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:

Introducción a OpenGL

Reynaldo Martell Avila

PRÁCTICA 2

Contents

1 Objetivos de aprendizaje

1.1 Objetivos generales:

El alumno aprenderá los conceptos básicos de OpenGL, el paradigma de programación y las funciones que se utilizan para renderizado de OpenGL.

1.2 Objetivos específicos:

El alumno aprenderá las funciones para el dibujo de primitivas geométricas en la pantalla, el paradigma de programación de OpenGL, a crear sus primeros Shaders de vértices y fragmentos, crear y utilizar el contexto de OpenGL, los conceptos de Vertex array object (VAO) y Vertex buffer object (VBO), así como la librería para crear ventanas y manejo de eventos.

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 le da a conocer al alumno los comandos **glGenVertexArrays**, **glBindVertexArray**, **glGenBuffers**, **glBindBuffer**, **glBufferData**, **glViewport**, nomenclatura de Vertex Shader, Fragment Shader, agregar, compilar y verificar y usar shaders, e identificarlos como bloques de información, se explica cuáles son los parámetros que pueden recibir estos comandos y eso como afecta a lo dibujado. Posteriormente se utilizará los atributos de vértices usando **glVertexAttribPointer**, **glEnableVertexAttribArray**.

- **Datos necesarios.** Librería OpenGL 3.3, librería de creación de ventanas, IDE de desarrollo (Visual Studio 2017.

3.1 Desarrollo de actividades

1. Navegar hasta el directorio de trabajo de la práctica pasada (Donde se clono el repositorio), abrir un bash de git y teclear **git pull origin master**. Debe validar que la actualización esté correcta.

2. Ejecutar el proyecto **02-IntroOpenGL** y revisar el funcionamiento del programa.

3. Se explica el uso de los eventos de los dispositivos convencionales mouse y teclado (Sí es necesario).

4. En base al código de ejemplo 1 agregar el shader de fragmento, no olvidar adjuntar el fragment shader al programa **glAttachShader(shaderProgramID, fragmentShaderID);** y utilizar en el loop principal el programa con los shaders **glUseProgram(shaderProgramID);**

Listing 1: Ejemplo para crear el Shader de Fragmento.

```
   // Se crea el id del Fragment Shader
   fragmentShaderID = glCreateShader(GL_FRAGMENT_SHADER);
   // Se agrega el codigo fuente al ID
   glShaderSource(fragmentShaderID, 1, &fragmentShaderSource, NULL);
   // Compilacion de Fragment Shader
   glCompileShader(fragmentShaderID);
   // Se obtiene el estatus de la compilacion del Fragment shader
   glGetShaderiv(fragmentShaderID, GL_COMPILE_STATUS, &success);
   if(!success){
      // En caso de error se obtiene el error y lanza mensaje con error
      glGetShaderInfoLog(fragmentShaderID, 512, NULL, infoLog);
      std::cout << "Error al compilar el FRAGMENT_SHADER." << infoLog << std::endl;
   }
```

5. Se muestra el ejemplo que crea un VAO, VBO, creación de un buffer y transferencia de memoria a la GPU, se muestra como reciben los bloques de memoria los Vertex shader y Fragment Shader.

6. Crear un estructura como se muestra en el código de la sección 2, en esta estructura se tienen dos atributos de vertices (Posición y color).

Listing 2: Ejemplo para crear el Shader de Fragmento.

```
typedef struct _Vertex {
   float m_Pos[3];
   float m_Color[3];
} Vertex;
```

7. Modificar el arreglo de vertices contemplando el nuevo atributo, como se muestra en el ejemplo 3, ¿Que forma tiene?.

Listing 3: Arreglo de vertices.

```
   // Se definen los vertices de la geometria a dibujar
   Vertex vertices [] =
   {
      {{-0.5f,-0.5f, 0.5f}, {1.0f, 0.0f, 0.0f}},
      {{ 0.5f,-0.5f, 0.5f}, {0.0f, 1.0f, 0.0f}},
      {{ 0.5f, 0.5f, 0.5f}, {0.0f, 0.0f, 1.0f}},
      {{-0.5f,-0.5f, 0.5f}, {1.0f, 0.0f, 0.0f}},
      {{ 0.5f, 0.5f, 0.5f}, {0.0f, 0.0f, 1.0f}},
```

```
    {{-0.5f, 0.5f, 0.5f}, {1.0f, 0.0f, 1.0f}}
};
```

8. Obtenga el tamaño del arreglo de vertices, el tamaño de solo un vertice y el tamaño del atributo posición, imprimirlos en la consola como se muestra en el ejemplo 4.

Listing 4: Tamaño del buffer, vertice y atributo posición.

```
size_t bufferSize = sizeof(vertices);
size_t vertexSize = sizeof(vertices[0]);
size_t rgbOffset = sizeof(vertices[0].m_Pos);

std::cout << "Vertices:" << std::endl;
std::cout << "bufferSize:" << bufferSize << std::endl;
std::cout << "vertexSize:" << vertexSize << std::endl;
std::cout << "rgbOffset:" << rgbOffset << std::endl;
```

9. Modificar el linkeo de los atributos para el buffer anterior 7, ádemas agregar como entrada al shader de vertices (Ejemplo 5) y fragmento (Ejemplo 6). Realice un diagrama de la estructura de los datos y sus atributos describiendo cada uno de los parámetros.

Listing 5: Vertex Shader.

```
#version 330 core
layout (location = 0) in vec3 in_position;
layout (location = 1) in vec3 in_color;

out vec3 our_color;

uniform mat4 projection;
uniform mat4 view;
uniform mat4 model;

void main(){

   gl_Position = projection * view * model * vec4(in_position, 1.0);
   our_color = in_color;

}
```

Listing 6: Fragment Shader.

```
#version 330 core

in vec3 our_color;
out vec4 color;

void main(){

```

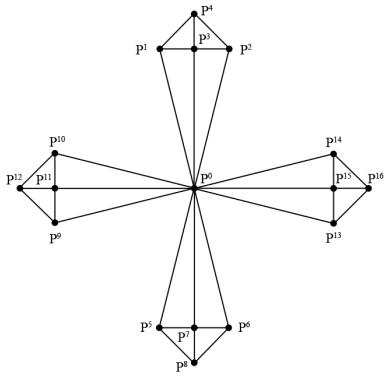
```
    color = vec4(our_color, 1.0);

}
```

Listing 7: Linkeo de los atributos.

```
    // Se crea un indice para el atributo del vertice posicion, debe corresponder al location
        del atributo del shader
    // indice del atributo, Cantidad de datos, Tipo de dato, Normalizacion, Tamanio del bloque
        (Stride), offset
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, vertexSize, (GLvoid*)0);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, vertexSize, (GLvoid*)rgbOffset);
    // Se habilita el atributo del vertice con indice 0 (posicion)
    glEnableVertexAttribArray(0);
    glEnableVertexAttribArray(1);
```

10. Modificar el main.cpp para dibujar las siguientes figuras, se deben usar dos VAOs y VBOs diferentes. Finalmente, con la tecla E se muestra la estrella y con la tecla C la casa.



11. Se debe reportar y subir a github todos sus ejercicios.

4 Observaciones y Conclusiones

5 Anexos

1. Cuestionario previo.
   (a) ¿Qué es un polígono?
   (b) ¿Qué es un polígono convexo y cóncavo?
   (c) ¿Qué es un pixel?
   (d) ¿Qué es OpenGL?

(e) ¿Qué es el contexto de OpenGL?

(f) ¿Cuáles son las etapas básicas del pipeline de renderizado de OpenGL?

(g) ¿Qué es un vertex shader?

(h) ¿Qué es un fragment shader?

(i) ¿Qué es un pixel?

(j) ¿Qué son las NDC (Normalized device coordinates)

2. Actividad de investigación previa.

   (a) ¿Qué es la resolución de un dispositivo?, investigue y anote la resolución del monitor de su computadora principal.