Programación de Videojuegos

Universidad Autónoma de San Luis Potosí
Facultad de Ingeniería
Parcial 1
Angel de Jesús Maldonado Juárez
Actividad 1
13 de febrero del 2023

Elementos del buffer con OpenGL

En programación gráfica se le puede indicar a la tarjeta la distribución (*layout*) del buffer, es decir, qué datos y cómo están distribuidos en el arreglo de locaciones. Esta distribución puede ser personalizada o ajustada de acuerdo a las necesidades del programa.

La actividad consiste en representar la siguiente figura, creando la distribución del buffer:

Utilizando la plantilla de la clase con los *shaders* listos para recibir el color mediante los elementos del buffer:

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aColor;

out vec3 ourColor;

void main()
```

El buffer (arreglo) en el cual se define la distribución de los datos *vértice* y color es el siguiente:

```
float vertices[] = {
    // Figura 1
    0.0f, 0.5f, 0.0f,       0.75f, 0.54f, 0.1f,     // 0
    -0.5f, 0.0f, 0.0f,      0.67f, 0.12f, 0.07f,    // 1
    0.0f, -0.5f, 0.0f,      0.75f, 0.54f, 0.1f,     // 2
    0.5f, 0.0f, 0.0f,       0.67f, 0.12f, 0.07f,    // 3
    // Figura 2 y 3
    0.0f, 0.4f, 0.0f,       0.75f, 0.54f, 0.1f,     // 4
    -0.2f, 0.2f, 0.0f,      0.67f, 0.12f, 0.07f,    // 5
    0.0f, 0.0f, 0.0f,       0.75f, 0.54f, 0.1f,     // 6
    0.2f, -0.2f, 0.0f,      0.67f, 0.12f, 0.07f,    // 8
    0.0f, -0.4f, 0.0f,      0.75f, 0.54f, 0.1f      // 9
};
```

Posteriormente, se define un arreglo en donde se especifica el orden en el cual se *pintan* los vértices:

```
unsigned int indices[] = {
    0, 1, 2,
    0, 2, 3,
    4, 5, 6,
    6, 7, 8
};
```

Posteriormente, en la función `GeneracionBuffer()` se crean los buffers `VAO` (Vertex Array Buffer), `EBO` (Element Buffer Object), `VBO` (Vertex Buffer Object), en donde el primero crea la inicialización para distribuir el buffer (Array), el segundo es para indicar los índices del objeto en pantalla, y el último declara la información de los vértices como tal (vértice - color en este caso).

```
void GeneracionBuffer()
{
    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);
    glGenBuffers(1, &EBO);
    glBindVertexArray(VAO);

    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);

    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);

    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 * sizeof(float)));
    glEnableVertexAttribArray(1);

    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindVertexArray(0);
}
```

Finalmente, en la función de `updateWindow()` se carga el `VAO` y se utiliza la función `glDrawElements()`, y se indica en el parámetro `count` la cantidad de indices que contiene el arreglo `indices`:

```
void updateWindow(GLFWwindow* window, Shader ourShader)
{
    while (!glfwWindowShouldClose(window))
    {
        processInput(window);

        glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);

        ourShader.use();

        glBindVertexArray(VAO);
        glDrawElements(GL_TRIANGLE_STRIP, 12, GL_UNSIGNED_INT, 0);
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);

        glfwSwapBuffers(window);
        glfwPollEvents();
    }
}
```

Al ejecutar el programa se obtiene el siguiente resultado: