



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

<b>1</b>	<b>Objetivos de aprendizaje</b>	<b>2</b>
1.1	Objetivos generales: . . . . .	2
1.2	Objetivos específicos: . . . . .	2
<b>2</b>	<b>Recursos a emplear</b>	<b>2</b>
2.1	Software . . . . .	2
2.2	Equipos . . . . .	2
2.3	Instrumentos . . . . .	2
<b>3</b>	<b>Fundamento Teórico</b>	<b>2</b>
3.1	Desarrollo de actividades . . . . .	2
<b>4</b>	<b>Observaciones y Conclusiones</b>	<b>5</b>
<b>5</b>	<b>Anexos</b>	<b>5</b>

# 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);`

Ejemplo 1: Ejemplo para crear el Shader de Fragmento.

```

1 // Se crea el id del Fragment Shader
2 fragmentShaderID = glCreateShader(GL_FRAGMENT_SHADER);
3 // Se agrega el codigo fuente al ID
4 glShaderSource(fragmentShaderID, 1, &fragmentShaderSource, NULL);
5 // Compilacion de Fragment Shader
6 glCompileShader(fragmentShaderID);
7 // Se obtiene el estatus de la compilacion del Fragment shader
8 glGetShaderiv(fragmentShaderID, GL_COMPILE_STATUS, &success);
9 if(!success){
10     // En caso de error se obtiene el error y lanza mensaje con error
11     glGetShaderInfoLog(fragmentShaderID, 512, NULL, infoLog);
12     std::cout << "Error al compilar el FRAGMENT_SHADER." << infoLog << std::endl;
13 }

```

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).

Ejemplo 2: Ejemplo para crear el Shader de Fragmento.

```

1 typedef struct _Vertex {
2     float m_Pos[3];
3     float m_Color[3];
4 } Vertex;

```

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

Ejemplo 3: Arreglo de vertices.

```

1 // Se definen los vertices de la geometria a dibujar
2 Vertex vertices [] =
3 {
4     {{-0.5f,-0.5f, 0.5f}, {1.0f, 0.0f, 0.0f}},
5     {{ 0.5f,-0.5f, 0.5f}, {0.0f, 1.0f, 0.0f}},
6     {{ 0.5f, 0.5f, 0.5f}, {0.0f, 0.0f, 1.0f}},
7     {{-0.5f,-0.5f, 0.5f}, {1.0f, 0.0f, 0.0f}},
8     {{ 0.5f, 0.5f, 0.5f}, {0.0f, 0.0f, 1.0f}},

```

```

9 |         {{-0.5f, 0.5f, 0.5f}, {1.0f, 0.0f, 1.0f}}
10 |     };

```

- 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.

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

```

1 |     size_t bufferSize = sizeof(vertices);
2 |     size_t vertexSize = sizeof(vertices[0]);
3 |     size_t rgbOffset = sizeof(vertices[0].m_Pos);
4 |
5 |     std::cout << "Vertices:" << std::endl;
6 |     std::cout << "bufferSize:" << bufferSize << std::endl;
7 |     std::cout << "vertexSize:" << vertexSize << std::endl;
8 |     std::cout << "rgbOffset:" << rgbOffset << std::endl;

```

- Modificar el linkeo de los atributos para el buffer anterior 7, además 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.

Ejemplo 5: Vertex Shader.

```

1 | #version 330 core
2 | layout (location = 0) in vec3 in_position;
3 | layout (location = 1) in vec3 in_color;
4 |
5 | out vec3 our_color;
6 |
7 | uniform mat4 projection;
8 | uniform mat4 view;
9 | uniform mat4 model;
10 |
11 | void main(){
12 |
13 |     gl_Position = projection * view * model * vec4(in_position, 1.0);
14 |     our_color = in_color;
15 |
16 | }

```

Ejemplo 6: Fragment Shader.

```

1 | #version 330 core
2 |
3 | in vec3 our_color;
4 | out vec4 color;
5 |
6 | void main(){
7 |

```

```

8 | | color = vec4(our_color, 1.0);
9 | |
10| }

```

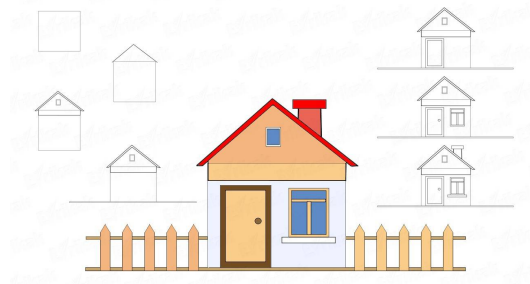
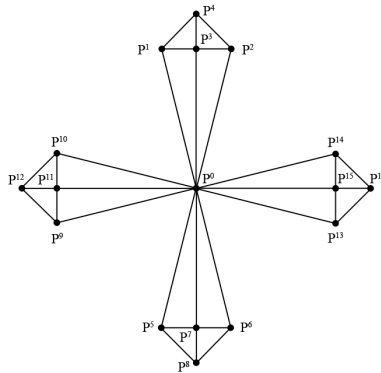
Ejemplo 7: Linkeo de los atributos.

```

1 | // Se crea un indice para el atributo del vertice posicion, debe corresponder al location
  | // del atributo del shader
2 | // indice del atributo, Cantidad de datos, Tipo de dato, Normalizacion, Tamano del bloque
  | // (Stride), offset
3 | glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, vertexSize, (GLvoid*)0);
4 | glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, vertexSize, (GLvoid*)rgbOffset);
5 | // Se habilita el atributo del vertice con indice 0 (posicion)
6 | glEnableVertexAttribArray(0);
7 | 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.