- Proyecto que dibuja Primitivas (Líneas) mediante el Mouse y sus Transformaciones (Traslación, Rotación, Escalación) (Ejemplo 05):

  - Se creará el proyecto jOpenGLEjem5

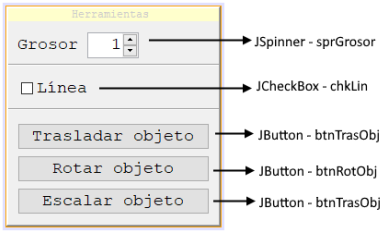
  - Se agregarán las librerías GLFW y OpenGL como también a GLM para Java (v1.0.1)

    - *OpenGL Mathematics (GLM)* es una biblioteca matemática para C++ como encabezado, solo para software de gráficos; basada en las especificaciones del lenguaje de sombreado OpenGL (GLSL). Este proyecto no se limita a las funciones GLSL, es un sistema basado en las convenciones de extensión GLSL, proporciona capacidades ampliadas como transformaciones matriciales, cuaterniones, empaquetamiento de datos, números aleatorios, ruido, etc.
    - Esta librería fue inspirada para adaptarla también al lenguaje Java, creando a ***"JGLM"*** por la *Java Graphics Society*, actualizada hasta octubre del 2016.
    - Para descargar la librería, está se encuentra en el paquete: ***jar_files.zip***; dentro de este se encuentra el archivo: ***glm-1.0.1.jar***.
      - https://jar-download.com/artifacts/io.github.java-graphics/glm/1.0.1

  - Se creará una carpeta de nombre "clases" dentro del proyecto, ahí deben ir las clases y archivos:

    - vcd.java
    - clsGlfwConSomb.java (modificado)
    - fragmentoSomb.fs
    - verticeSomb.vs (esta tomará ahora el nombre de: verticeSombMod.vs)
    - clsCoordMouse.java (esta tomará ahora el nombre de: clsGestMouse.java)

  - La ventana de Herramientas (*clsVenDeHerr.java*), tendrá cambios agregando elementos Spinner, CheckBox y Buttons como también la declaración de sus variables que interactúen con las transformaciones:




```
6   package jopenglejem5;
7
8   /**
9    *
10   * @author PC-MAX
11   */
12  public class clsVenDeHerr extends javax.swing.JFrame {
13
14      /**
15       * Creates new form clsVenDeHerr
16       */
17      public int posVenX, posVenY;
18      public float valGrosor; //valor del Grosor
19      public int valPrim = 0; //valor de la primitiva
20      public float vTrasX = 0.0f; //valor de incremento para trasladar en eje X;
21      public float vTrasY = 0.0f; //valor de incremento para trasladar en eje Y;
22      public float vRot = 0.0f; //valor angular de rotación;
23      public float vEsc = 1.0f; //valor de escalación;
24
25      public clsVenDeHerr() {
26          initComponents();
            setLocation(210, 155); //pos. temporal... !!!mejorar de forma dinámica
28      }
```

Es importante tener en cuenta la inicialización de la variable *vEsc = 1.0f* (según el análisis de esta variable que se encarga de escalar un objeto, este debe comenzar en la unidad mayor del plano (-1.0 ; 1.0)

```
    private void lblZonMovMousePressed(java.awt.event.MouseEvent evt) {
        //Se obtiene la coord. x;y al presionar botones del mouse
        posVenX = evt.getX();
        posVenY = evt.getY();
    }

    private void lblZonMovMouseDragged(java.awt.event.MouseEvent evt) {
        //Se ejecuta el desplazamiento de la Ventana según las coord. del mouse
        this.setLocation(this.getLocation().x + evt.getX() - posVenX, this.getLocation().y + evt.getY() - posVenY);
    }

    private void sprGrosorStateChanged(javax.swing.event.ChangeEvent evt) {
        //Se obtiene el valor de tipo entero a tipo Float en la variable valGrosor (grosor de línea)
        valGrosor = ((Integer)sprGrosor.getValue()).floatValue();
    }

    private void chkLinActionPerformed(java.awt.event.ActionEvent evt) {
        if(chkLin.isSelected()){
            valPrim = 1;
        }else{
            valPrim = 0;
        }
    }

    private void btnTrasObjActionPerformed(java.awt.event.ActionEvent evt) {
        vTrasX += 0.05f;
        vTrasY += 0.05f;
    }

    private void btnRotObjActionPerformed(java.awt.event.ActionEvent evt) {
        vRot += 0.05f;
    }

    private void btnEscObjActionPerformed(java.awt.event.ActionEvent evt) {
        vEsc += 0.05f;
    }
```

Para los demás eventos *ActionPerformed*, se muestra el incremento en 0.05f como medio de demostración de las trasformaciones.

- Programando la ventana principal
  - Para la ventana principal se debe tener en cuenta a la clase *clsGlfwConSomb.java:*

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package clases;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.net.URISyntaxException;
import java.net.URL;
import java.nio.FloatBuffer;
import java.util.ArrayList;
import java.util.List;
import org.lwjgl.*;
import org.lwjgl.glfw.*;
import org.lwjgl.opengl.*;
import jopenglejem5.clsVenDeHerr;

import static org.lwjgl.glfw.GLFW.*;
import static org.lwjgl.opengl.GL30.*;
import static org.lwjgl.system.MemoryUtil.*;

/**
 *
 * @author PC-MAX
 */
public class clsGlfwConSomb {
    public static List<Float> alVertices = new ArrayList<>();
    public static List<Float> alColores = new ArrayList<>();
    public static int ID;
    private static long ventana;
    private static clsVenDeHerr objVDH;

    //Constructor
    public clsGlfwConSomb(clsVenDeHerr ven1){
        objVDH = ven1; //obtener la ventana clsVenDeHerr para gestionar sus métodos
    }

    private static vcd ini_GLFW(long ven, int ancho, int alto, String titulo){
        vcd objVCD = new vcd();
        clsGestMouse objCM = new clsGestMouse(); //Clase con eventos a interactuar con el Mouse

```

- El objetivo aquí es manejar arreglos de listas de datos de nombres: *alVertices* y *alColores* (por ahora el ejercicio le da exclusividad es con *alVertices*) esto debido a que se debe capturar las coordenadas del Mouse y estas, puedan interactuar en los diferentes métodos de la clase. Por otra parte, en el constructor de la clase; se llama a la ventana *clsVenDeHerr* guardada en la variable *objVDH*, para rescatar los valores de "grosor", y valores de las "trasformaciones" en tiempo real.

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

        //Repintado de la ventana
        glfwSetFramebufferSizeCallback(ven, (objVentana, an, al)->{
            glViewport(0, 0, an, al);
        });

        //Gestión del teclado
        glfwSetKeyCallback(ven, (objVentana, tecla, scancode, accion, mods) -> {
            if (tecla == GLFW_KEY_ESCAPE && accion == GLFW_RELEASE) {
                glfwSetWindowShouldClose(objVentana, true);
                objVDH.dispose(); //cerrar clase venDeHerr
            }
        });

        //Gestión del Mouse
        glfwSetCursorPosCallback(ven, (long objVentana, double xPos, double yPos)->{
            /*xPos y yPos son variables enfrascadas en glfwSetCursorPosCallback, por lo que en esta
             sección sería dificil enviar los valores de xPos y yPos hacia variables locales del Main
            declarando la clase en el Main que encapsule a xPos y yPos y devuelva sus valores.
            */
            int objDib = 0;
            objCM.setX(xPos);
            objCM.setY(yPos);
            switch (objVDH.valPrim) {
                case 1: //primitiva línea
                    if(objCM.getEsPosIni() == 2){
                        //Borrando Punto y color anterior
                        alVertices.remove(alVertices.size()-1); //borra coord x
                        alVertices.remove(alVertices.size()-1); //borra coord y
                        alVertices.remove(alVertices.size()-1); //borra coord z
                        alColores.remove(alColores.size()-1); //borra color R
                        alColores.remove(alColores.size()-1); //borra color G
                        alColores.remove(alColores.size()-1); //borra color B

                        //Agregando Punto y color actual
                        alVertices.add(cPAV(xPos,ancho,true));
                        alVertices.add(cPAV(yPos,alto,false));
                        alVertices.add(0.0f); //pos Z no utilizada pero obligatoriamente declarada
                        alColores.add(0.0f); //R
                        alColores.add(0.0f); //G
                        alColores.add(1.0f); //B
                    }else if(objCM.getEsPosIni() == 1){
                        objCM.setEsPosIni(2);
                        //Agregando Punto y color actual
                        alVertices.add(cPAV(xPos,ancho,true));
                        alVertices.add(cPAV(yPos,alto,false));
                        alVertices.add(0.0f); //pos Z no utilizada pero obligatoriamente declarada
                        alColores.add(0.0f); //R
                        alColores.add(0.0f); //G
                        alColores.add(1.0f); //B
                    }
                    break;
            }
        });

        glfwSetMouseButtonCallback(ven,(long objVentana, int boton, int accion, int mod)->{
            if(boton == GLFW_MOUSE_BUTTON_LEFT && accion == GLFW_RELEASE){
                //Evaluar Switch de primitivas
                switch (objVDH.valPrim) {
                    case 1: //primitiva línea
                        //Agregando Punto y color actual
                        if(objCM.getEsPosIni() == 2){
                            objCM.setEsPosIni(0); //se dejo de dibujar el punto par
                        }else{//se ingresa el punto inicial impar
                            alVertices.add(cPAV(objCM.getX(),ancho,true));
                            alVertices.add(cPAV(objCM.getY(),alto,false));
                            alVertices.add(0.0f); //pos Z no utilizada pero obligatoriamente declarada
                            alColores.add(0.0f); //R
                            alColores.add(0.0f); //G
                            alColores.add(1.0f); //B
                            objCM.setEsPosIni(1);
                        }
                        break;
                }
            }
        });
```

```
133
134         GLFWVidMode vidmode = glfwGetVideoMode(glfwGetPrimaryMonitor());
            glfwSetWindowPos(ven,(vidmode.width() - ancho) / 2,(vidmode.height() - alto) / 2);
136
137         glfwMakeContextCurrent(ven);
138         glfwSwapInterval(1);
139
140         GL.createCapabilities();
141
142         objVCD.setVentana(ven);
143         return objVCD;
144     }
```

Aquí se usa un CallBack llamado *SetCursorPos*, encargado de capturar las coordenadas a la hora de mover el Mouse, por lo que al combinarlo con *MouseButton* lo que se logra es: cada vez que se tenga activo el checkBox de la primitiva "Línea", al hacer clic sobre el Mouse se captura el primer par de puntos, luego; al ir moviendo el Mouse, se dibuja en tiempo real la línea, para que al volver hacer clic sobre el Mouse se finalice el dibujo de la línea. Estas coordenadas se guardan en el *alVertices.* Se eligió usar una sentencia "*Switch*" con el objetivo de escalar código (por ejemplo, dibujo de otras primitivas).

- Los archivos de compilación para GLSL, cambian ligeramente:

verticeSombMod.vs

```
1  #version 330 core
2  layout (location = 0) in vec3 aPos;
3  layout (location = 1) in vec3 aColor;
4
5  uniform mat4 transform;
6  out vec3 ourColor;
7
8  void main()
9  {
10     gl_Position = transform * vec4(aPos,1.0f);
11     ourColor = aColor;
12 }
```

fragmentoSomb.fs

```
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
```

Se aprecia que *verticeSombMod.vs* tiene una nueva variable de nombre "*transform*" y lleva el tipo *mat4*; este tipo es una matriz de 4x4 que es necesaria para realizar las transformaciones de traslación, rotación y escalado, pero estas deben ser evaluadas en la *compilación GLSL;* es por ello, que se usa la característica "*uniform*" el cual uniformizará las operaciones vectoriales para cada conjunto de puntos que dibujen una primitiva; cuyos valores se configuran en el "*Main*" principal.

- Con respecto al evento *Sombreador*; al ver que se creó el Package: "clases", la modificación a realizar descansa en la llamada mediante parámetros de los archivos *fragmentoSombMod.fs* y *verticeSomb.vs*, debido a que se usará la forma relativa "*class.getResource*" encargada de buscar dichos archivos dentro de este paquete, como también agregar su controlador de excepciones "*URISyntaxException ex*":

```
145
146     private static void Sombreador(String archVertice, String archFragmento, boolean relleno){
147         //1. recuperar el código fuente de vértice y fragmento desde el directorio raíz
            String cadVertice = "", cadFragmento = "", cadLin = "";
            File arch = null;
150         FileReader fr = null;
            BufferedReader br = null;
            URL url = null;
153         try{
154             url = clsGlfwConSomb.class.getResource(archVertice); //obtengo la raiz relativa del paquete clases
155             arch = new File(url.toURI()); //verifico que la ruta entienda caracteres esp (esp. tildes, etc...)
156             fr = new FileReader (arch.getAbsolutePath());
157             br = new BufferedReader(fr);
158             while ((cadLin = br.readLine())!=null) {
159                 cadVertice =  cadVertice + cadLin + "\n";
160             }
                cadLin = "";
162             url = clsGlfwConSomb.class.getResource(archFragmento); //obtengo la raiz relativa del paquete clases
163             arch = new File(url.toURI()); //verifico que la ruta entienda caracteres esp (esp. tildes, etc...)
164             fr = new FileReader (arch.getAbsolutePath());
165             br = new BufferedReader(fr);
```

```
            while ((cadLin = br.readLine())!=null) {
                cadFragmento =  cadFragmento + cadLin + "\n";
            }
        }catch(IOException | URISyntaxException ex){
            System.out.println("Error al leer archivo de compilación:\n"+
                               "Descripción: "+ex+"\n"+
                               "Verificar!!!");
            ex.printStackTrace();
            System.exit(1);
        }finally{
            try{
                if( null != fr ){
                    fr.close();
                }
            }catch (Exception ex2){
                ex2.printStackTrace();
            }
        }

        //2. Compilar los Sombreadores y el programa
        int vertexShader = glCreateShader(GL_VERTEX_SHADER);
        glShaderSource(vertexShader,cadVertice);
        glCompileShader(vertexShader);
        verifErrorComp(vertexShader,"VERTICE");

        int fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
        glShaderSource(fragmentShader,cadFragmento);
        glCompileShader(fragmentShader);
        verifErrorComp(fragmentShader,"FRAGMENTO");

        ID = glCreateProgram();
        glAttachShader(ID, vertexShader);
        glAttachShader(ID, fragmentShader);
        glLinkProgram(ID);
        verifErrorComp(ID,"PROGRAMA");

        glDeleteShader(vertexShader);
        glDeleteShader(fragmentShader);

        if(!relleno){glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);}
    }

    private static void verifErrorComp(int shader, String tipo){
        int success, infoLogLength;
        if(!"PROGRAMA".equals(tipo)){
            success =  glGetShaderi(shader, GL_COMPILE_STATUS);
            if(success == GL_FALSE){
                infoLogLength = glGetShaderi(shader, GL_INFO_LOG_LENGTH);
                System.out.println("Error al compilar el: "+tipo+"\n"+
                                   "Descripción: "+glGetShaderInfoLog(shader,infoLogLength)+"\n"+
                                   "Verificar!!!");
            }
        }else{
            success =  glGetProgrami(shader, GL_LINK_STATUS);
            if(success == GL_FALSE){
                infoLogLength = glGetProgrami(shader, GL_INFO_LOG_LENGTH);
                System.out.println("Error al vincular los Sombreadores:\n"+
                                   "Descripción: "+glGetProgramInfoLog(shader,infoLogLength)+"\n"+
                                   "Verificar!!!");
            }
        }
    }
```

Cabe recalcar que se eliminó el evento *use()*, puesto que el código dentro de este debe interactuar en tiempo real mediante el "*Main*" principal.

- El evento *genObjADibMod* permanece igual que lo explicado en el anterior ejemplo (Nro. 04), con la única excepción de que se retira la línea de código: *glLineWidth(objVDH.valGrosor);//grosor de línea*, para agregarla en el "*Main*" principal, como medida de interacción entre las ventanas de "*Herramientas y Principal*" en tiempo real.

- De la misma forma, los eventos *ini_ventana* y *cPAV*, permanecen igual que el ejemplo anterior (Nro. 04) a excepción de que el 1er. parámetro del evento *Sombreador* será: "*verticeSombMod.vs*"

- Finalmente se programa el "*Main*" principal:

```
public static void main(String[] args) {
    final int _Ancho = 1024;
    final int _Alto = 720;
    long ventana = NULL;
    clsVenDeHerr objVDH = new clsVenDeHerr();
    clsGlfwConSomb objGCS = new clsGlfwConSomb(objVDH);
    int objDib;

    objVDH.setVisible(true);
    try {
        ventana = objGCS.ini_ventana(_Ancho, _Alto, "Ejem 5: OpenGL con Java - Primitivas + Transformaciones, usando el Mouse"
        if(ventana != NULL){
            glClearColor(1.0f, 1.0f, 1.0f, 0.0f);//configurar color de fondo - Blanco
```

```
            //Bucle de renderizado
            while (!glfwWindowShouldClose(ventana)) {
                glClear(GL_COLOR_BUFFER_BIT); // limpiar el FrameBuffer
                objDib = objGCS.genObjADibMod(objGCS.alVertices,objGCS.alColores);
                glUseProgram(objGCS.ID);
                glBindVertexArray(objDib); //vincular el objeto a dibujar con su Arreglo de vértices
                if((objGCS.alVertices.size()/3)%2 == 0){//verificar si lleva siempre par de puntos (de 2 en 2)
                    glDrawArrays(GL_LINES, 0,objGCS.alVertices.size()/3);//uso de puntos (vértices basados en 3 coord)

                    //Transformación de traslación
                    Mat4 tras = new Mat4(1.0f);
                    tras = tras.translate(new Vec3(objVDH.vTrasX, objVDH.vTrasY, 0.0f));

                    //Transformación de rotación
                    Mat4 rota = new Mat4(1.0f);
                    //trans = trans.rotate(-1*objVDH.vRot, new Vec3(0.0f,0.0f,1.0f)); //rotación sobre eje Z (horario)
                    rota = rota.rotate(objVDH.vRot, new Vec3(0.0f,0.0f,1.0f)); //rotación sobre eje Z (antihorario)

                    //Transformación de Escalación
                    Mat4 esca = new Mat4(1.0f);
                    esca = esca.scale(objVDH.vEsc,objVDH.vEsc,0.0f); //OJO!!! vEsc debe valer 1.0f = tamaño original

                    //Llevar las transformaciones al área de dibujo
                    int transformLoc = glGetUniformLocation(objGCS.ID, "transform");
                    Mat4 trans = tras.mul(rota).mul(esca); //Unir la traslación, rotación y Escalación (vectorialmente)
                    if (transformLoc > -1) {// ya declarado
                        FloatBuffer MatrizTrans = BufferUtils.createFloatBuffer(trans.toFa_().length);
                        MatrizTrans.put(trans.toFa_());
                        MatrizTrans.flip();
                        glUniformMatrix4fv(transformLoc, false, MatrizTrans);
                    }
                }
                glfwSwapBuffers(ventana); //activar el doble búfer
                glDeleteVertexArrays(objDib);
                glDeleteBuffers(objDib);
                glLineWidth(objVDH.valGrosor);//grosor de línea
                glfwPollEvents(); //capturar cualquier interacción del usuario con la ventana
            }
            glfwFreeCallbacks(ventana); //Liberar las llamadas que interactuen con la ventana
            glfwDestroyWindow(ventana); //Liberar memoria de la ventana creada
        }
    } finally {
        glfwTerminate(); //limpia y termina la aplicación
    }
}
```

Lo interesante del bloque *While* es que usa el Programa sombreador (*objGCS.ID*) con el objeto de dibujo (*objDib*) con datos vacíos como forma de inicialización, luego hay un *if* que determina si existen un par de puntos listos para dibujar (usando el artificio matemático para comparar el residuo de ingreso par).

Dentro de este *if* se dibuja la línea y se procede a realizar 3 tipos de trasformaciones:

- Transformación de traslación: Usando el tipo de dato *Mat4* de la librería *glm*, se inicializa la *"matriz identidad"(esto por teoría de vectores de trasformación gráfica)* y se guardar en la variable *"tras"* el evento *tras.translate* cuyo parámetro es un vector de 3 elementos (x,y,z); con ello, se consigue el desplazamiento del dibujo (en este caso de la línea) cuyos datos se obtienen de las variables de la ventana *clsVenDeHerr*, botón "Trasladar Objeto". Aquí, para demostración; siempre se hará un desplazamiento diagonal derecho superior.

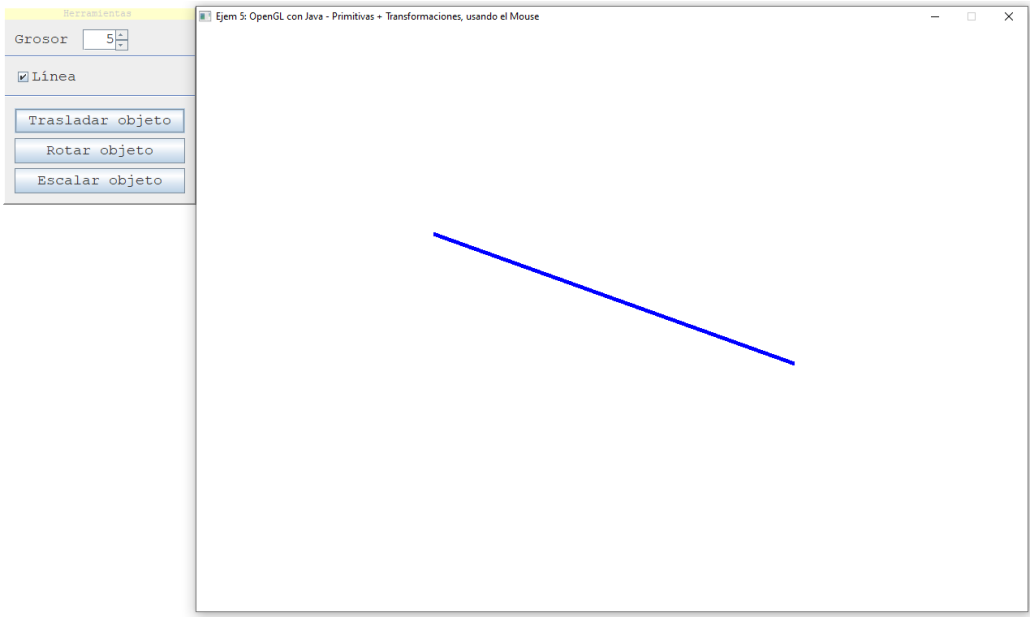
- Transformación de rotación: tiene la misma funcionalidad que el evento tras.translate pero se guarda el resultado en la variable *"rota"*, para este evento: *rota.rotate*, su primer parámetro es un valor de tipo *float* que indicará la rotación (si es negativo es rotación horaria y si es positivo es rotación antihoraria) el otro parámetro es un vector de 3 elementos (x,y,z); indicando en que eje debe realizarse la rotación (por defecto se indica el eje *"Z"*, 1.0f). El botón "Rotar objeto" realizará la rotación en sentido antihorario.

- Transformación de escalado: tiene similar funcionalidad que los otros eventos; *esca.scale* lleva parámetros de los 3 ejes (en este caso solo se toma el eje "X" y se repite este valor en el eje "Y", no se toca el eje "Z") y se guarda el resultado en la variable *"esca"*. Aca, es

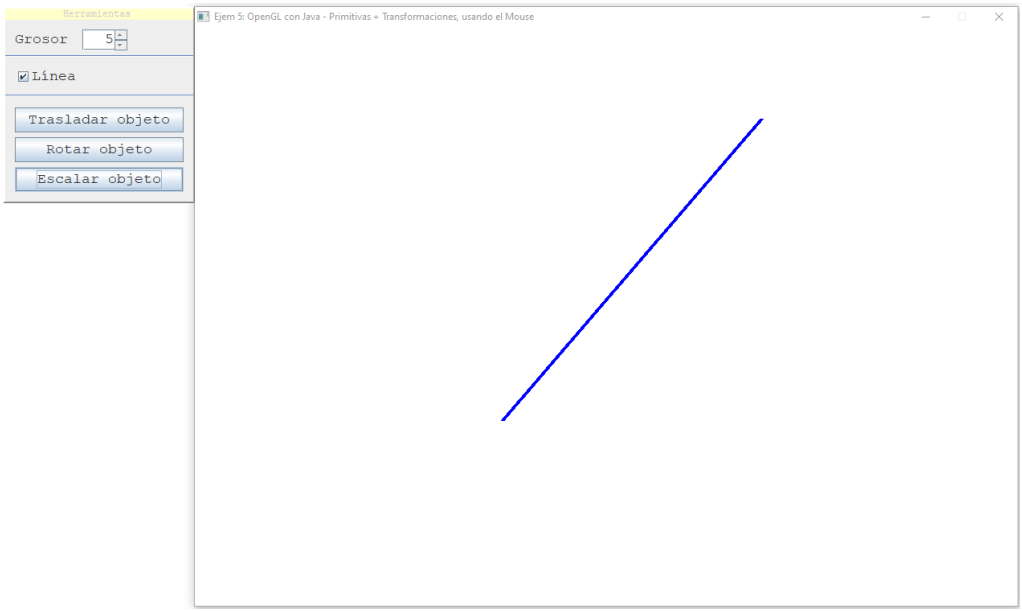
muy importante tener presente el valor de *objVDH.vEsc* puesto que debe comenzar con 1.0f como medida de plano original, ya que si es *0.0f* no hay escalación. El botón "Escalar objeto" se incrementará como forma de demostración.

- Por último, se debe volcar toda esta información en la variable uniforme "*transform*" que está declarada en "*verticeSombMod.vs*"; cuyo resultado se guarda en "*transformLoc*", pero al ver que se ha realizado 3 transformaciones, cada resultado se debe ir multiplicando vectorialmente para tener el resultado simultáneo, es por ello que, se declara la variable *Mat4 trans = tras.mul(rota).mul(esca);* usando sus eventos respectivamente; esto logra una matriz búfer de nombre "*MatrizTrans*" que se vuelca a "*transform*".

o El programa tiene como resultado:



Se procede a presionar los botones de trasformación:



- Material de apoyo
  - https://glm.g-truc.net/0.9.9/index.html (Definición de GLM)
  - https://github.com/java-graphics/glm (GLM para Java - Modelo teórico)
  - http://forum.lwjgl.org/index.php?topic=5686.0 (manejo del búfers basado en matrices)
  - https://learnopengl.com/Getting-started/Transformations (teoría de vectores)
  - https://community.khronos.org/t/rotating-object-after-transformation-in-modern-opengl/75620 (Rotación de objetos después de transformaciones)
  - https://stackoverflow.com/questions/29187765/lwjgl-3-gluniformmatrix4-causes-jre-to-crash (Entendiendo los tipos Uniform)
  - https://stackoverflow.com/questions/3844307/how-to-read-file-from-relative-path-in-java-project-java-io-file-cannot-find-th (manejo de rutas relativas)