- Proyecto que dibuje primitivas geométricas con independencia de colores (Ejemplo 03):
    - Se creará el proyecto jOpenGLEjem3
    - Se agregarán las librerías GLFW y OpenGL
    - Programando la aplicación
        - Tal igual como el ejemplo anterior (Ejemplo 01), se trabajará con una clase *"vcd"* que permitirá gestionar los mensajes de alerta a la hora de crear una ventana; de la misma forma se usará el evento inicializador: *"ini_GLFW"* con su modificación respectiva (ver Ejemplo 02).
        - Se creará la clase Java: *"Sombreador"* que mejora la interacción del anterior *"genProgSombreador"* permitiendo gestionar las compilaciones y vinculaciones de los sombreadores (shaders) y cargar el *Programa Sombreador:*

```
6   package jopenglejem3;
7
8   import java.io.BufferedReader;
9   import java.io.File;
10  import java.io.FileReader;
11  import java.io.IOException;
12
13  import static org.lwjgl.opengl.GL20.*;
14
15  /**
16   *
17   * @author LAPTOP-MAX
18   */
19  public class Sombreador {
20      public int ID;
21
22      //Constructor de la clase
23      public Sombreador(String archVertice, String archFragmento, boolean relleno){
24          //1. recuperar el código fuente de vértice y fragmento desde el directorio raíz
            String cadVertice = "", cadFragmento = "", cadLin = "";
            File arch = null;
27          FileReader fr = null;
            BufferedReader br = null;
29          try{
30              arch = new File("src\\jopenglejem3\\"+archVertice);
31              fr = new FileReader (arch.getAbsolutePath());
32              br = new BufferedReader(fr);
33              while ((cadLin = br.readLine())!=null) {
34                  cadVertice =  cadVertice + cadLin + "\n";
35              }
                cadLin = "";
37              arch = new File("src\\jopenglejem3\\"+archFragmento);
38              fr = new FileReader (arch.getAbsolutePath());
39              br = new BufferedReader(fr);
40              while ((cadLin = br.readLine())!=null) {
41                  cadFragmento =  cadFragmento + cadLin + "\n";
42              }
43          }catch(IOException ex){
44              System.out.println("Error al leer archivo de compilación:\n"+
45                                 "Descripción: "+ex+"\n"+
46                                 "Verificar!!!");
                ex.printStackTrace();
48              System.exit(1);
49          }finally{
50              try{
51                  if( null != fr ){
52                      fr.close();
53                  }
                }catch (Exception ex2){
                    ex2.printStackTrace();
56              }
57          }
58
59          //2. Compilar los Sombreadores y el programa
60          int vertexShader = glCreateShader(GL_VERTEX_SHADER);
61          glShaderSource(vertexShader,cadVertice);
62          glCompileShader(vertexShader);
63          verifErrorComp(vertexShader,"VERTICE");
64
65          int fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
66          glShaderSource(fragmentShader,cadFragmento);
67          glCompileShader(fragmentShader);
68          verifErrorComp(fragmentShader,"FRAGMENTO");
```

```
69
70              ID = glCreateProgram();
71              glAttachShader(ID, vertexShader);
72              glAttachShader(ID, fragmentShader);
73              glLinkProgram(ID);
74              verifErrorComp(ID,"PROGRAMA");
75
76              glDeleteShader(vertexShader);
77              glDeleteShader(fragmentShader);
78
79              if(!relleno){glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);}
80          }
81
82          public void use(){
83              glUseProgram(ID);
84          }
85
86          private static void verifErrorComp(int shader, String tipo){
87              int success, infoLogLength;
88              if(!"PROGRAMA".equals(tipo)){
89                  success =  glGetShaderi(shader, GL_COMPILE_STATUS);
90                  if(success == GL_FALSE){
91                      infoLogLength = glGetShaderi(shader, GL_INFO_LOG_LENGTH);
92                      System.out.println("Error al compilar el: "+tipo+"\n"+
93                                         "Descripción: "+glGetShaderInfoLog(shader,infoLogLength)+"\n"+
94                                         "Verificar!!!");
95                  }
96              }else{
97                  success =  glGetProgrami(shader, GL_LINK_STATUS);
98                  if(success == GL_FALSE){
99                      infoLogLength = glGetProgrami(shader, GL_INFO_LOG_LENGTH);
100                     System.out.println("Error al vincular los Sombreadores:\n"+
101                                        "Descripción: "+glGetProgramInfoLog(shader,infoLogLength)+"\n"+
102                                        "Verificar!!!");
103                 }
104             }
105         }
106 }
107
```

Lo relevante de esta clase está *primero*, en definir su constructor, debido a que cargará los archivos de texto plano para compilar los Vértices y Fragmentos (*"verticeSomb.vs"; "fragmentoSomb.fs"*); esto es útil pues si se quieren modificar los códigos de compilación, se harán directamente en estos archivos, mostrando flexibilidad para su gestión; para ello, se hace uso de la gestión de archivos mediante librerías ***File, FileReader y BufferedReader.*** La idea elemental es que estos archivos sean cargados en cadenas String: *cadVertice*; *cadFragmento*, e integrarlos en la aplicación:

vcd.java | Sombreador.java | JOpenGLEjem3.java | verticeSomb.vs | fragmentoSomb.fs

Origen | History

```
1  #version 330 core
2  layout (location = 0) in vec3 aPos;
3  layout (location = 1) in vec3 aColor;
4
5  out vec3 ourColor;
6
7  void main()
8  {
9      gl_Position = vec4(aPos,1.0f);
10     ourColor = aColor;
11 }
```

vcd.java | Sombreador.java | JOpenGLEjem3.java | verticeSomb.vs | fragmentoSomb.fs

Origen | History

```
1  #version 330 core
2  out vec4 FragColor;
3
4  in vec3 ourColor;
5
6  void main()
7  {
8      FragColor = vec4(ourColor,1.0f);
9  }
10
```

En la *segunda parte* se realizará el control de las compilaciones en tiempo de ejecución para *vertexShader*, *fragmentShader* y posteriormente en una variable *ID* (ProgramShader), que será el *shader* final donde se realizará la gestión de dibujo. En estos procesos se crea un evento de nombre: *"verifErrorComp"* encargado de gestionar los mensajes de alerta y posteriormente el evento *"use()"* encargado de comenzar a usar el *ID*.

- Finalmente, el evento ***"Main"*** quedaría así:

```
6   package jopenglejem3;
7   import java.nio.FloatBuffer;
8   import org.lwjgl.*;
9   import org.lwjgl.glfw.*;
10  import org.lwjgl.opengl.*;
11
12  import static org.lwjgl.glfw.Callbacks.*;
13  import static org.lwjgl.glfw.GLFW.*;
14  import static org.lwjgl.opengl.GL30.*;
15  import static org.lwjgl.system.MemoryUtil.*;
16  /**
```

```
 *
 * @author LAPTOP-MAX
 */
public class JOpenGLEjem3 {

    /**
     * @param args the command line arguments
     */

    private static vcd ini_GLFW(long ventana, int ancho, int alto, String titulo){
        vcd objVCD = new vcd();

        glfwInit();
        glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
        glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
        glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
        glfwWindowHint(GLFW_RESIZABLE, GLFW_TRUE);

        ventana = glfwCreateWindow(ancho, alto, titulo, NULL, NULL);
        if(ventana == NULL){
            objVCD.setMsj("Fallo al crear la ventana con GLFW");
            objVCD.setVentana(NULL);
            return objVCD;
        }

        glfwSetFramebufferSizeCallback(ventana, (objVentana, an, al)->{
            glViewport(0, 0, an, al);
        });

        glfwSetKeyCallback(ventana, (objVentana, tecla, scancode, accion, mods) -> {
            if (tecla == GLFW_KEY_ESCAPE && accion == GLFW_RELEASE) {
                glfwSetWindowShouldClose(objVentana, true);
            }
        });

        GLFWVidMode vidmode = glfwGetVideoMode(glfwGetPrimaryMonitor());
        glfwSetWindowPos(ventana,(vidmode.width() - ancho) / 2,(vidmode.height() - alto) / 2);

        glfwMakeContextCurrent(ventana);
        glfwSwapInterval(1);

        GL.createCapabilities();

        objVCD.setVentana(ventana);
        return objVCD;
    }

    private static int genObjaDib(float vertices[],float colores[]){
        int pVBO = 0,cVBO = 0 , VAO = 0;
        VAO = glGenVertexArrays();
        glBindVertexArray(VAO);

        //Búfer y Atributos para posición de coordenadas 3D (vértice)
        pVBO = glGenBuffers();
        glBindBuffer(GL_ARRAY_BUFFER, pVBO);
        FloatBuffer vbo = BufferUtils.createFloatBuffer(vertices.length); //vertex buffer object
        vbo.put(vertices);
        vbo.flip();
        glBufferData(GL_ARRAY_BUFFER, vbo, GL_STATIC_DRAW);
        glVertexAttribPointer(0, 3, GL_FLOAT, false, 0, 0);
        glEnableVertexAttribArray(0);
        glBindBuffer(GL_ARRAY_BUFFER, 0); //liberando búfer vinculado

        //Atributo para posición de colores RGB
        cVBO = glGenBuffers();
        glBindBuffer(GL_ARRAY_BUFFER, cVBO);
        FloatBuffer cbo = BufferUtils.createFloatBuffer(colores.length); //color buffer object
        cbo.put(colores);
        cbo.flip();
        glBufferData(GL_ARRAY_BUFFER, cbo, GL_STATIC_DRAW);
        glVertexAttribPointer(1, 3, GL_FLOAT, false, 0, 0);
        glEnableVertexAttribArray(1);
        glBindBuffer(GL_ARRAY_BUFFER, 0); //liberando búfer vinculado

        //Liberando array
        glBindVertexArray(0);
        return VAO;
    }

    public static void main(String[] args) {
        long ventana = NULL;
        vcd venConDat = new vcd();
        int objDib;
        try {
            //Se Inicializa GLFW, si todo es correcto; se crea la ventana
            venConDat = ini_GLFW(ventana,1024, 720, "Ejem 3: OpenGL con Java - Colores y Objetos");
            if (!"".equals(venConDat.getMsj())) {
                System.out.println(venConDat.getMsj());
                glfwTerminate();
            }
            else {
                ventana = venConDat.getVentana();
            }

            //Se instancia la clase Sombreador y se compilan los sombreadores y el Programa
            Sombreador objSomb = new Sombreador("verticeSomb.vs","fragmentoSomb.fs",false);

```

```
114                 //Se ingresa las coordenadas y colores para dibujar la(s) primitiva(s)
115                 float vertices[] = {
116                     0.5f, -0.5f, 0.0f,
117                    -0.5f, -0.5f, 0.0f,
118                    -0.5f, -0.5f, 0.0f,
119                     0.0f,  0.5f, 0.0f,
120                     0.0f,  0.5f, 0.0f,
121                     0.5f, -0.5f, 0.0f
122                 };
123
124                 float colores[] = {
125                     1.0f, 0.0f, 0.0f,
126                     1.0f, 0.0f, 0.0f,
127                     0.0f, 1.0f, 0.0f,
128                     0.0f, 1.0f, 0.0f,
129                     0.0f, 0.0f, 1.0f,
130                     0.0f, 0.0f, 1.0f
131                 };
132
133                 //Se genera el dibujo con el arreglo de puntos
134                 objDib = genObjaDib(vertices,colores);
135
136                 //Bucle de renderizado
137                 while (!glfwWindowShouldClose(ventana)) {
138                     glClearColor(1.0f, 1.0f, 1.0f, 0.0f);//color de fondo - Blanco
139                     glClear(GL_COLOR_BUFFER_BIT); // limpiar el FrameBuffer
140                     objSomb.use();//Se llama al evento use para usar el Programa creado
141                     glBindVertexArray(objDib);
142                     glDrawArrays(GL_LINE_STRIP, 0, 6);//uso de 6 puntos (3 pares de puntos)
143                     glLineWidth(3.0f);//grosor de línea
144
145                     glfwSwapBuffers(ventana); //activar el doble búfer
146                     glfwPollEvents(); //capturar cualquier interacción del usuario con la ventana
147                 }
148                 glDeleteVertexArrays(objDib);
149                 glDeleteBuffers(objDib);
150                 glfwFreeCallbacks(ventana); //Liberar las llamadas que interactuen con la ventana
151                 glfwDestroyWindow(ventana); //Liberar memoria de la ventana creada
152             } finally {
153                 glfwTerminate(); //limpia y termina la aplicación
154             }
155         }
156     }
157
```

La parte esencial de este código, radica en el evento *"genObjaDib(float vertices[],float colores[])"*, que recibirá los parámetros de 2 arreglos (los *vértices* o coordenadas de puntos y los *colores* para cada par de puntos); *se enfatiza que: pasar parámetros de arreglos en Java es mucho fácil que gestionarlo con el dolor de cabeza de punteros en C++ (apreciación propia).*

Aquí la variable ***"VBO"***, encargada de guardar los datos de los vértices en la memoria de una tarjeta o procesador gráfico; debe ser definida 2 veces (pVBO y cVBO). Esto debido a que la aplicación gestiona el dibujo de *"puntos"* y posteriormente debe *"colorearlos"* (esto se logra gracias a la configuración declarada en *"verticeSomb.vs"* y *"fragmentoSomb.fs"*).

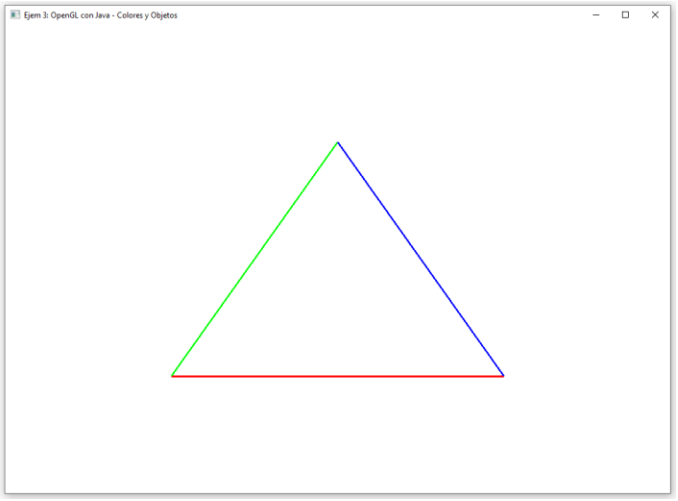
De la misma forma, se tomó la decisión de manejar arreglos separados (no muy eficaz) para los puntos y colores; puesto que al evento *glVertexAttribPointer* (explicado en VS.) se le indicará como enviar los datos, tal como se pide en compilaciones y como gestionar el arreglo donde están los datos (esto en VS. Se realizaba dosificando la memoria sizeof(), lo cual en Java; no es muy claro hacerlo), por lo tanto; se solucionaba esta distribución de memoria eficientemente.

Regresando a la programación del "Main", como ya se sabe, se procede inicializando el *INI_GLFW*, para crear la ventana; luego se instancia la clase Sombreador en *"objSomb"* para realizar las compilaciones; se declaran los puntos a dibujar (*vertices[]*) y los colores (*colores[]*) por cada par de puntos (es por eso que hay repitencia cada 3 datos – inicio y final); se llama al evento *genObjaDib* para dibujar y colorear y finalmente se declara el ciclo de renderizado.

Cabe destacar que en el ciclo se llama al método *objSomb.use()* que contiene el Programa Sombreador configurado; se vincula el VAO obtenido desde

genObjaDib mediante la variable *objDib* y se le indica que debe dibujar 6 puntos mediante líneas: *glDrawArrays(GL_LINE_STRIP, 0, 6)*; opcionalmente se configura el grosor de línea; finalmente, se libera memoria usada (demás líneas de código ya explicadas).

- El ejemplo a mostrar por la ventana sería de la siguiente forma:



- Material de apoyo que da ideas del uso de GLFW y OpenGL con Java
  - http://wiki.lwjgl.org/wiki/The_Quad_colored.html