección.

```
glm::mat4 model = glm::mat4(1.0f);
glm::mat4 view = glm::mat4(1.0f);
view = glm::translate(view, glm::vec3(0.0f, 0.0f, -3.0f));

shader.setMatrix4("model", 1, false, glm::value_ptr(model));
shader.setMatrix4("view", 1, false, glm::value_ptr(view));
shader.setMatrix4("projection", 1, false, glm::value_ptr(projection));
```

8. Agregar la bandera **GL_DEPTH_BUFFER_BIT** a la función **glClear**, con esta bandera se refrescan el buffer de color y el buffer de profundidad.
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);, despúes de esta línea de código agregar la activación del shader **shader.turnOn();**, esta función se encarga de indicar que programa usar, es equivalente a **glUseProgram(shaderProgramID);**.

9. Ejecutar el programa y revisar sí la geometría corresponde a la especificada en el arreglo.

10. Crear otro buffer de tipo **GL_ELEMENT_ARRAY_BUFFER**.

Ejemplo 2: Ejemplo para crear un arreglo de indices.

```
glGenBuffers(1, &EBO);

.
.
.

// Cambiamos el estado para indicar que usaremos el id del EBO como Arreglo de indices
    (GL_ELEMENT_ARRAY_BUFFER)
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
// Copiamos los datos de los vertices a memoria del procesador grafico
//          TIPO DE BUFFER   TAMANIO        DATOS   MODO (No cambian los datos)
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);
```

No olvidar eliminar el buffer de indices en el método **destroy** como se muestra en el ejemplo 3.

Ejemplo 3: Eliminar el buffer de indices.

```
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
glDeleteBuffers(1, &EBO);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
```

En éste se almacenará el arreglo **indices**, el arreglo indices indica la conexión de cada vertice, ésto ahorra la cantidad de memoria que se ocupa, aún que por otro lado se requiere un arreglo de enteros. Remplazar la función **glDrawArrays(GL_TRIANGLES, 0, 3);** por **glDrawElements(GL_TRIANGLES, 24, GL_UNSIGNED_INT, 0);**

11. Cambiar el sistema de referencia de puerto de vista con la función **glViweport(GLint x, GLint y, GLsizei width, GLsizei height)**:

    - x, y: Son las coordenadas de la esquina inferior izquierda de la zona de dibujo
    - width, height: Especifican el ancho y alto de la zona de dibujo.

    Probar con los siguientes parámetros:

    - Dividir a la mitad la zona de dibujo en el eje y.
    - La esquina inferior en las coordenadas (0, 0) y el tamaño de la zona de dibujo a la mitad de cada eje.

12. Colocar dos cubos en dos diferentes posiciones y diferente profundidad en z, ejemplo

Ejemplo 4: Ejemplo para colocar dos cubos.

```
shader.turnOn();

glm::mat4 view = glm::mat4(1.0f);
view = glm::translate(view, glm::vec3(0.0f, 0.0f, -3.0f));

shader.setMatrix4("view", 1, false, glm::value_ptr(view));
shader.setMatrix4("projection", 1, false, glm::value_ptr(projection));

glm::mat4 model = glm::translate(glm::mat4(1.0), glm::vec3(-1.0, 1.0, -2.0));
shader.setMatrix4("model", 1, false, glm::value_ptr(model));

// Se indica el buffer de datos y la estructura de estos utilizando solo el id del VAO
glBindVertexArray(VAO);
// Primitiva de ensamble
glDrawElements(GL_TRIANGLES, 24, GL_UNSIGNED_INT, 0);
glBindVertexArray(0);

model = glm::translate(glm::mat4(1.0), glm::vec3(2.0, 1.0, -4.0));
shader.setMatrix4("model", 1, false, glm::value_ptr(model));

// Se indica el buffer de datos y la estructura de estos utilizando solo el id del VAO
glBindVertexArray(VAO);
// Primitiva de ensamble
```

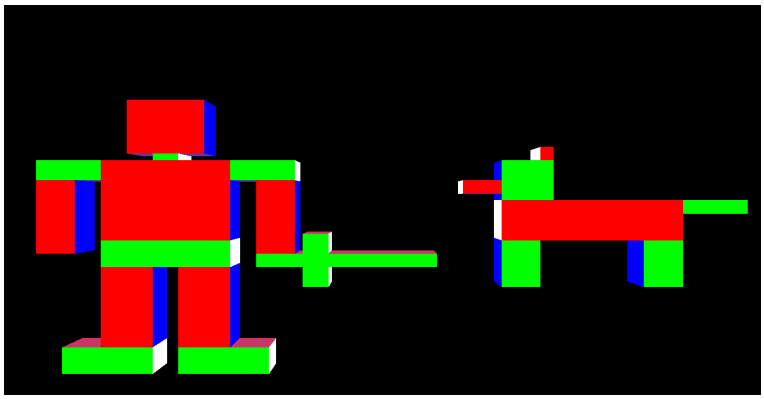
```
glDrawElements(GL_TRIANGLES, 24, GL_UNSIGNED_INT, 0);
glBindVertexArray(0);

shader.turnOff();
```

13. Se explica y se cambian los parámetros que definen los diferentes tipos de proyecciones.

    - **glm::mat4 projection = glm::perspective(glm::radians(45.0f), (float) screenWidth / (float) screenHeight, 0.01f, 100.0f);**
    - **glm::mat4 projection = glm::frustum(-0.005, 0.005, -0.005, 0.005, 0.01, 100.0);**
    - **glm::mat4 projection = glm::ortho(-5.0, 5.0, -5.0, 5.0, 0.01, 10.0);**

3.2 Ejercicios

1. Los siguientes ejercicios se deben crear con un cubo con un color por cada lado. Se debe crear otro VAO y VBO
2. Utilizando un cubo unitario con centro en el origen como primitiva, y la transformación de translación y escalamiento, se creará una escena con las siglas CG 2019.
3. Se procede a crear un par de figuras instanciado el cubo y aplicando transformaciones básicas a cada una de las instancias.



4. Crear la misma forma de la estrella de la práctica 2 con indices.
5. Deben subir sus ejercicios en Github y colocar la liga en su reporte.

4 Observaciones y Conclusiones

5 Anexos

1. Cuestionario previo.

(a) ¿Qué es una clase en c++?

(b) ¿Qué es un constructor y destructor de la clase, y cómo se declara en c++?

(c) ¿Cómo se instancia un objeto en c++?

(d) Investigar como abrir un archivo en c++.

(e) Investigue para qué sirve la función **glViewport** y que parámetros recibe.

(f) Investigue que es la matriz de Modelo, Vista, Proyección.

(g) ¿Qué es una proyección e investigue los tipos de proyecciones, en el ámbito de gráficos?

(h) Que utilidad tiene las funciones **glm::ortho**, **glm::frustum** y **glm::perspective**, y que son los parámetros que reciben.

(i) Para qué sirve la función **glfwSetWindowPos** y que parámetros recibe.

(j) ¿Cuáles son las transformaciones geométricas básicas en tres dimensiones y sus matrices asociadas?

(k) Investigué para sirve la función **glm::scale**, **glm::translate**, **glm::rotate**, y que parámetros reciben.

2. Actividad de investigación previa.

   (a) Realizar un **git pull origin master** y un **git pull myRepo master**, antes de comenzar la práctica.