APUNTES, MANUALES, PRESENTACIONES

Universidad de Jaén

Escuela Politécnica Superior de Jaén

Manual complementario de prácticas de informática gráfica y visualización

Alfonso López Ruiz

2023-2024

Informática gráfica y visualización

Universidad de Jaén

Departamento de Informática

EXCELENCIA EUROPEA 500+ EFQM

EFQM
Recognised for Excellence
5 Star - 2017

Manual complementario de prácticas

Informática Gráfica y Visualización

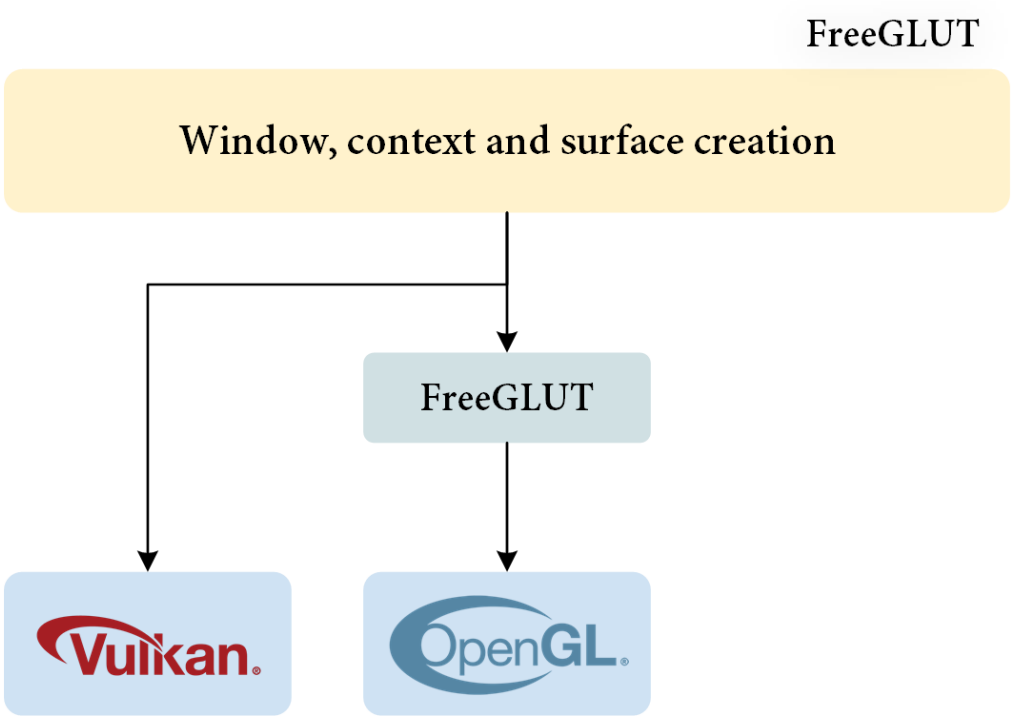
Grado en Ingeniería Informática

Alfonso López Ruiz

Elaborado en el curso académico 2023-2024

Entorno de desarrollo

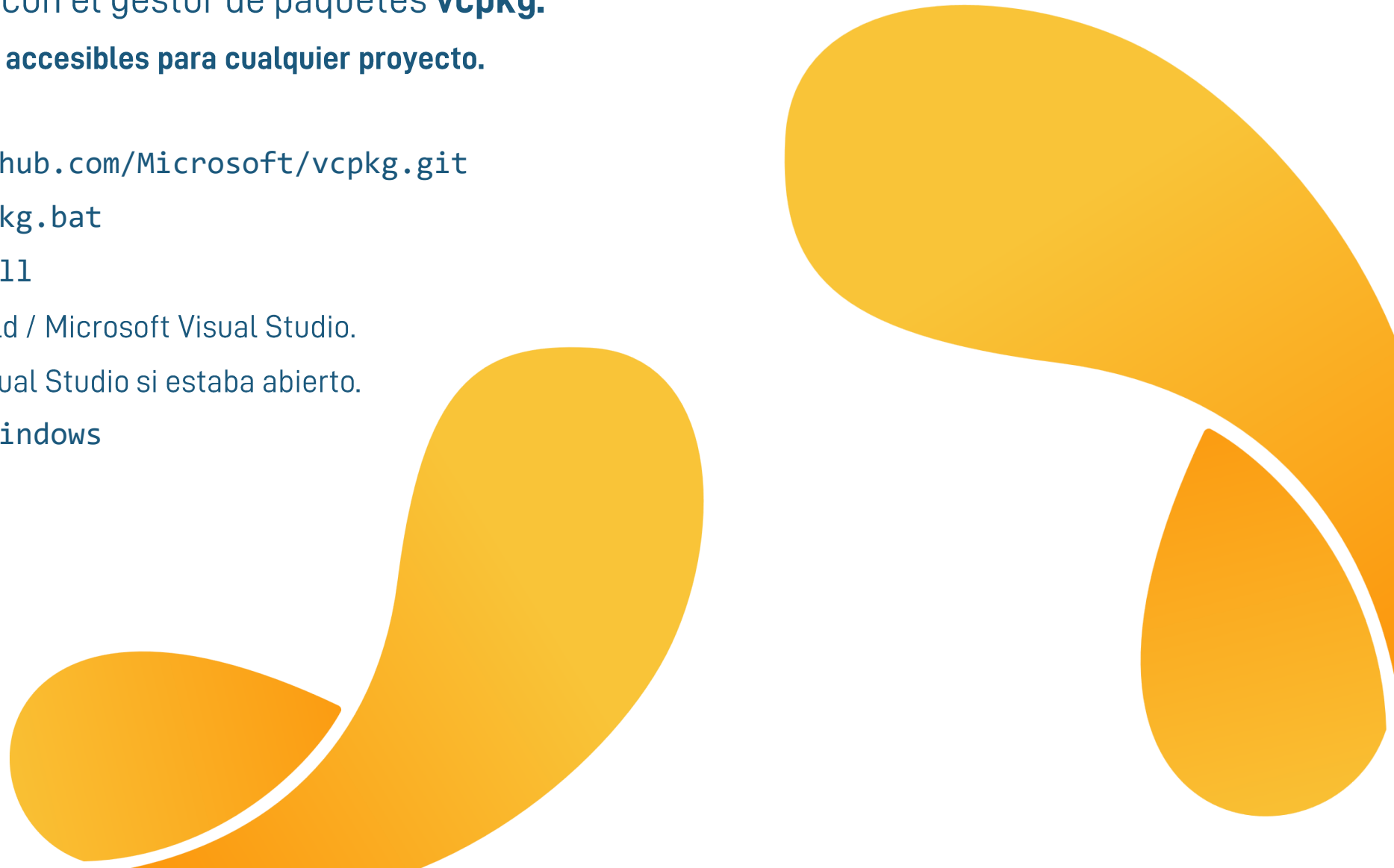
- Utilizaremos el **lenguaje C++** junto con la interfaz gráfica **OpenGL.**
- **FreeGLUT** como interfaz de acceso a funciones OpenGL**.**
- **Herramientas de desarrollo:** Microsoft Visual Studio / CLion**.**



Entorno de desarrollo: Microsoft Visual Studio

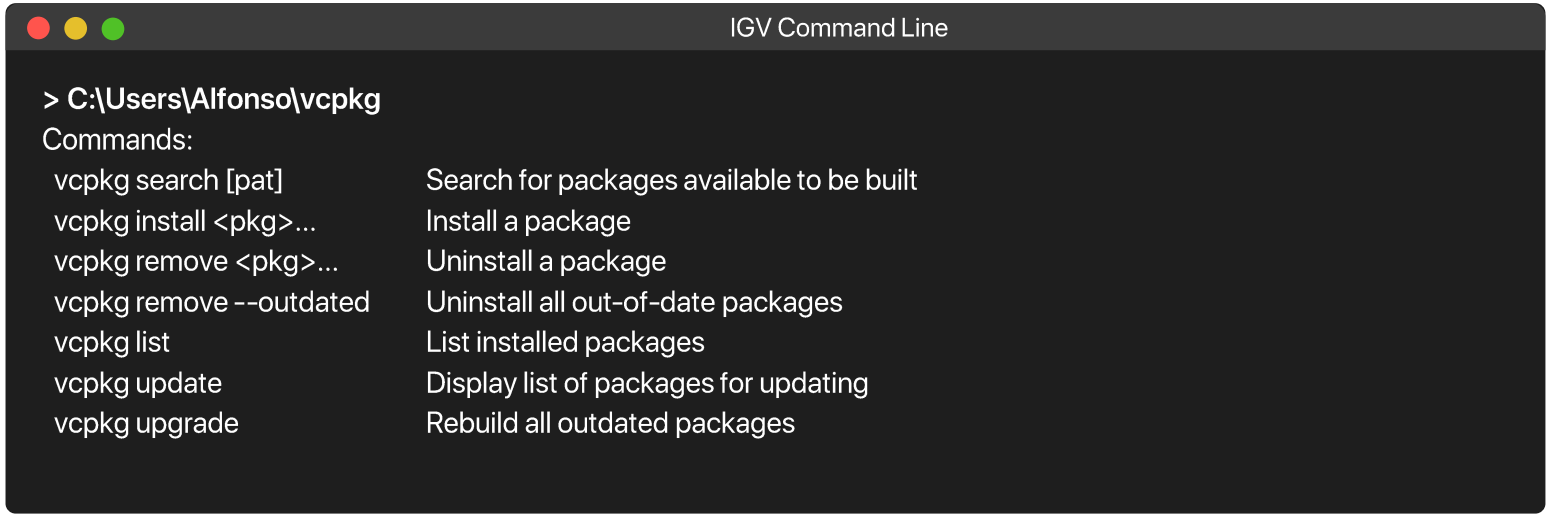
- Se puede utilizar junto con el gestor de paquetes **vcpkg.**
  - **Paquetes globalmente accesibles para cualquier proyecto.**

1. `git clone https://github.com/Microsoft/vcpkg.git`
2. `.\vcpkg\bootstrap-vcpkg.bat`
3. `vcpkg integrate install`
   - Integración con MSBuild / Microsoft Visual Studio.
   - Reiniciad Microsoft Visual Studio si estaba abierto.
4. `vcpkg install x:x64-windows`



Entorno de desarrollo: Microsoft Visual Studio

- Por comodidad:
  - Enlazad vcpkg al **Path** del sistema.
    - Una nueva línea en 'Path' con la ruta de vcpkg.
  - Definid la **arquitectura del sistema** (y de las librerías que buscamos).
    - x64, x86 (normalmente es la arquitectura de 64 bits).
    - Nueva variable de entorno: `VCPKG_DEFAULT_TRIPLET: x64-windows`

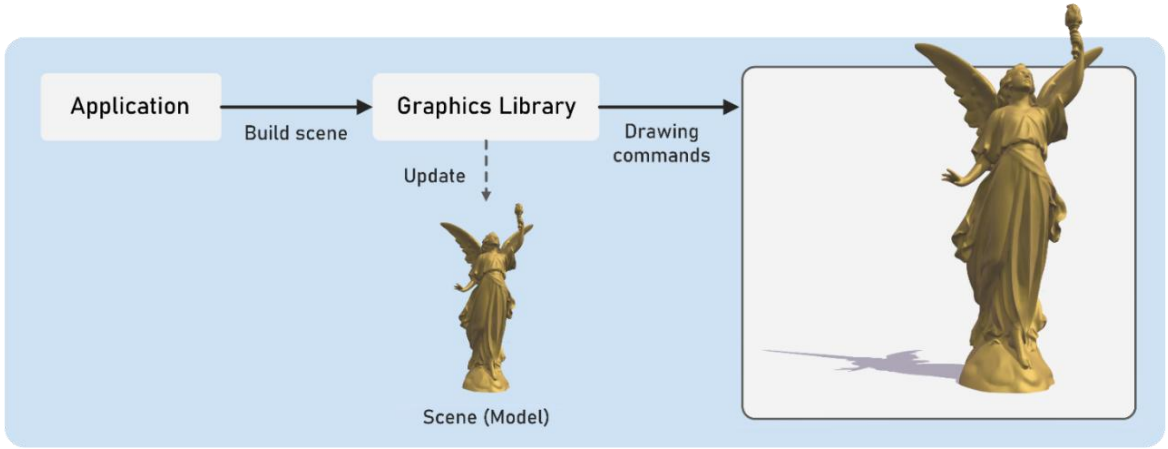


Entorno de desarrollo: CLion

- **CLion + vcpkg [necesita al menos Visual Studio 2015] + CMake**
  - *https://blog.jetbrains.com/clion/2023/01/support-for-vcpkg-in-clion/*
- **CLion + Conan + CMake**
  - Plugin de Conan disponible en el entorno de desarrollo.

OpenGL

- **OpenGL no es una biblioteca de clases, es una biblioteca de funciones.**
- **Funciona como una máquina de estados.**
- **Opera en modo inmediato.**
- **La biblioteca gráfica OpenGL incluye funciones para:**
  - Definir primitivas gráficas y sus atributos.
  - Especificar transformaciones geométricas.
  - Definir los parámetros de visualización.
  - Definir luces, propiedades de materiales, interacción con los modelos, etc.



Funciones de OpenGL

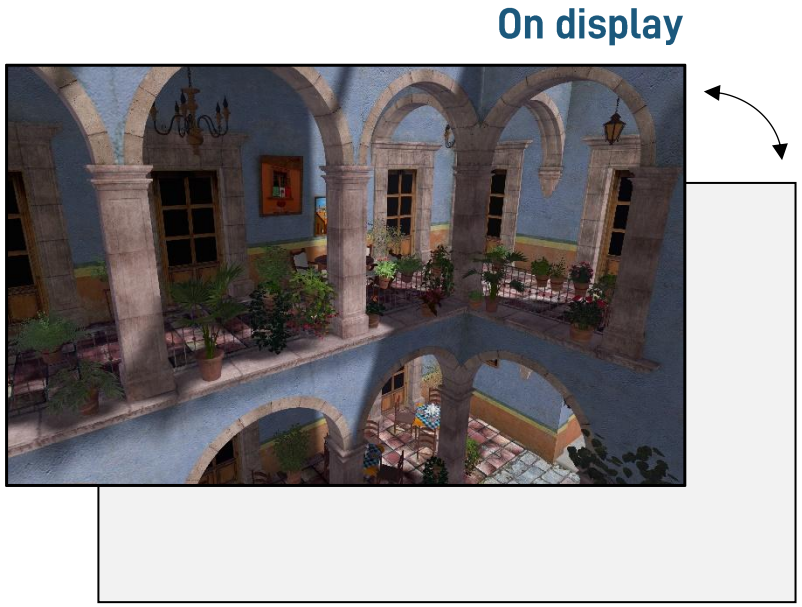
- **El nombre de las funciones lleva el prefijo de la biblioteca que las implementa:**
  - `OpenGL core library`: **gl**, por ejemplo `glTranslate`(...);
  - `OpenGL utility`: **glu**, por ejemplo `gluLookAt`(...);
  - `OpenGL utility toolkit`: **glut**, por ejemplo `glutCreateWindow`(...);
- **Muchos parámetros hay que especificarlos utilizando constantes de su biblioteca correspondiente:**
  - `GL, GLU o GLUT: GL_PROJECTION, GLUT_DEPTH`...

Manejo de ventanas

- **OpenGL es independiente del dispositivo. Utilizamos el toolkit GLUT (GL Utility Toolkit):**
  - `glutInit(int *argc, char **argv)`: inicializa las estructuras de GLUT.
  - `glutInitDisplayMode(unsigned int mode)`: especifica el modo de color y los tipos de *buffers* a utilizar para la visualización:
    - **color**: GLUT_RGBA, GLUT_INDEX
      - `glClearColor(r,g,b,a)`: color de fondo de la ventana.
      - `glColor3f(r,g,b)`: color del objeto.

Manejo de ventanas

- **OpenGL es independiente del dispositivo. Utilizamos el toolkit GLUT (GL Utility Toolkit):**
  - `glutInit(int *argc, char **argv)`: inicializa las estructuras de GLUT.
  - `glutInitDisplayMode(unsigned int mode)`: especifica el modo de color y los tipos de *buffers* a utilizar para la visualización:
    - **Framebuffer**: GLUT_SINGLE, GLUT_DOUBLE
      - `glFlush()`: fuerza procesar las instrucciones de visualización.
      - **`glutSwapBuffers()`**: intercambia los *buffers*.



Manejo de ventanas

- **OpenGL es independiente del dispositivo. Utilizamos el toolkit GLUT (GL Utility Toolkit):**
  - `glutInit(int *argc, char **argv)`: inicializa las estructuras de GLUT.
  - `glutInitDisplayMode(unsigned int mode)`: especifica el modo de color y los tipos de buffers a utilizar para la visualización:
    - **z-buffer**: GLUT_DEPTH
      - `glEnable(GL_DEPTH_TEST)`: activa el z-buffer.

Manejo de ventanas

- **OpenGL es independiente del dispositivo. Utilizamos el toolkit GLUT (GL Utility Toolkit):**
  - `glutInit(int *argc, char **argv)`: inicializa las estructuras de GLUT.
  - `glutInitDisplayMode(unsigned int mode)`: especifica el modo de color y los tipos de buffers a utilizar para la visualización:
    - **z-buffer**: GLUT_DEPTH
      - `glEnable(GL_DEPTH_TEST)`: activa el z-buffer.

Manejo de ventanas

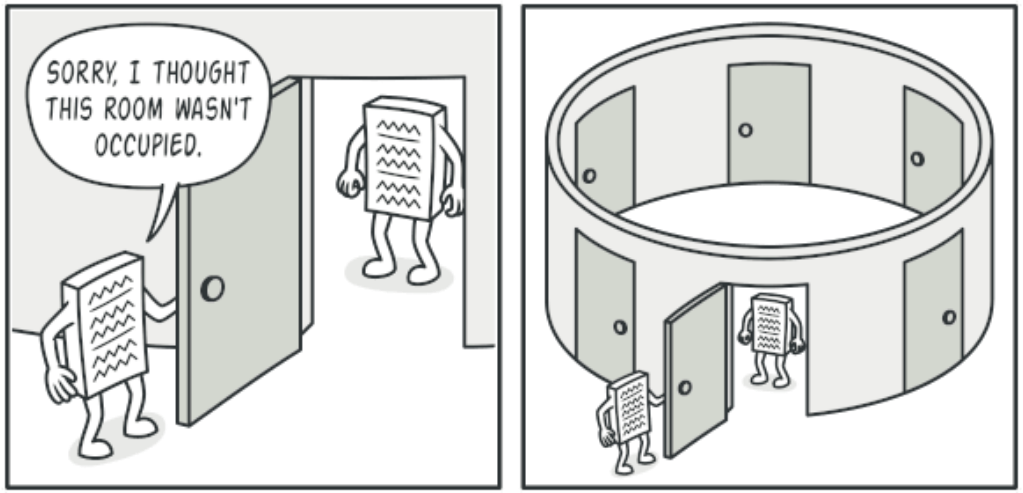
- **OpenGL es independiente del dispositivo. Utilizamos el toolkit GLUT (GL Utility Toolkit):**
  - `glutInitWindowPosition(int x, int y)`: posición de la esquina superior izquierda de la ventana, relativa a la esquina superior izquierda de la pantalla.
  - `glutInitWindowSize(int ancho, int alto)`: indica el tamaño en píxeles de la ventana.
  - `glutCreateWindow(char *title)`: crea la ventana.
  - `glutDisplayFunc(void (*func)(void))`: *func* es una función donde se define la escena que se va a visualizar.
  - `glutMainLoop()`: inicia el bucle de visualización.

Funciones de manejo de ventanas y callbacks

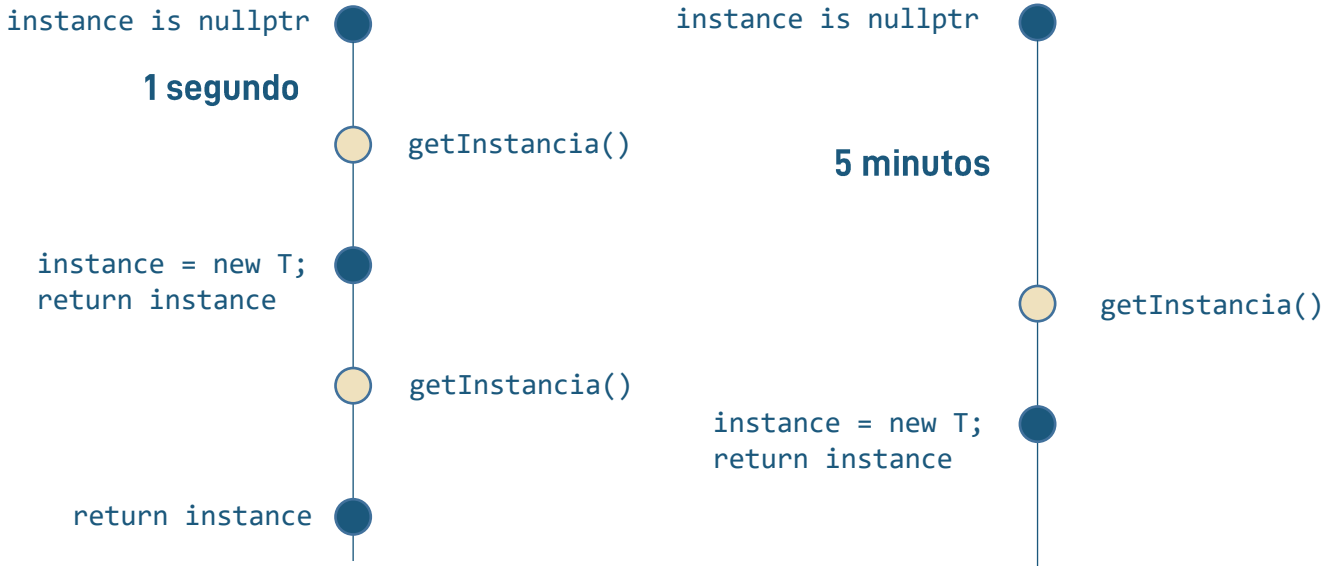

```
#include "igvInterfaz.h“

int main (int argc, char** argv)  {  // Inicializa la ventana de visualización
    igvInterfaz::getInstancia().configura_entorno(
                          argc, argv,
                          500, 500 // Tamaño de la ventana,
                          100, 100 // Posición de la ventana,
                          "CG&V. Practice 1");
    // Establece las funciones callback para la gestión de eventos
    igvInterfaz::getInstancia().inicializa_callbacks();
    // Inicia el bucle de visualización de GLUT
    igvInterfaz::getInstancia().inicia_bucle_visualizacion();

    return(0);
}
```

Patrón “Singleton”

- **Construcción mediante `getInstancia`.**
- **Todas las clases que implementen este patrón tienen una estructura en común:**
  - Variable estática.
  - Método *`getInstance`*.
  - Inicialización perezosa: se crea la instancia cuando se pide.

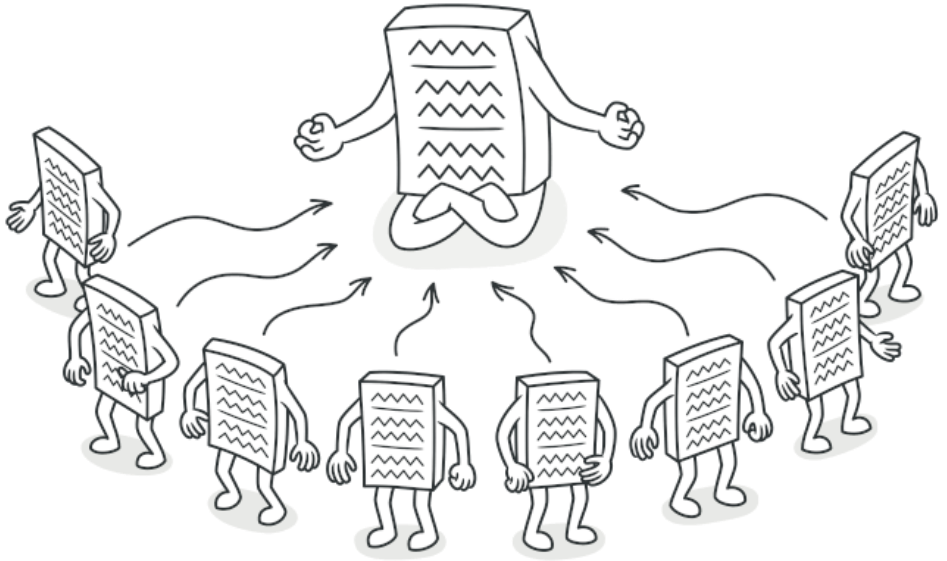


[Dive into Design Patterns]



Patrón “Singleton”

- **Instancia globalmente accesible.**
  - Características:
    - **Construcción dependiente de un método `getInstancia`.**
      - Comprueba si ya existe una instancia creada.
      - Si no es así, se construye.
    - **Construcción y borrado no accesible de manera pública.**
      - Si fuera público, no se podría garantizar la unicidad.
    - **Variable estática**: almacena la instancia única. Es un puntero nulo por defecto.



Dive into Design Patterns

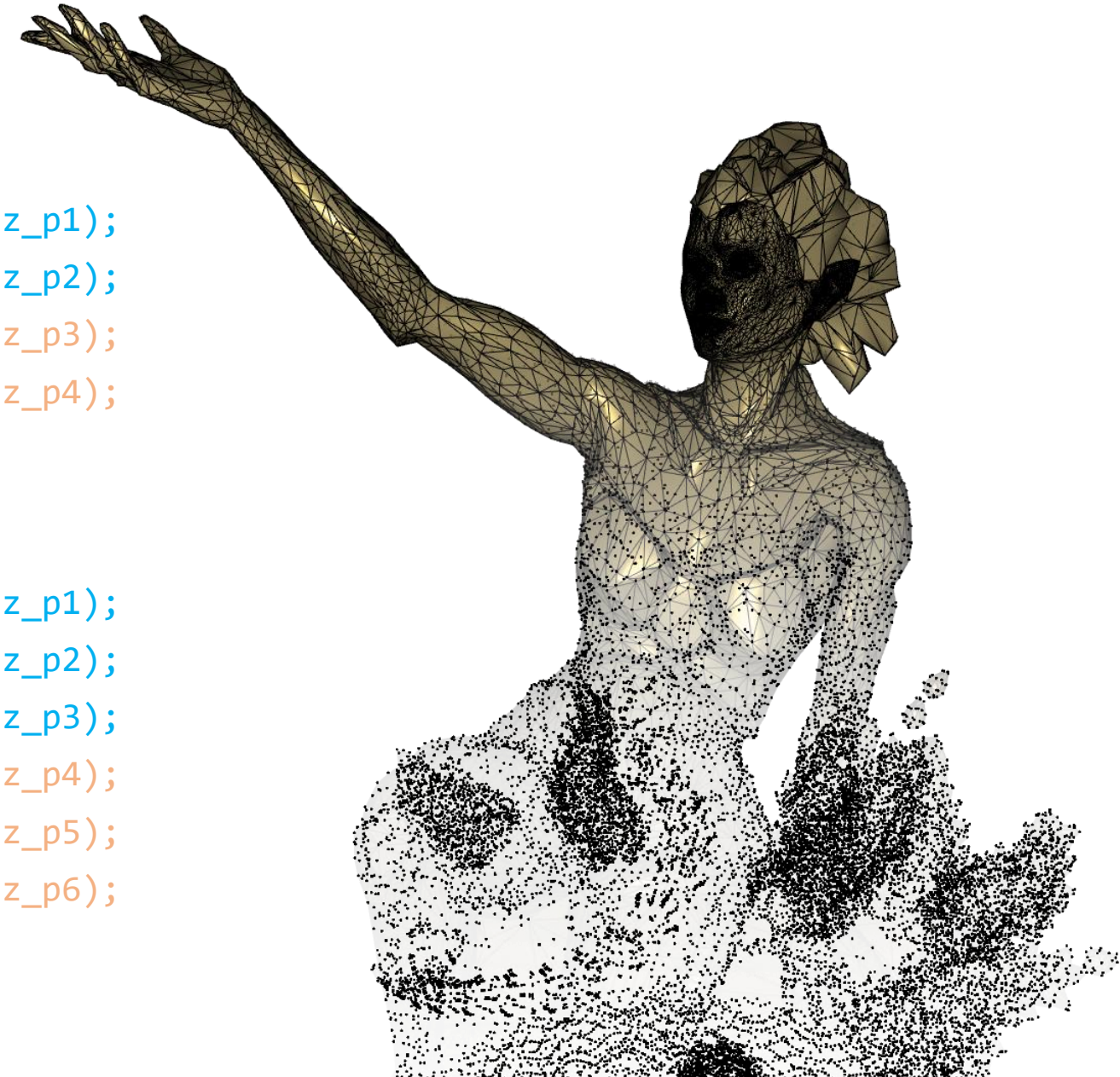
Funciones de OpenGL

- **Primitivas básicas:**
  - Líneas:

```
glBegin(GL_LINES);
    glVertex3f(x_p1, y_p1, z_p1);
    glVertex3f(x_p2, y_p2, z_p2);
    glVertex3f(x_p3, y_p3, z_p3);
    glVertex3f(x_p4, y_p4, z_p4);
glEnd();
```

  - Triángulos:

```
glBegin(GL_TRIANGLES);
    glVertex3f(x_p1, y_p1, z_p1);
    glVertex3f(x_p2, y_p2, z_p2);
    glVertex3f(x_p3, y_p3, z_p3);
    glVertex3f(x_p4, y_p4, z_p4);
    glVertex3f(x_p5, y_p5, z_p5);
    glVertex3f(x_p6, y_p6, z_p6);
glEnd();
```



Funciones de OpenGL

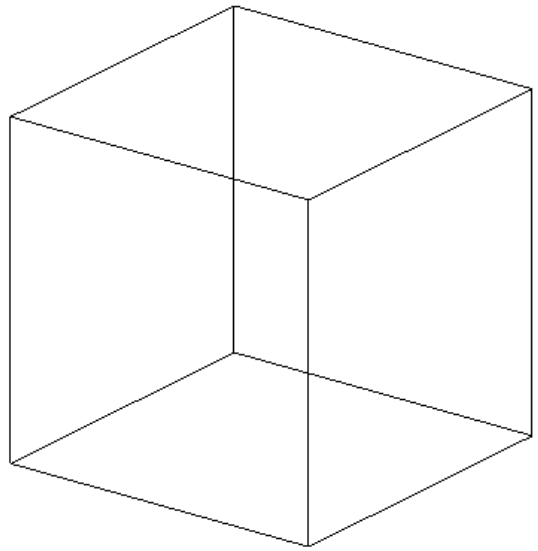
- **Primitivas básicas:**
  - Líneas:

```
glBegin(GL_LINES);
    glVertex3f(x_p1, y_p1, z_p1);
    glVertex3f(x_p2, y_p2, z_p2);
    glVertex3f(x_p3, y_p3, z_p3);
    glVertex3f(x_p4, y_p4, z_p4);
glEnd();
```

  - Triángulos:

```
glBegin(GL_TRIANGLES);
    glVertex3f(x_p1, y_p1, z_p1);
    glVertex3f(x_p2, y_p2, z_p2);
    glVertex3f(x_p3, y_p3, z_p3);
    glVertex3f(x_p4, y_p4, z_p4);
    glVertex3f(x_p5, y_p5, z_p5);
    glVertex3f(x_p6, y_p6, z_p6);
glEnd();
```

```
glutWireCube(n_arista);
```



Tipos básicos de OpenGL

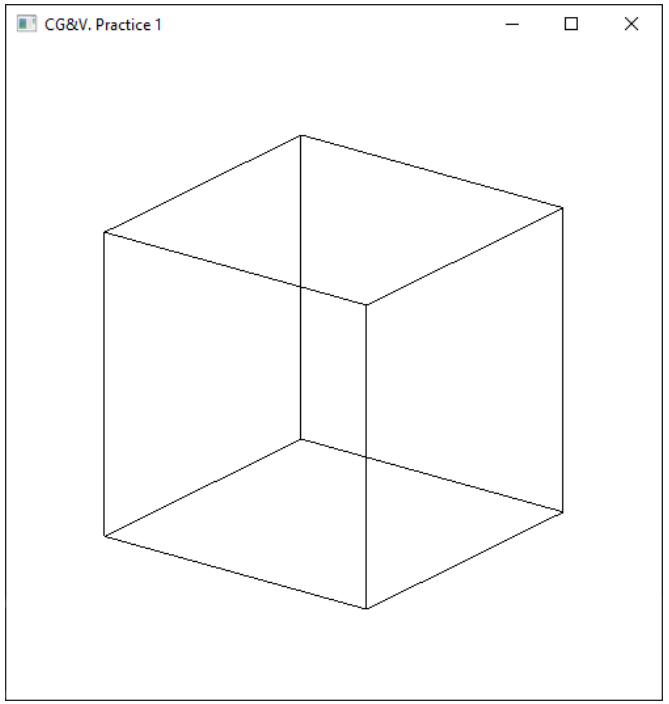
Sufijo	Tipo de datos	Tipo de datos en C	Definición OpenGL
b	Entero 8 bits	signed char	GLbyte
s	Entero 16 bits	short	GLbyte
i	Entero 32 bits	long	GLbyte
f	Real 32 bits	float	GLbyte
d	Real 64 bits	double	GLbyte
ub	Entero sin signo, 8 bits	unsigned char	GLubyte, GLboolean
us	Entero sin signo, 16 bits	unsigned short	GLushort
ui	Entero sin signo, 32 bits	unsigned long	GLuint, GLenum, GLbitfield

```
glVertex{1234}{sifd}[v](coordenadas);
glVertex2s(2,3);                                  // Vértice 2D coordenadas enteras
glVertex3d(0.0, 0.0, 3.1415926536898);            // Vértice 3D coord. doubles

GLdouble dvect[3]={5.0, 9.0, 1992.0};             // Array de doubles
glVertex3dv(dvect);                               // Vértice 3D dado como array de doubles
```

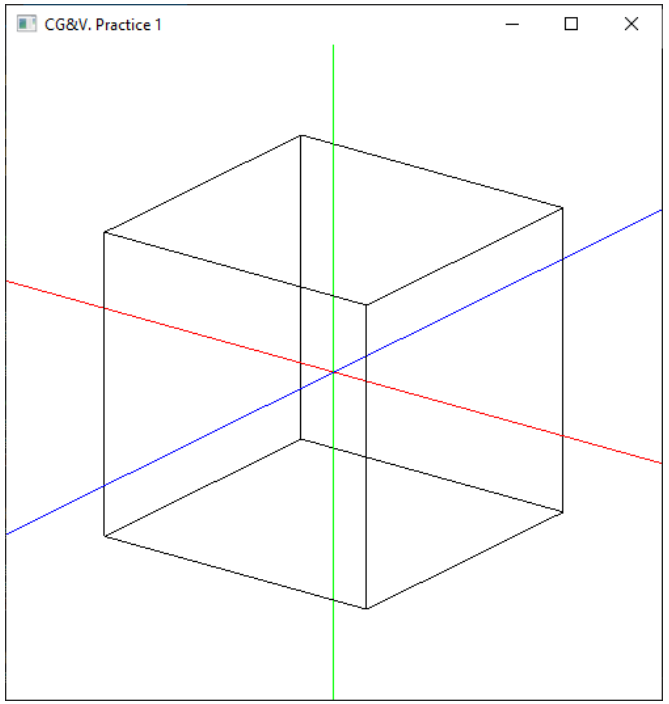
Ejercicios de la práctica 1

- **Primitivas básicas:**

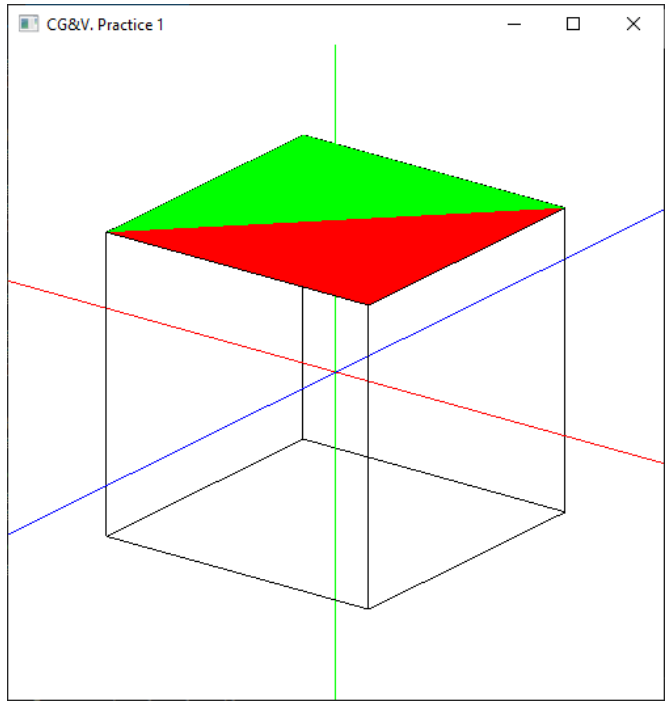
```
glutWireCube(n_arista);
```




```
glBegin(GL_LINES);
    ...
```




```
glBegin(GL_TRIANGLES);
    ...
```



Práctica 2. Transformaciones geométricas

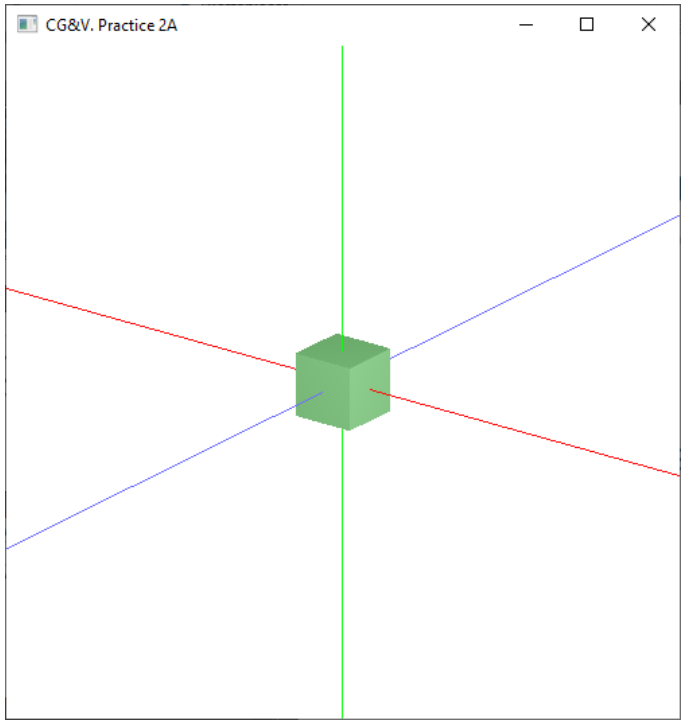
Curso académico 2023-2024

Informática Gráfica y Visualización

Grado en Ingeniería Informática

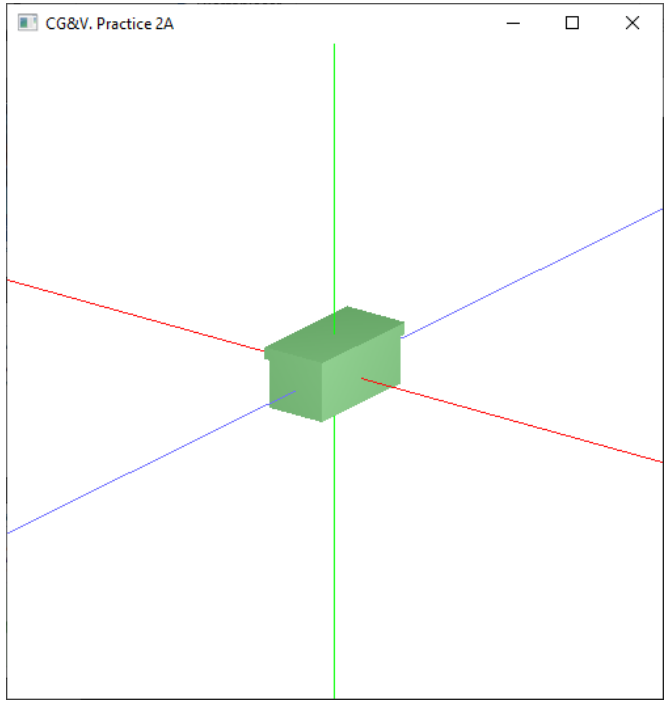
Estructura del proyecto

- **pr2a:** main.
- **igvInterfaz:** gestiona *input* del usuario.
- **igvEscena:** gestiona visualización.
  - `glutSolidCube (GLdouble size)`



Ejercicios

- **Ejercicio A.**
  - Dibujar un cubo base (escalado no uniforme).
  - Dibujar un cube sobre este ultimo (**traslación** + escalado no uniforme).

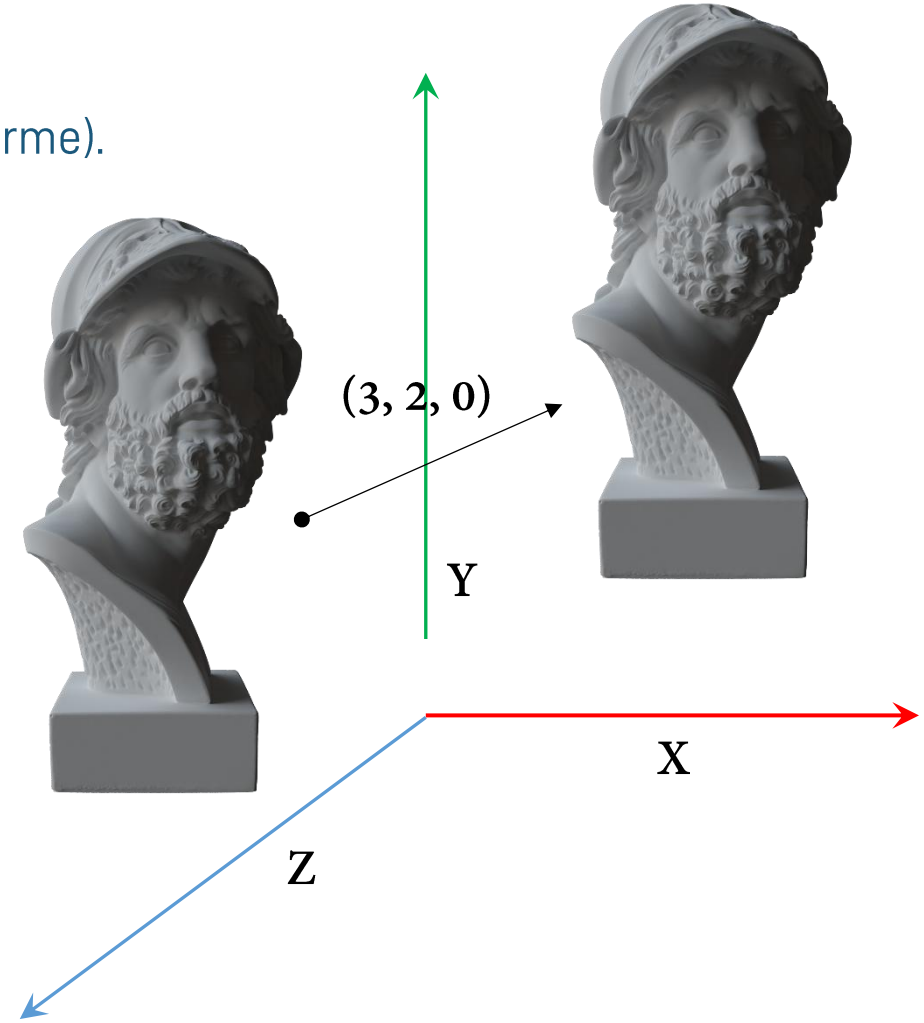


$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \end{array}\right]$$

Ejercicios

- **Ejercicio A.**
  - Dibujar un cubo base (escalado no uniforme).
  - Dibujar un cube sobre este ultimo (**traslación** + escalado no uniforme).

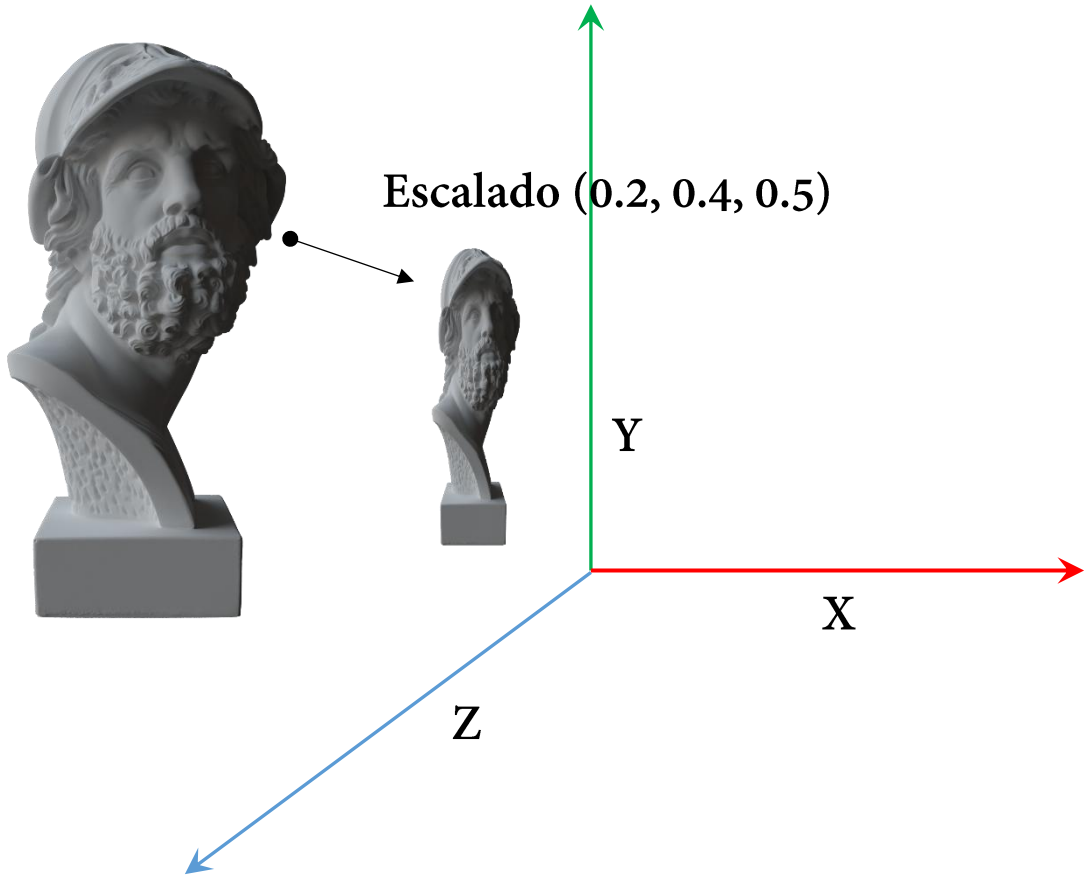
$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \end{array}\right] \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \longrightarrow \begin{bmatrix} x + t_x \\ y + t_y \\ z + t_z \end{bmatrix}$$



Ejercicios

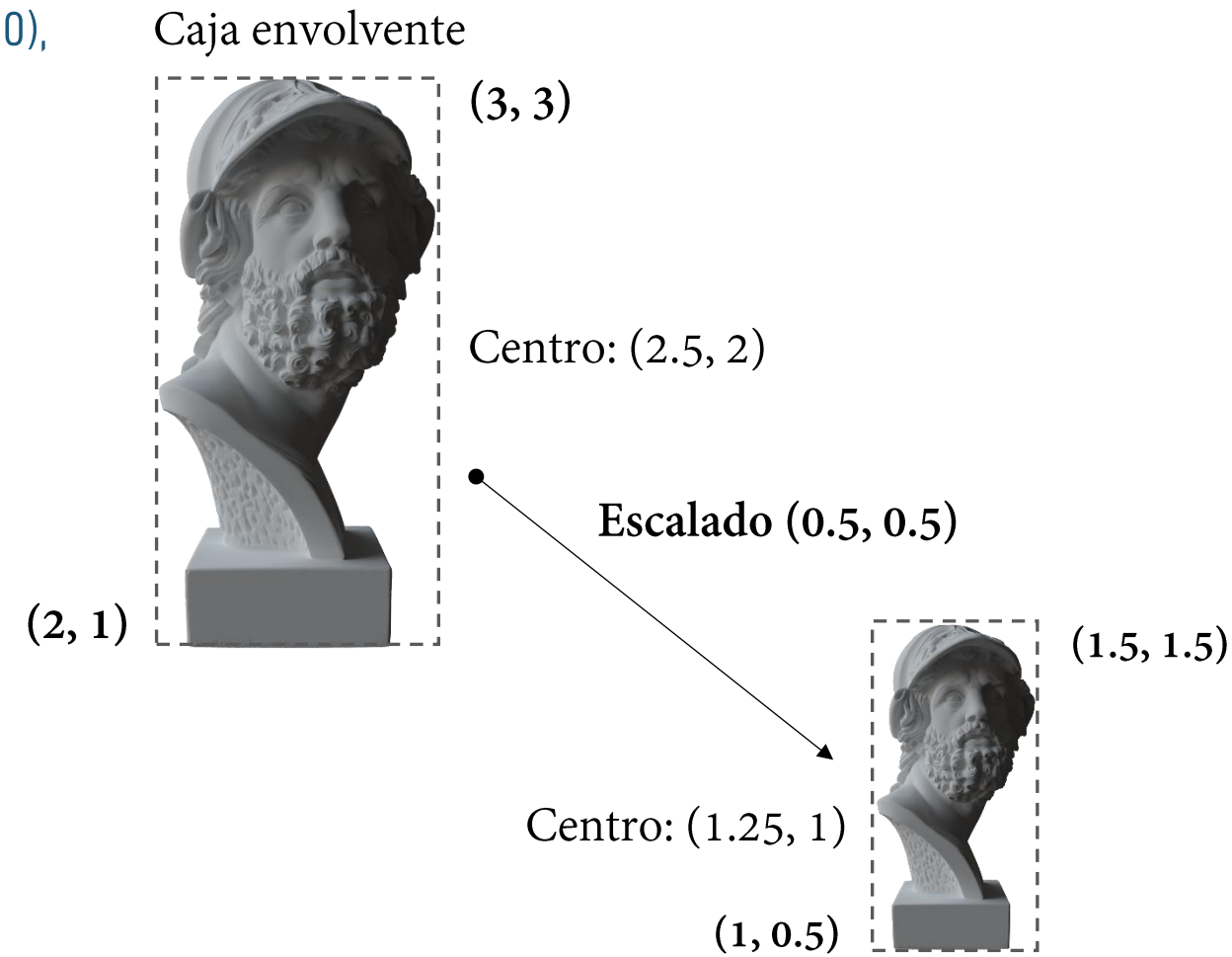
- **Ejercicio A.**
  - Dibujar un cubo base (**escalado no uniforme**).
  - Dibujar un cube sobre este ultimo (traslación + **escalado no uniforme**).

$$\left[\begin{array}{ccc|c} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \end{array}\right] \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \longrightarrow \begin{bmatrix} s_x x \\ s_y y \\ s_z z \end{bmatrix}$$



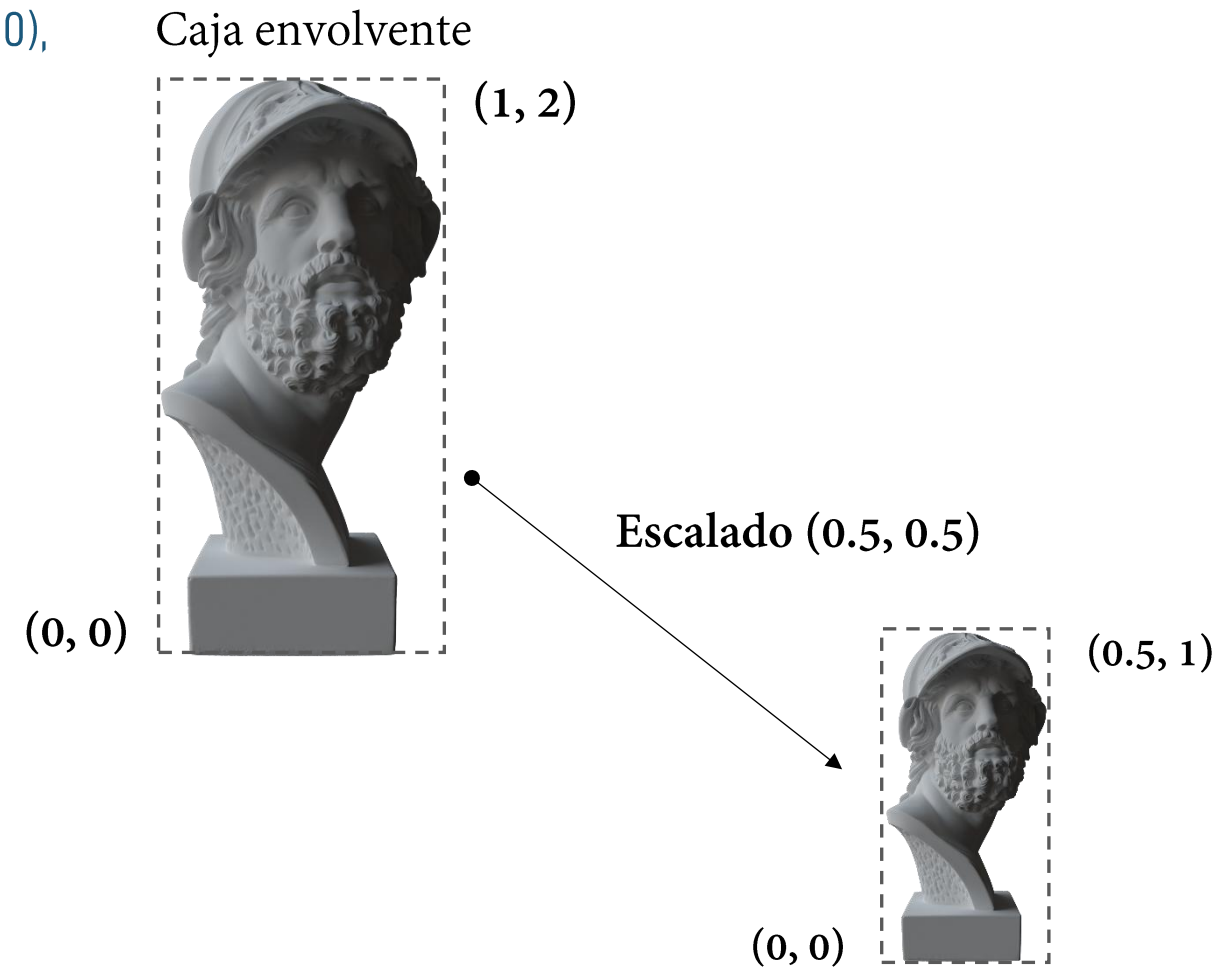
Transformaciones geométricas

- **Escalado.**
  - **Problema**: si no lo hacemos en torno al (0, 0, 0), es más difícil seguir el proceso de transformación.
  - Centro: de (2.5, 2) a (1.25, 1).



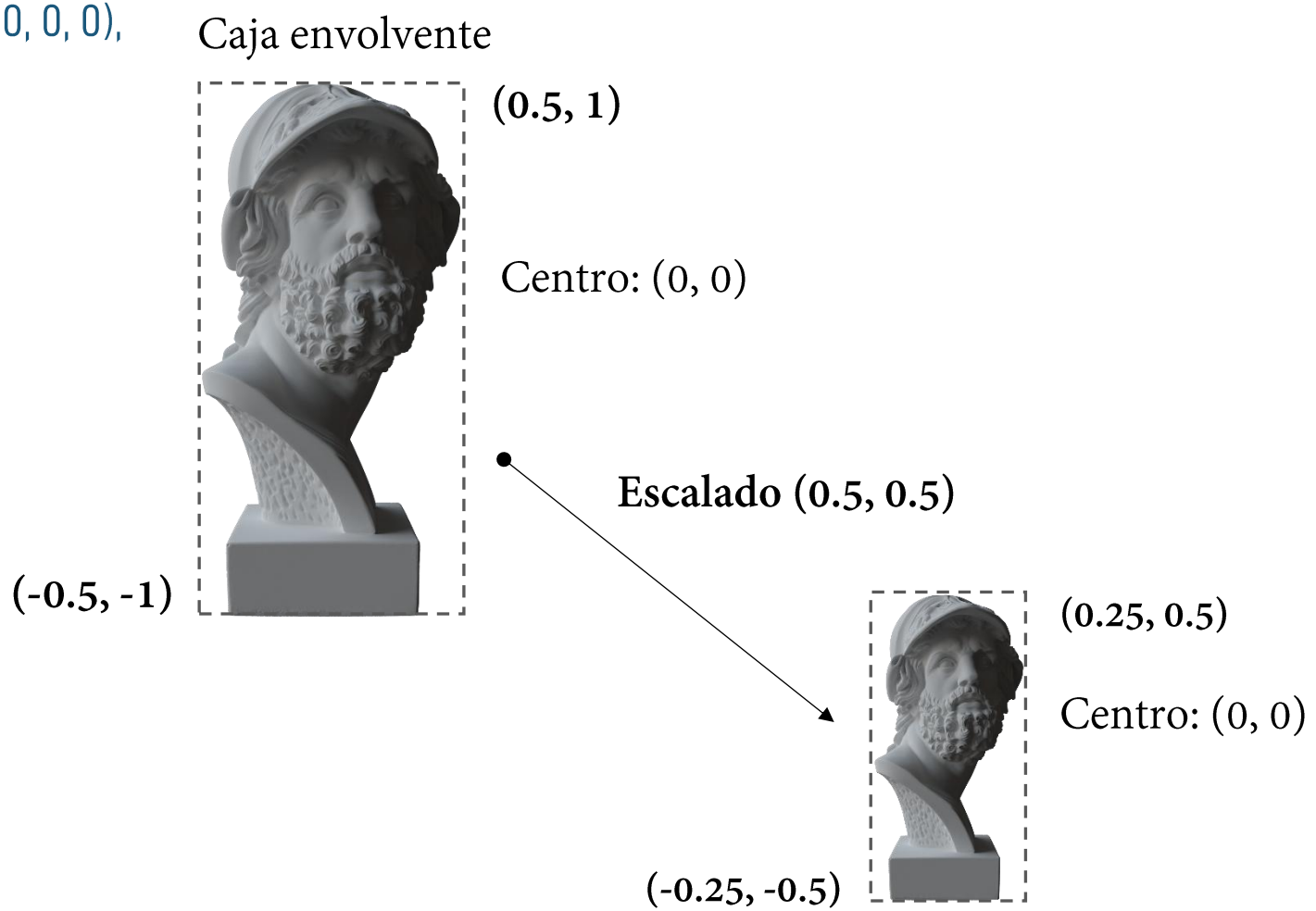
Transformaciones geométricas

- **Escalado.**
  - **Problema**: si no lo hacemos en torno al (0, 0, 0), es más difícil continuar el proceso de transformación.
  - Mínimo: de (0, 0) a (0, 0).



Transformaciones geométricas

- **Escalado.**
  - **Problema**: si no lo hacemos en torno al (0, 0, 0), es más difícil continuar el proceso de transformación.
  - Centro: de (0, 0) a (0, 0).



Transformaciones geométricas

- **Pila de matrices.**
  - PushMatrix()
    - ...
    - Escalado 1
    - Rotación 1
    - Traslación 2
    - Traslación 1
  - PopMatrix()

Composición de matrices

¡Ojo con el orden! Al multiplicar matrices, las últimas añadidas a la pila serán las primeras en aplicarse.

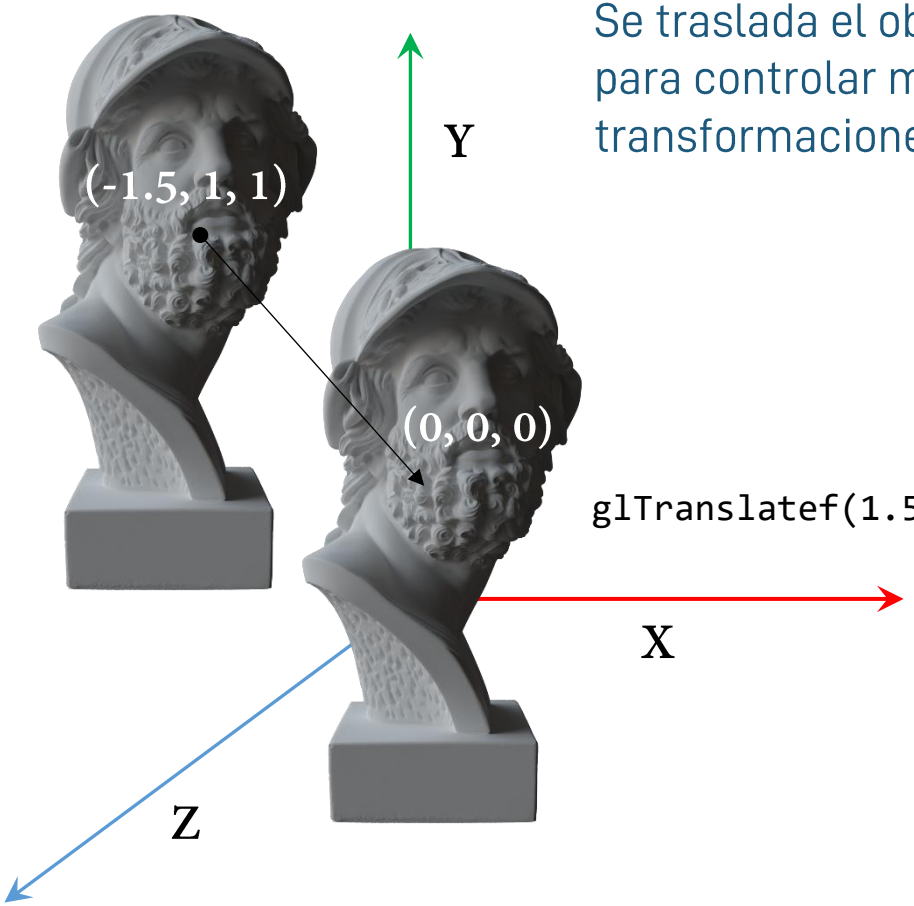
Se suele controlar primero la posición del objeto ([Traslación (centrado)]), después rotar y escalar, y finalmente se aplican las traslaciones.

No tiene por qué mantenerse este orden siempre y cuando estemos seguros de la posición del objeto.

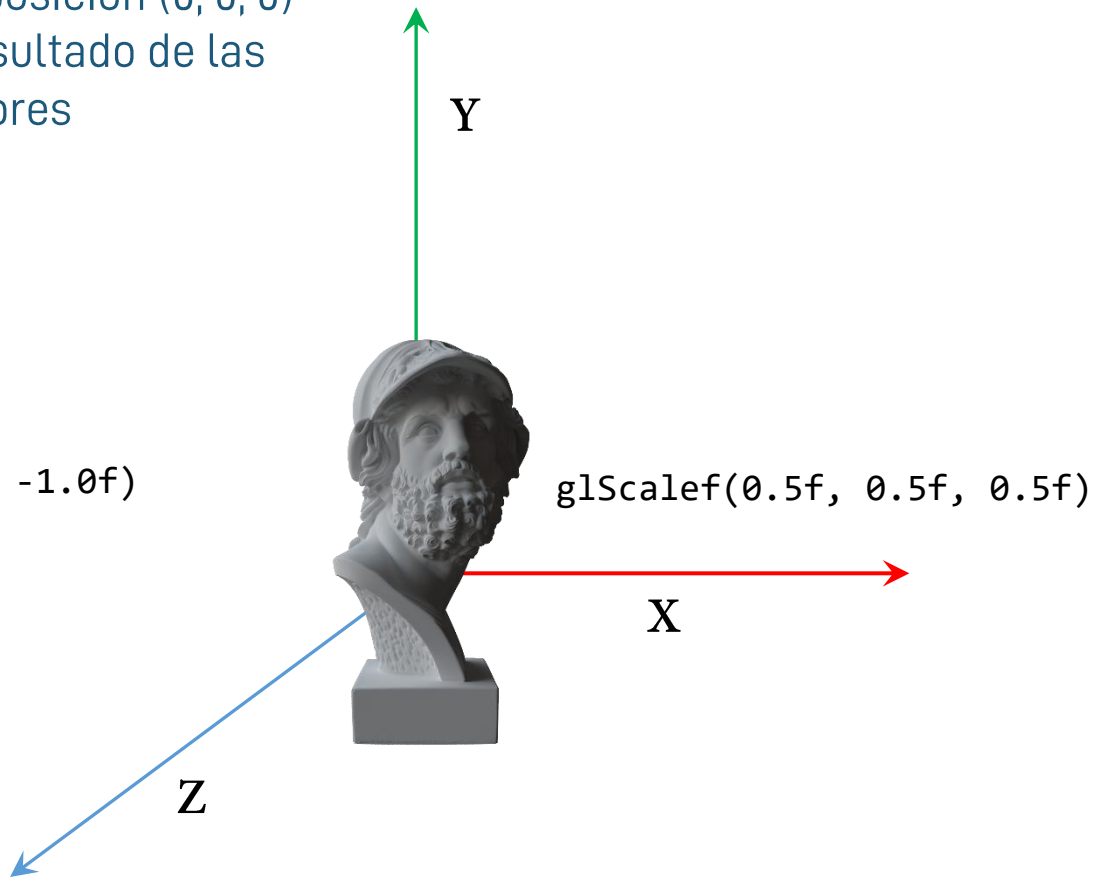
- PushMatrix()
  - ...
  - Traslación 2
  - Traslación 1
  - Escalado 1
  - Rotación 1
  - [Traslación (centrado)]
- PopMatrix()

Transformaciones geométricas

Se traslada el objeto a la posición (0, 0, 0) para controlar mejor el resultado de las transformaciones posteriores



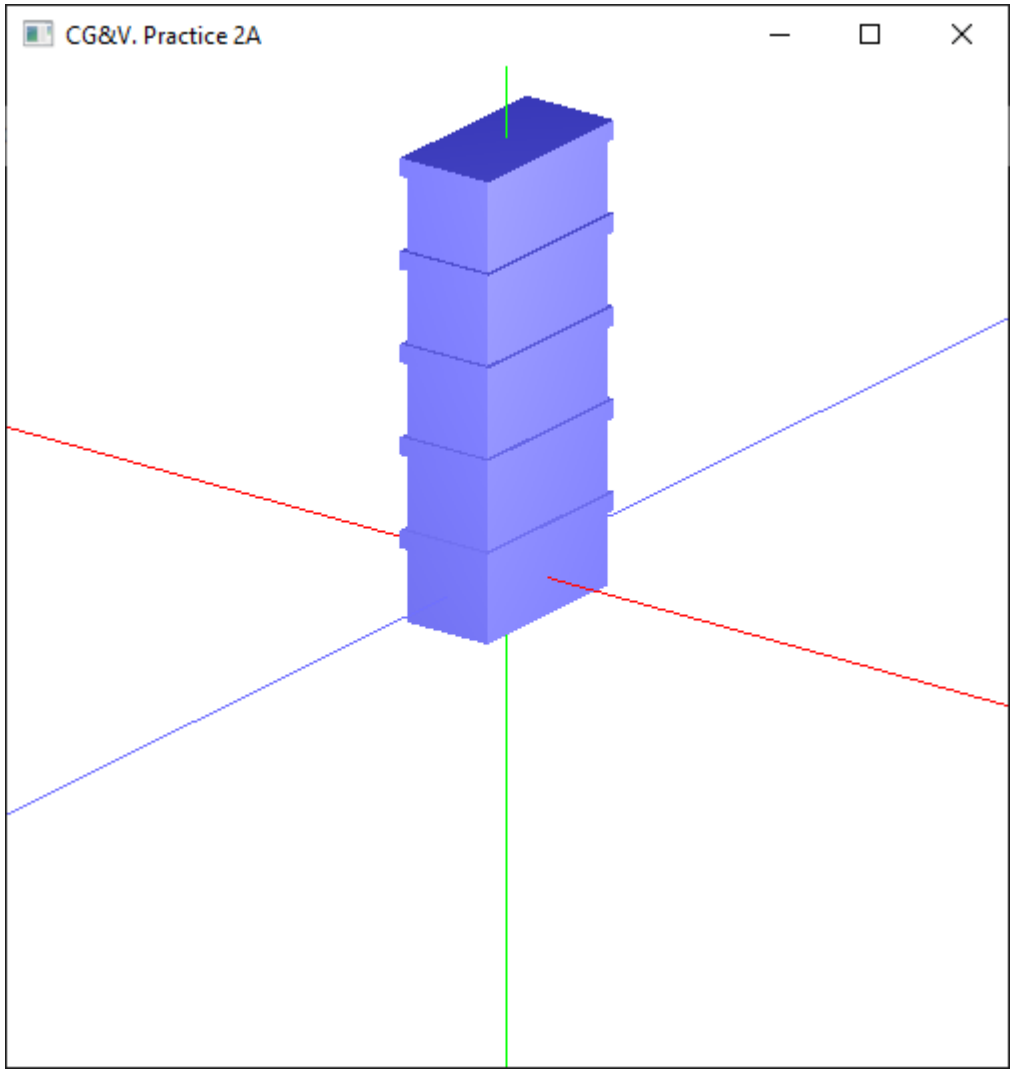

```
glTranslatef(1.5f, -1.0f, -1.0f)
```




```
glScalef(0.5f, 0.5f, 0.5f)
```

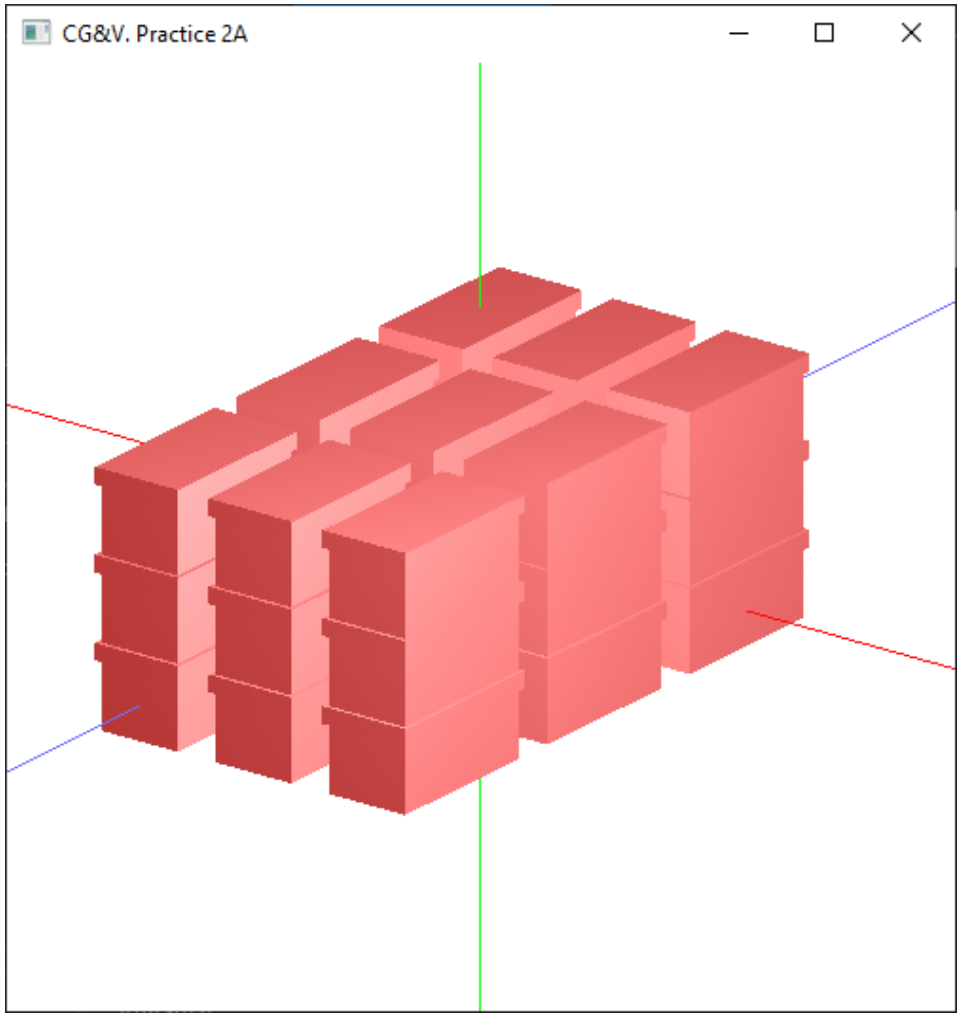
Ejercicios

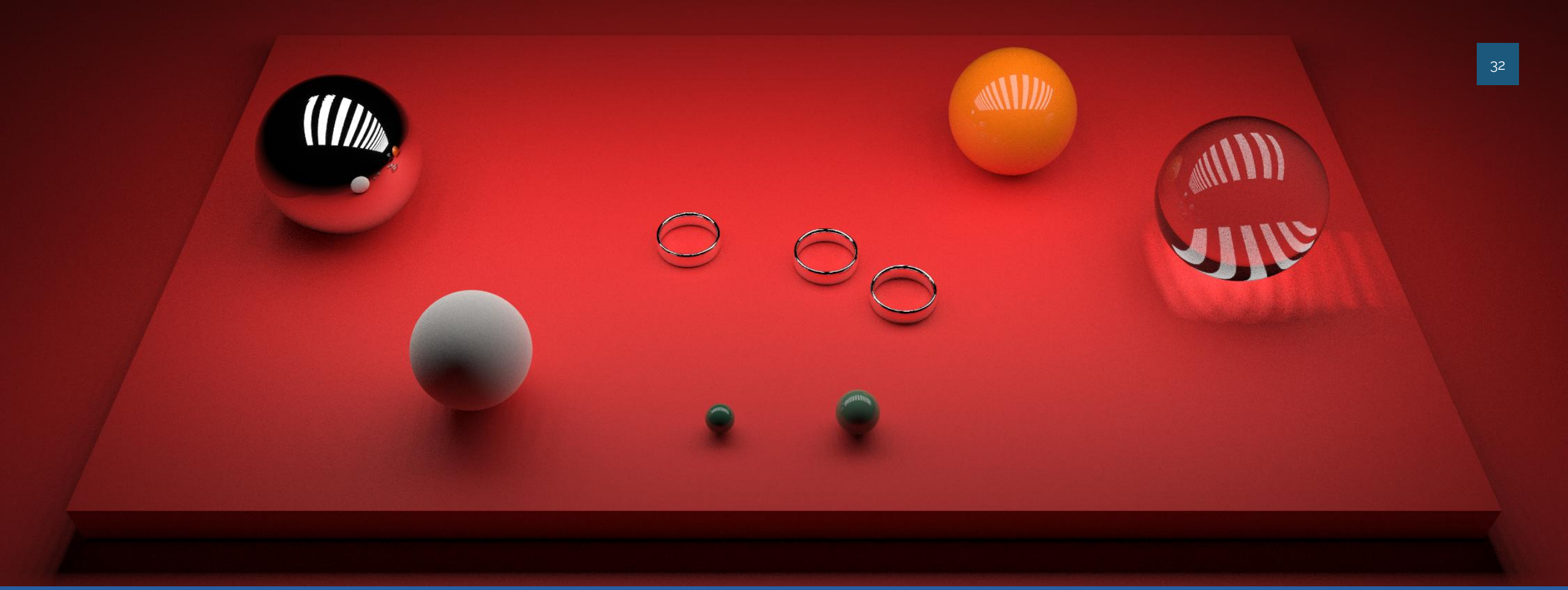
- **Ejercicio B.**
  - Acumular bloques en eje Y.
    - La forma A tiene escala (x_1, y_1, z_1).
    - La forma B tiene escala (x_2, y_2, z_2).
    - Acumular bloques sabiendo que la altura es:
      - **Traslación**: (0, **? * y**, 0)



Ejercicios

- **Ejercicio C.**
  - Acumular bloques en eje Y.
    - La forma A tiene escala (x_1, y_1, z_1).
    - La forma B tiene escala (x_2, y_2, z_2).
    - Acumular bloques sabiendo que la altura es …
      - Traslación: (0, ? * y, 0)
    - Dispersar bloques con un **desplazamiento** de 0.5.
      - Primeros bloques: **desplazamiento cero**.





Práctica 2. La cámara de visión, proyecciones y viewport

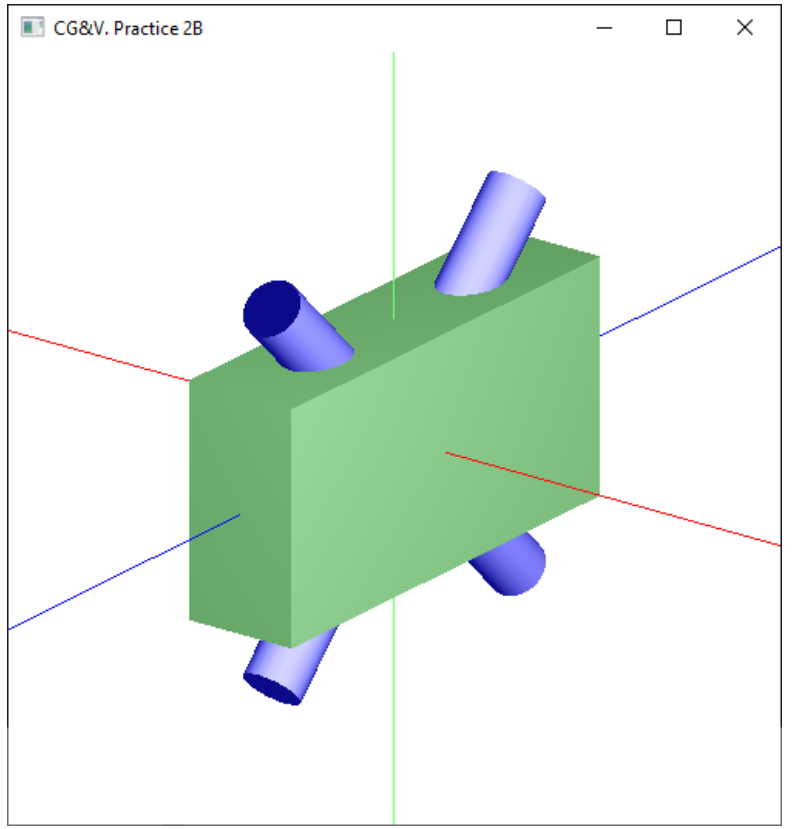
Curso académico 2023-2024

Informática Gráfica y Visualización

Grado en Ingeniería Informática

Estructura del proyecto

- **pr2b:** main.
- **igvInterfaz:** gestión de eventos y ventana.
- **igvEscena:** gestiona contenido de la escena.
- **igvPunto3D**: declaración de objetos tipo punto y vector.
- **igvCamara**: configuración de visualización de la aplicación.



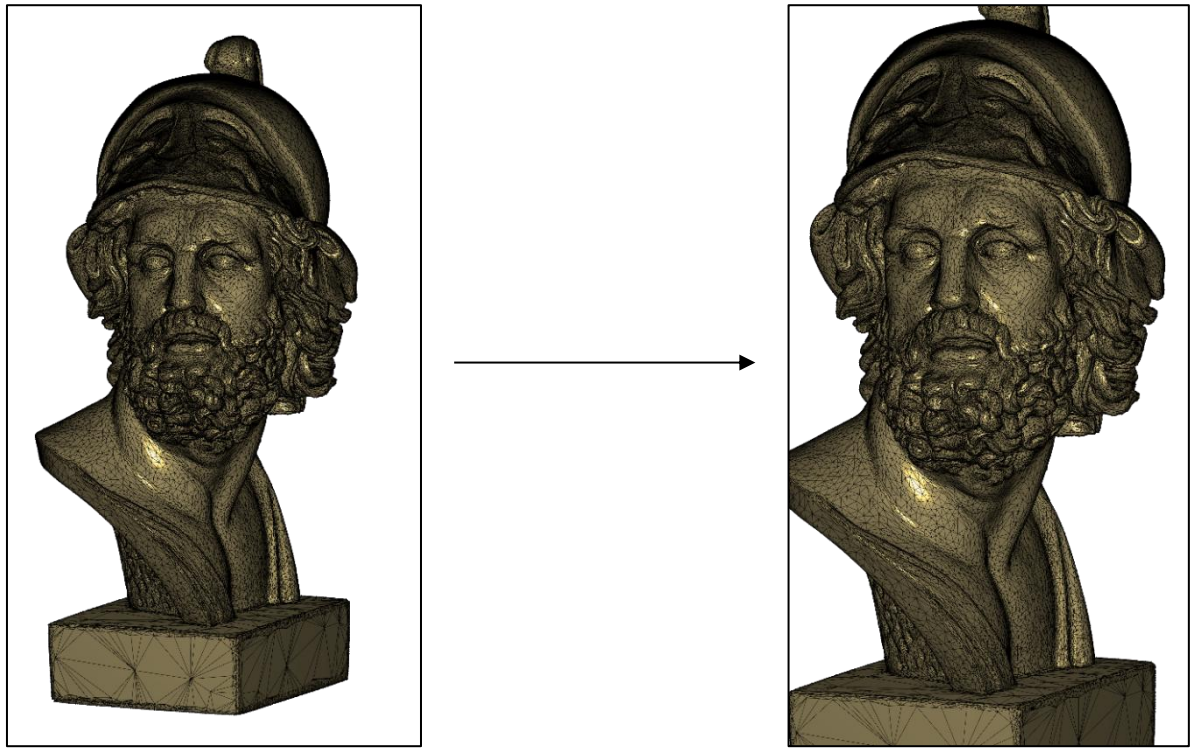
Ejercicios

- **Ejercicio A.**
  - Cambiar tipo de proyección.
    - Perspectiva.
    - Ortográfica.
  - Tecla p/P.

Ejercicios

- **Ejercicio A.**
  - `igvCamera::aplicar()`
    - **`GL_PROJECTION`**
      - `glOrtho ( xmin, xmax, ymin, ymax, znear, zfar )`

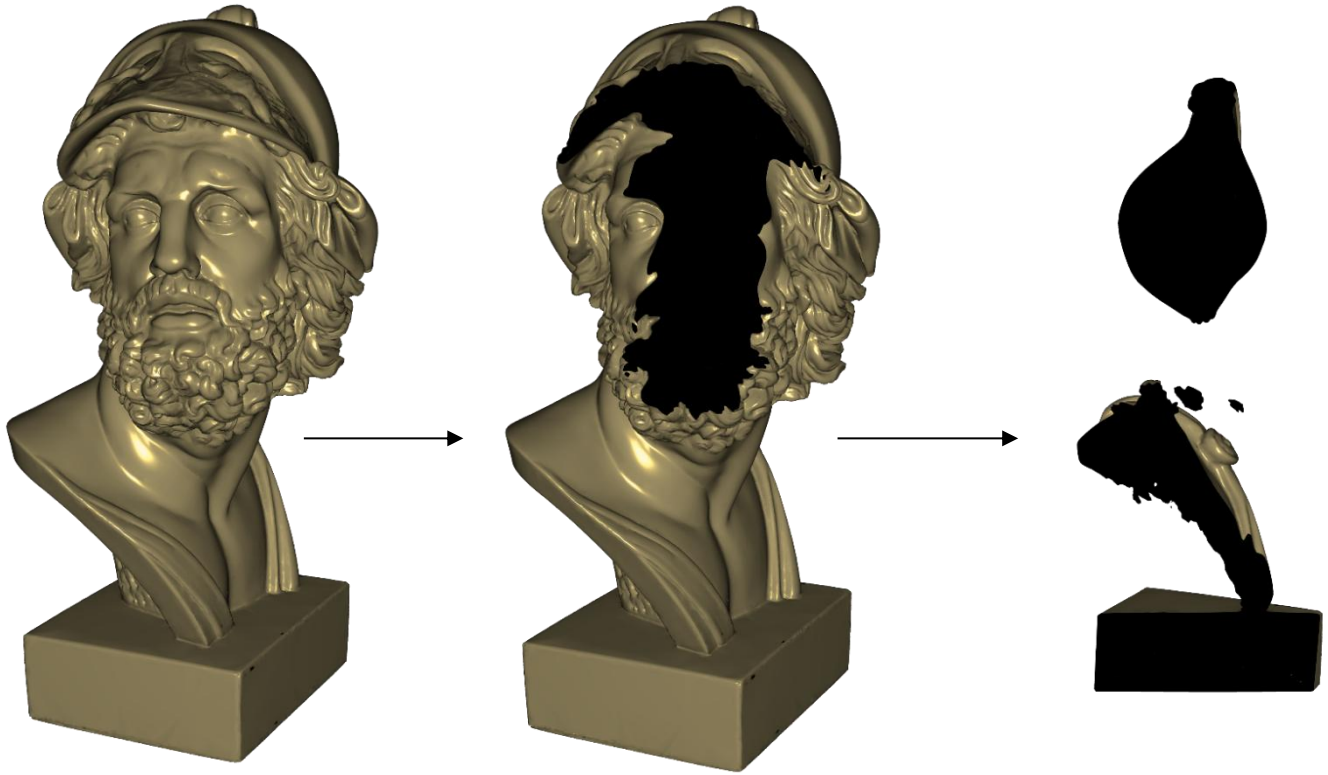
(xmax, ymax)



(xmin, ymin)

Ejercicios

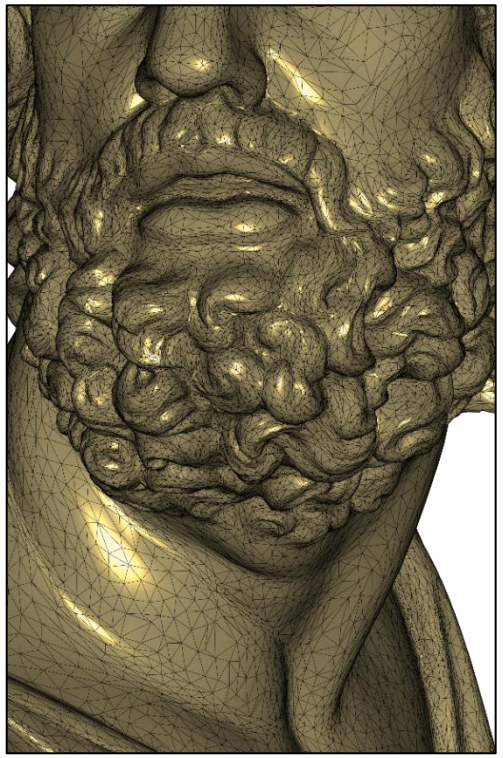
- **Ejercicio A.**
  - `igvCamera::aplicar()`
    - **`GL_PROJECTION`**
      - `glOrtho ( xmin, xmax, ymin, ymax, `**`znear, zfar`**` )`

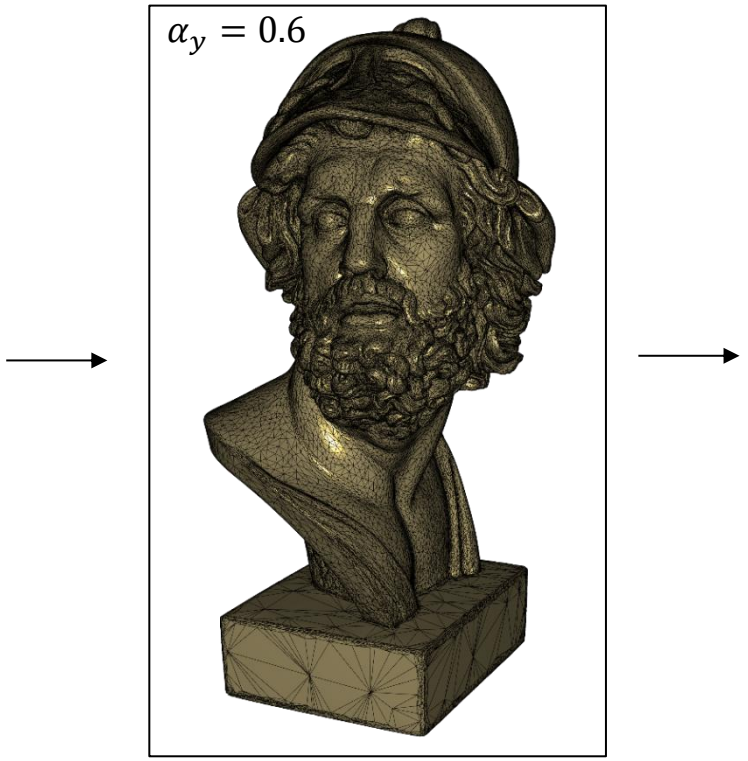


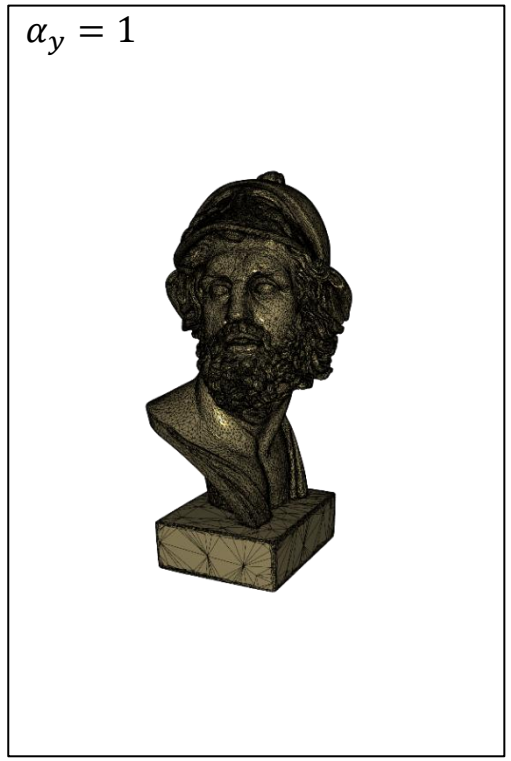
Ejercicios

- **Ejercicio A.**
  - `igvCamera::aplicar()`
    - **`GL_PROJECTION`**
      - `gluPerspective ( angulo, raspecto, znear, zfar )`

$\alpha_y = 0.2$



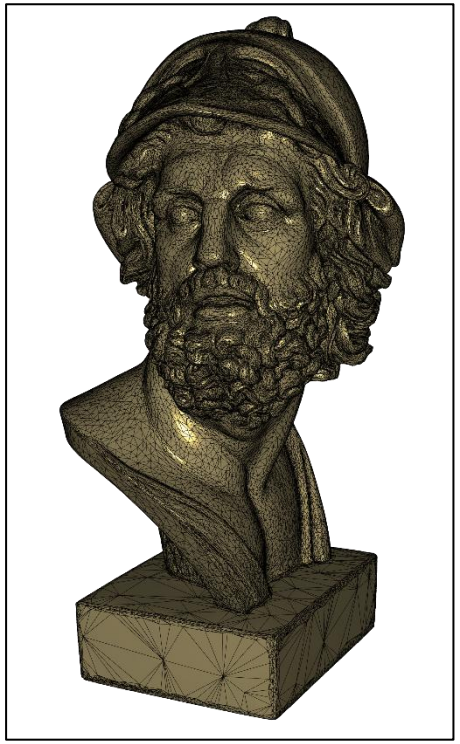




Ejercicios

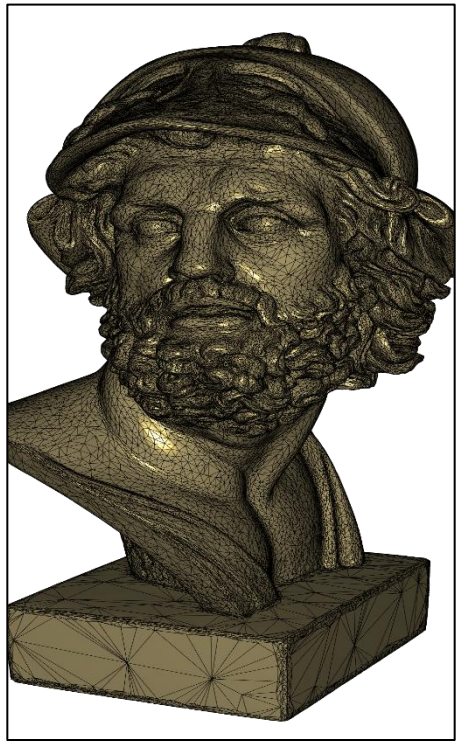
- **Ejercicio A.**
  - `igvCamera::aplicar()`
    - **`GL_PROJECTION`**
      - `gluPerspective ( angulo, raspecto, znear, zfar )`

width = 560



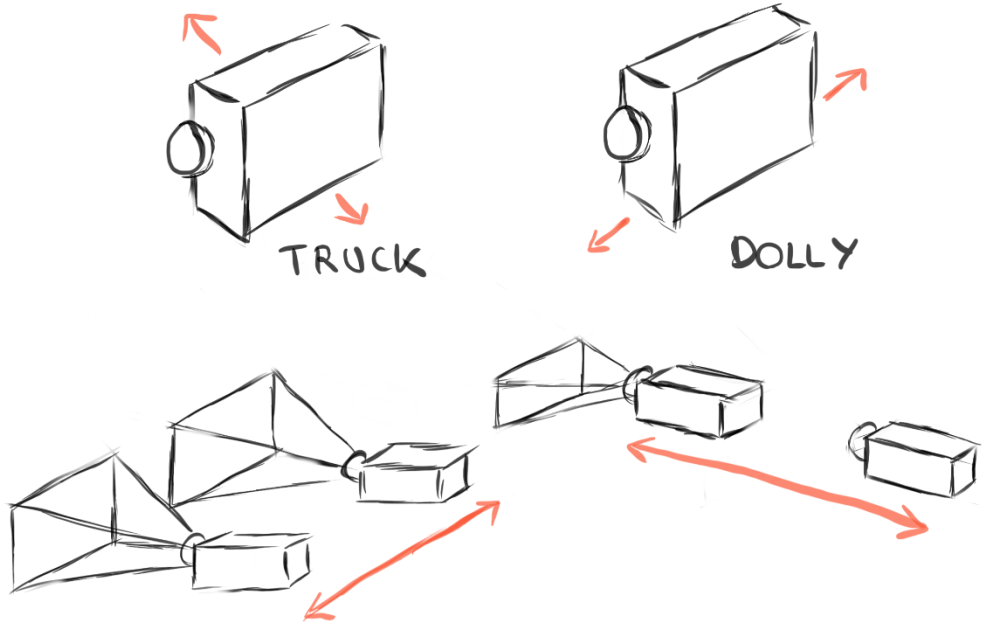
height = 780

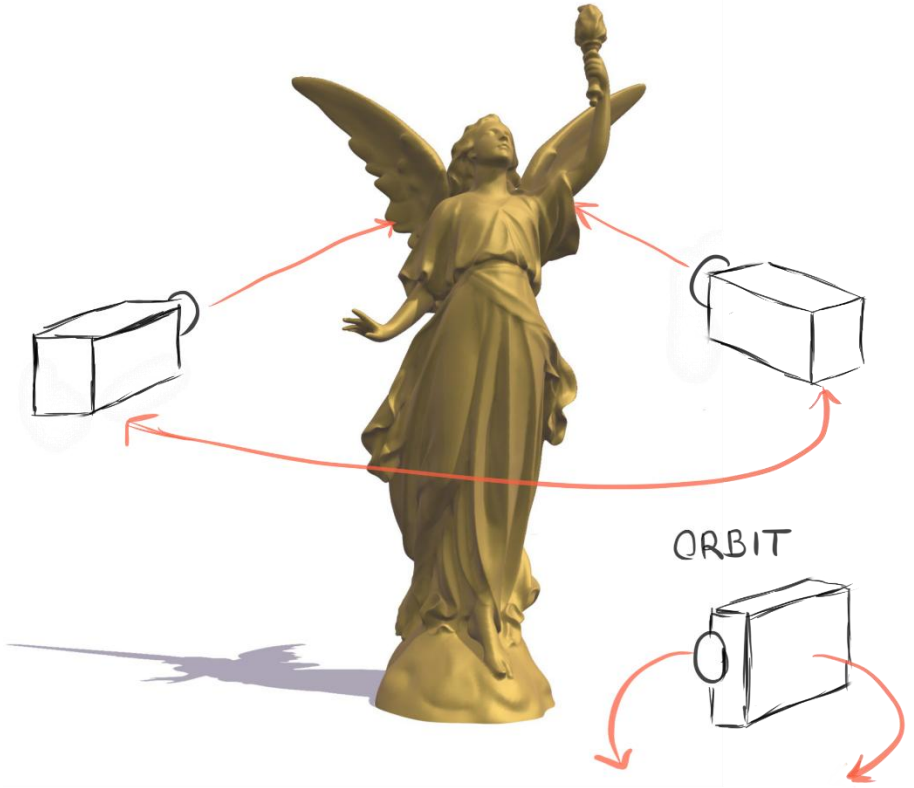
$$aspect = \frac{width}{height}$$



Ejercicios

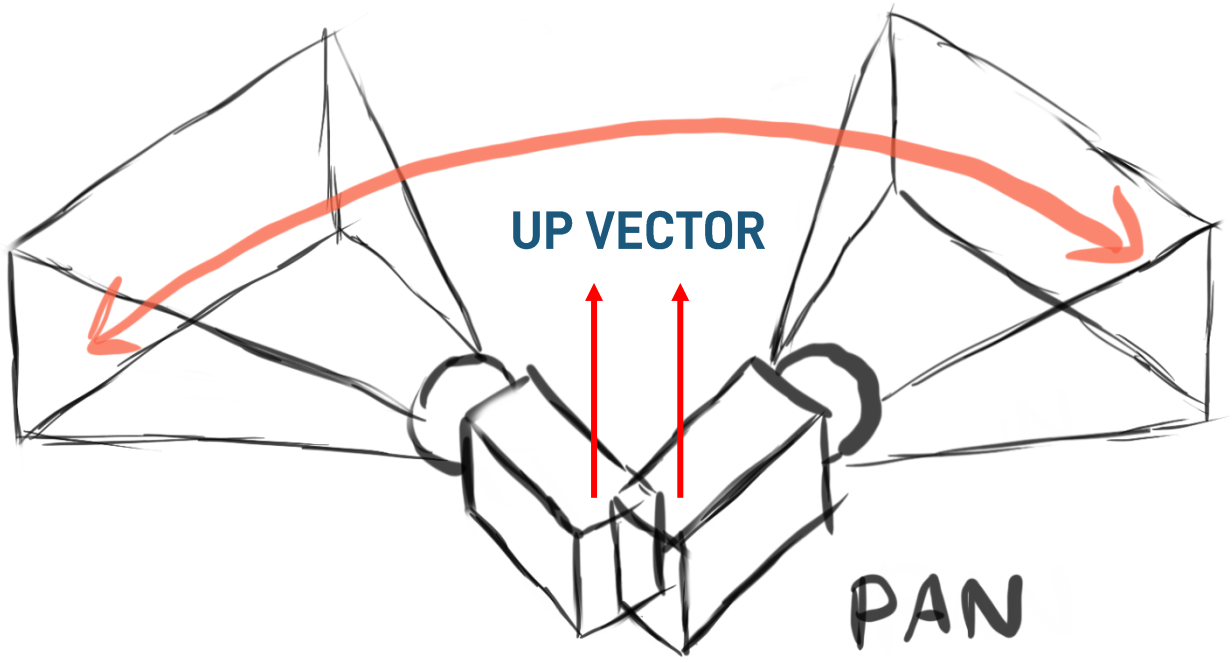
- **Ejercicio A.**
  - `igvCamera::aplicar()`
    - **`GL_VIEW`**
      - `gluLookAt (`**`eye_x, eye_y, eye_z`**`, lookat_x, lookat_y, lookat_z, up_x, up_y, up_z)`

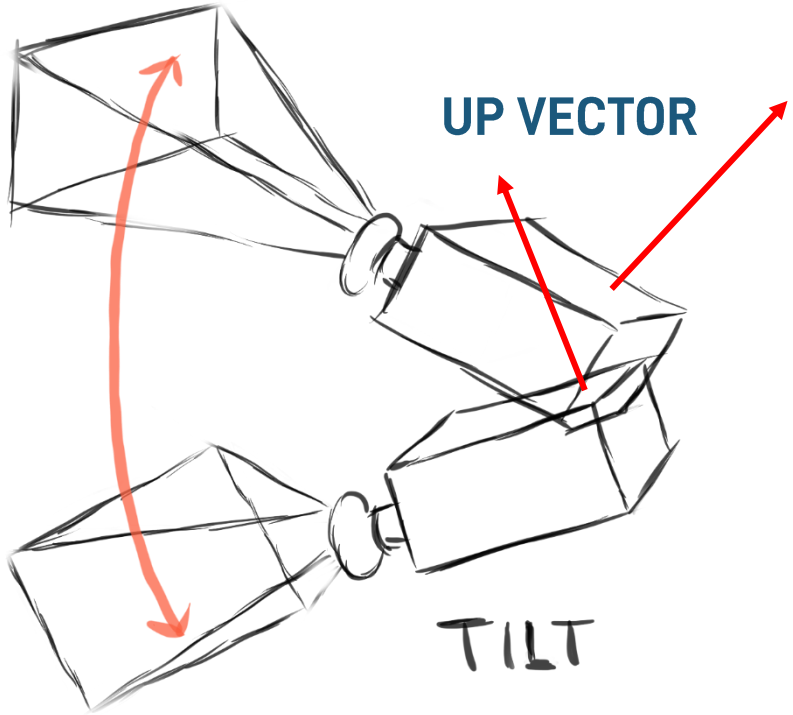




Ejercicios

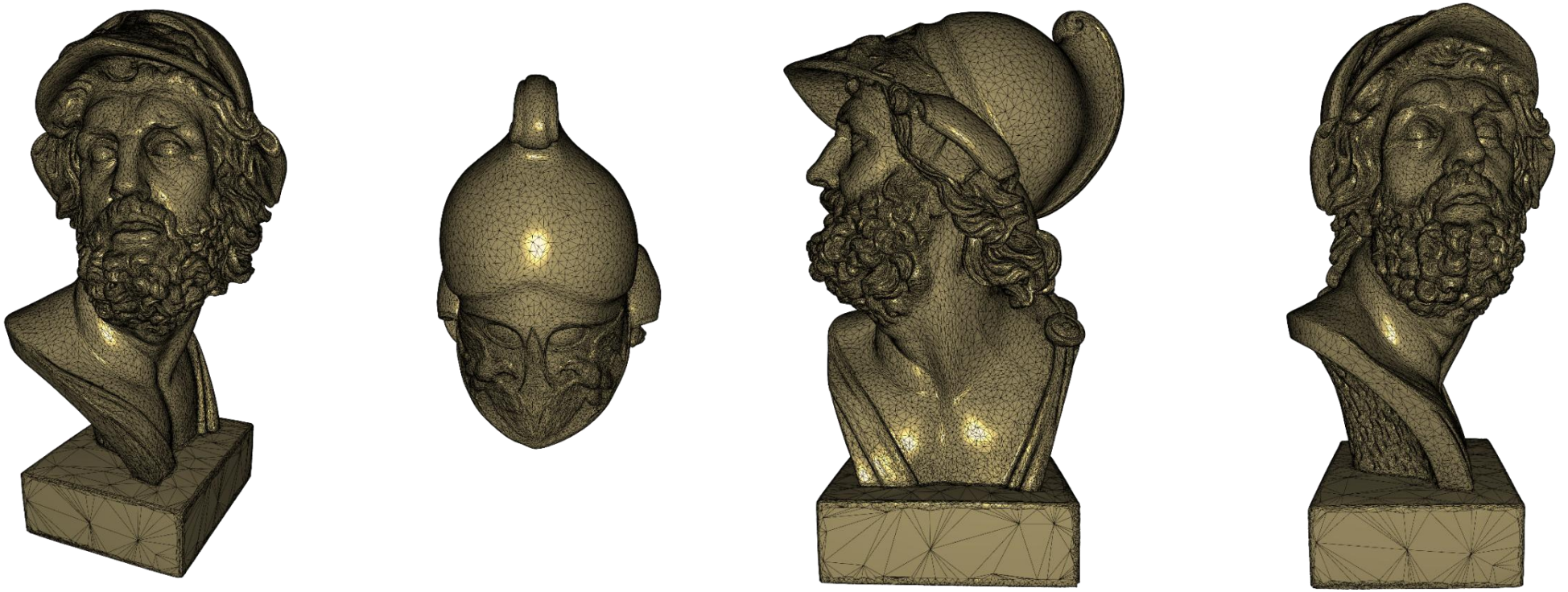
- **Ejercicio A.**
  - `igvCamera::aplicar()`
    - **`GL_VIEW`**
      - `gluLookAt (eye_x, eye_y, eye_z, lookat_x, lookat_y, lookat_z, up_x, up_y, up_z)`





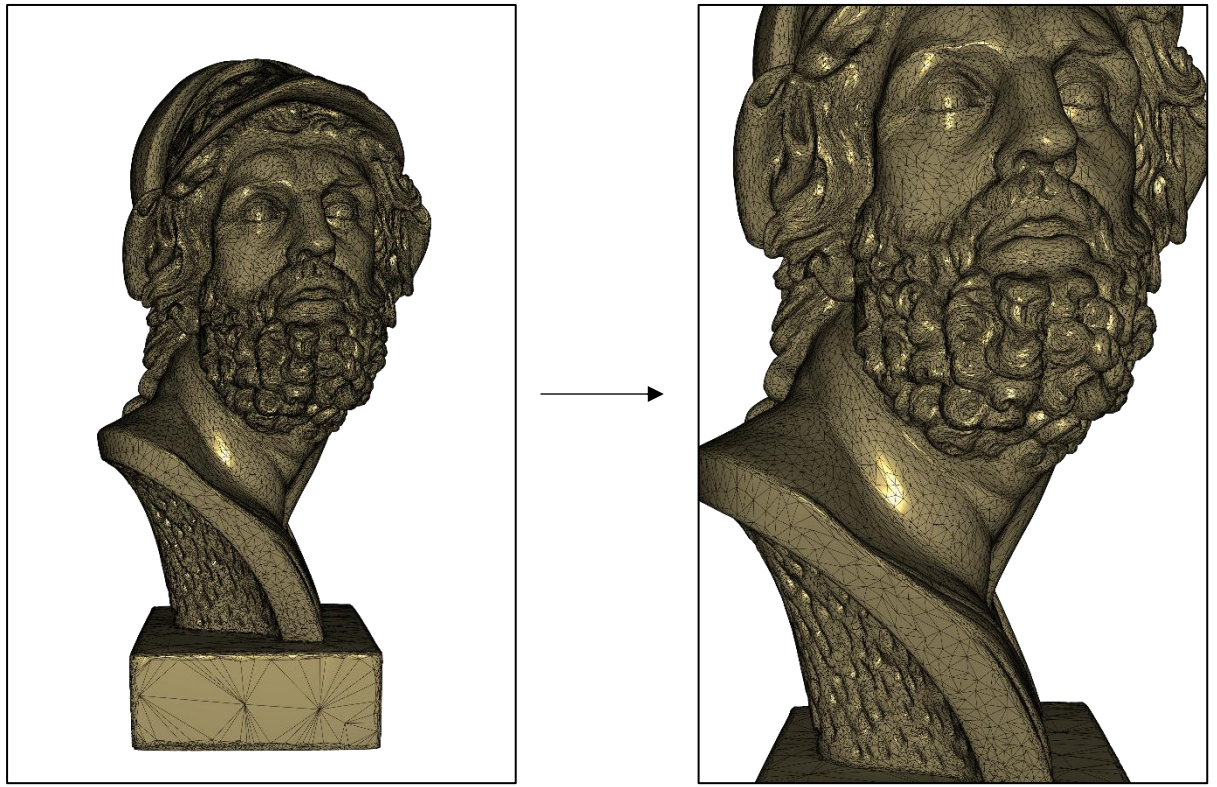
Ejercicios

- **Ejercicio B.**
  - Iteración por cuatro tipo de vistas: inicial, planta, perfil y alzado.
  - Debe seguir funcionando junto con el cambio de proyección.



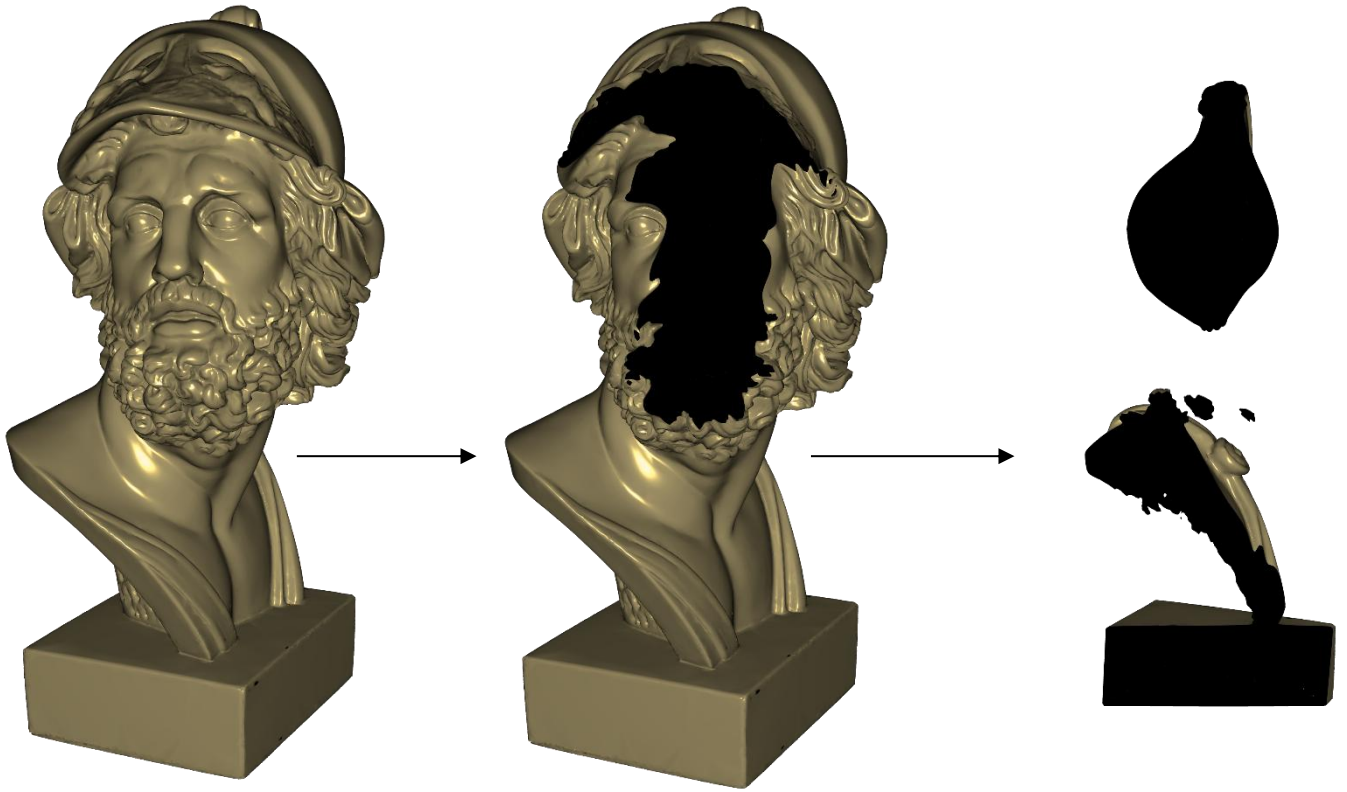
Ejercicios

- **Ejercicio C.**
  - Zoom de la cámara.
  - No se modifica posición de escena ni de cámara.
  - Teclas +/- para acercarse y alejarse.



Ejercicios

- **Ejercicio D.**
  - Modificar *znear* de cámara.
  - Tecla n: +0.2.
  - Tecla N: -0.2.



Ejercicios

- **Ejercicio E.**
  - Cuatro vistas simultáneas de la ventana.
    - Pulsando la tecla 4.
    - Inicial, planta, alzado y perfil.
    - `glViewport ( origen_x, origen_y, tamaño_x, tamaño_y )`

`glViewport ( 0, 0, 1200, 800 )`

Minimapa:

`glViewport ( 1000, 600, 200, 200 )`

Práctica 3a. Mallas de triángulos

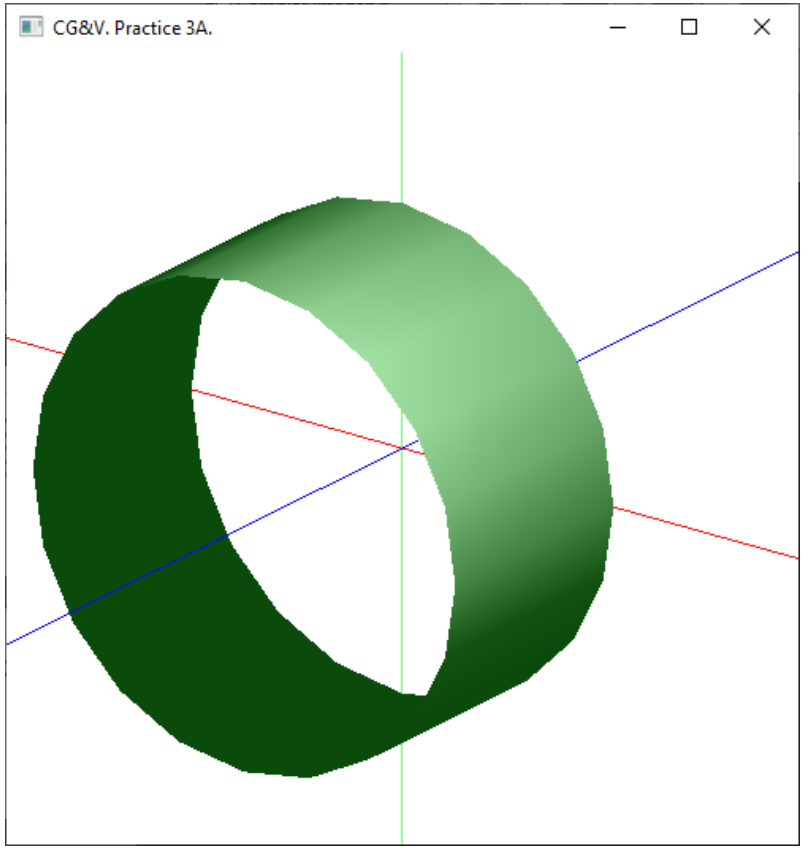
Curso académico 2023-2024

Informática Gráfica y Visualización

Grado en Ingeniería Informática

Estructura del proyecto

- **pr3a:** main.
- **igvInterfaz:** gestión de eventos y ventana.
- **igvEscena:** gestiona contenido presente en la escena.
- **igvPunto3D**: declaración de objetos tipo punto y vector.
- **igvCamara**: configuración de cámara.
- **igvMallaTriangulos**: visualización de malla de triángulos.



Ejercicios

- **Ejercicio A.**
  - Rotación de modelo en los ejes X, Y y Z.
  - Debemos controlar la posición del objeto para saber dónde se encuentra tras una rotación.

$$\begin{pmatrix} \boxed{1} & 0 & 0 \\ \boxed{0} & \cos\beta & -\sin\beta \\ \boxed{0} & \sin\beta & \cos\beta \end{pmatrix}$$

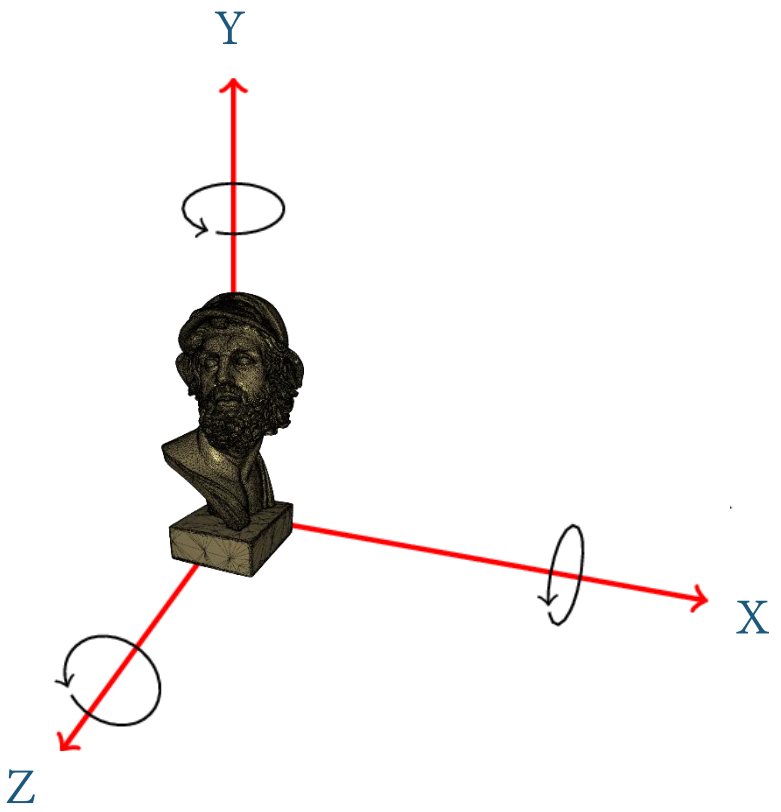
Rotación alrededor del eje X

$$\begin{pmatrix} \cos\beta & \boxed{0} & -\sin\beta \\ 0 & \boxed{1} & 0 \\ \sin\beta & \boxed{0} & \cos\beta \end{pmatrix}$$

Rotación alrededor del eje Y

$$\begin{pmatrix} \cos\beta & -\sin\beta & \boxed{0} \\ \sin\beta & \cos\beta & \boxed{0} \\ 0 & 0 & \boxed{1} \end{pmatrix}$$

Rotación alrededor del eje Z

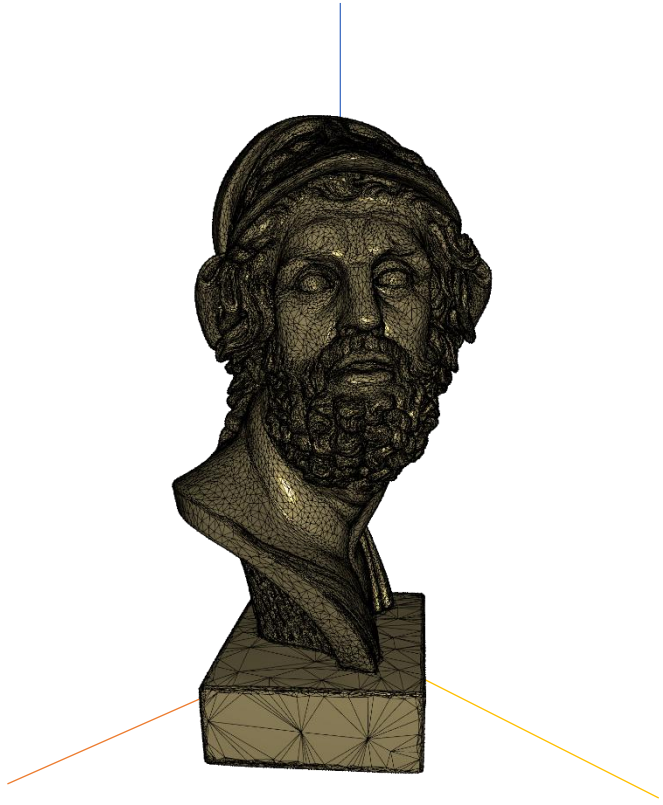


Ejercicios

- **Ejercicio A.**
  - Rotación de modelo en los ejes X, Y y Z.
  - Debemos controlar la posición del objeto para saber dónde se encuentra tras una rotación.

$$\begin{pmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{pmatrix}$$

Rotación alrededor del eje Y



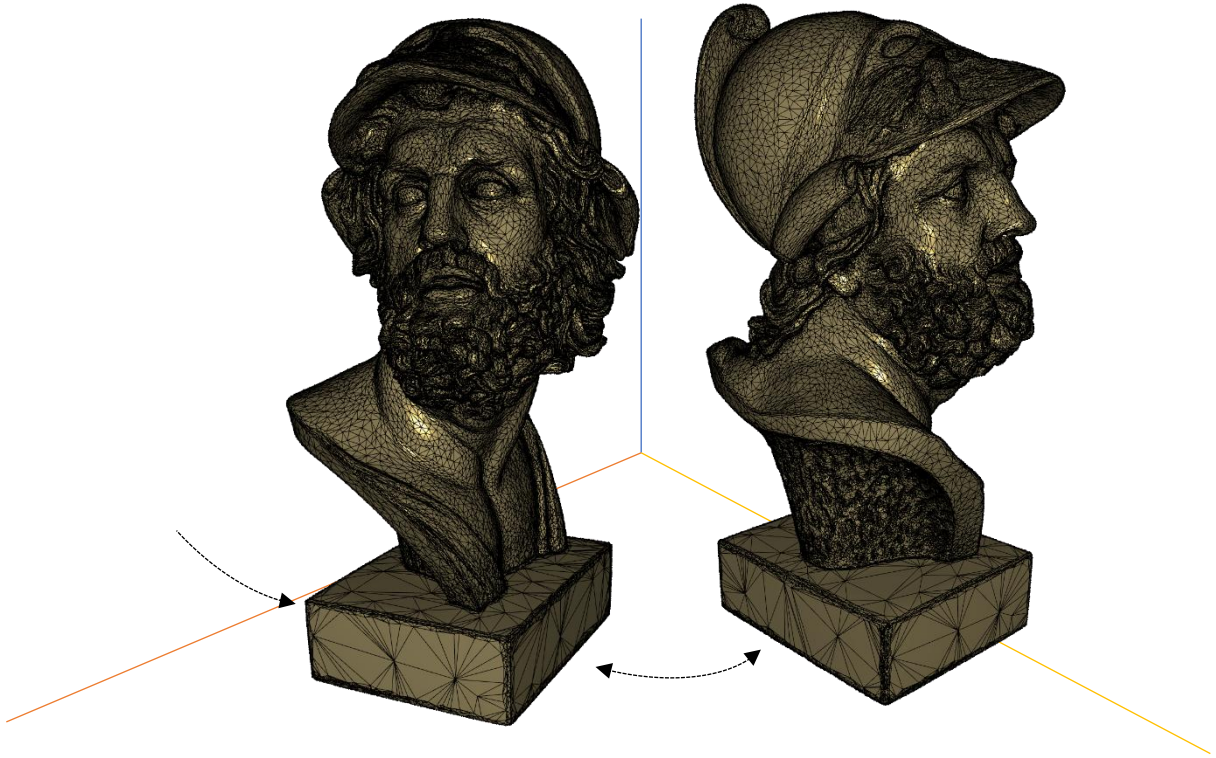
Ejercicios

- **Ejercicio A.**
  - Rotación de modelo en los ejes X, Y y Z.
  - Debemos controlar la posición del objeto para saber dónde se encuentra tras una rotación.

$$\begin{pmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{pmatrix}$$

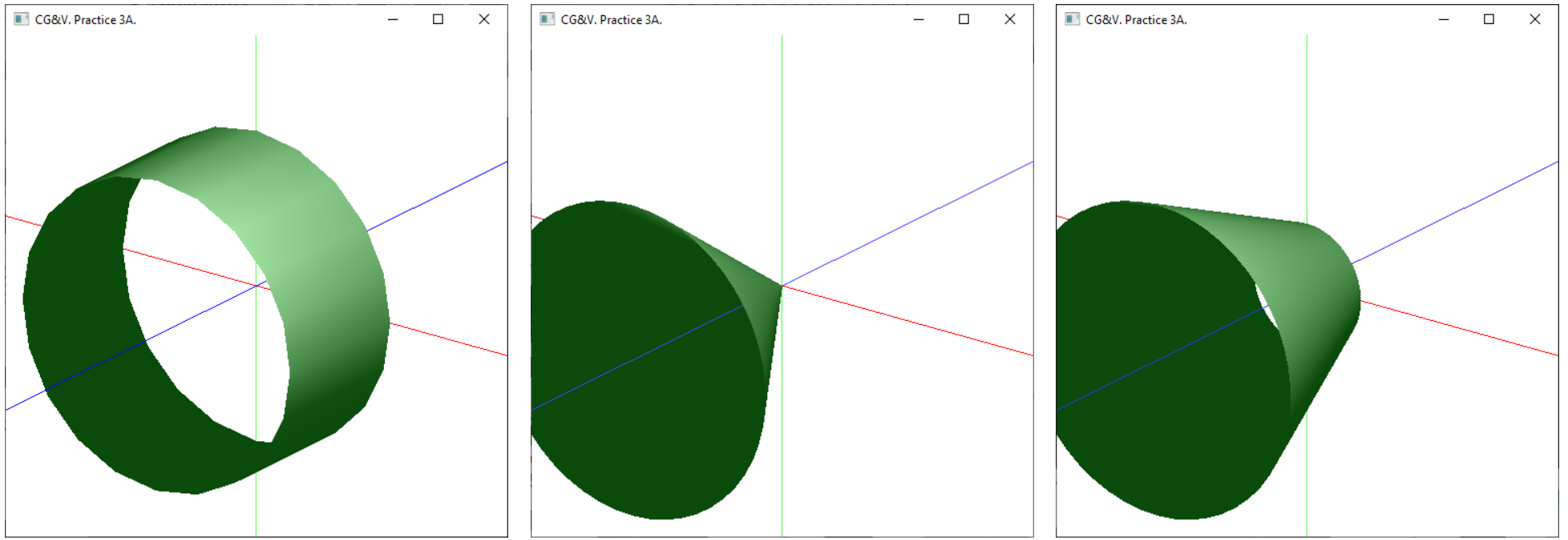
Rotación alrededor del eje Y

https://www.shadertoy.com/view/cd3cRj



Ejercicios

- **Ejercicio A.**
  - `gluCylinder(GLQuadric* obj, baseRadius, topRadius, height, slices, stacks)`



Ejercicios

- **Ejercicio A.**
  - `GLUquadric *obj = gluNewQuadric ();`
  - `gluQuadricDrawStyle(obj, GLU_LINE; GLU_POINT; GLU_SILHOUETTE; GLU_FILL);`
    - `gluSphere(obj, radius, nlong, nlat);`

      $x^2 + y^2 + z^2 = r$

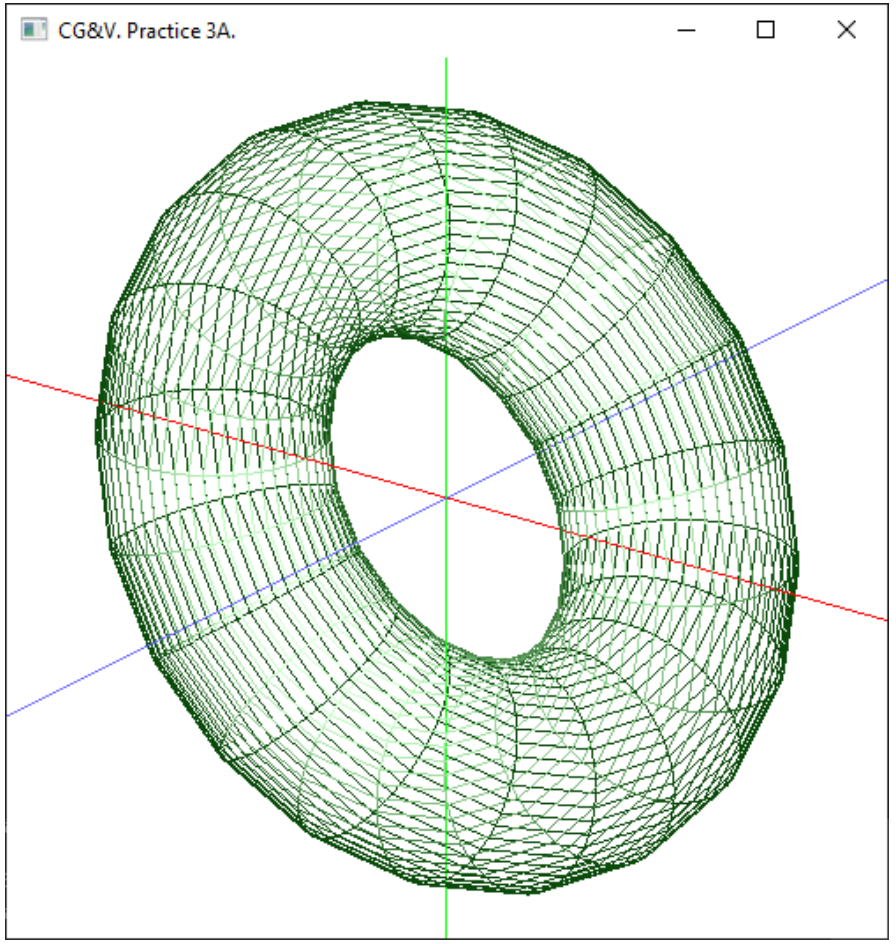
    - `gluCylinder(obj, radius_base, radius_end, height, slices, stacks);`

      $x^2 + z^2 = r$
  - `gluDeleteQuadric(name);`

Ejercicios

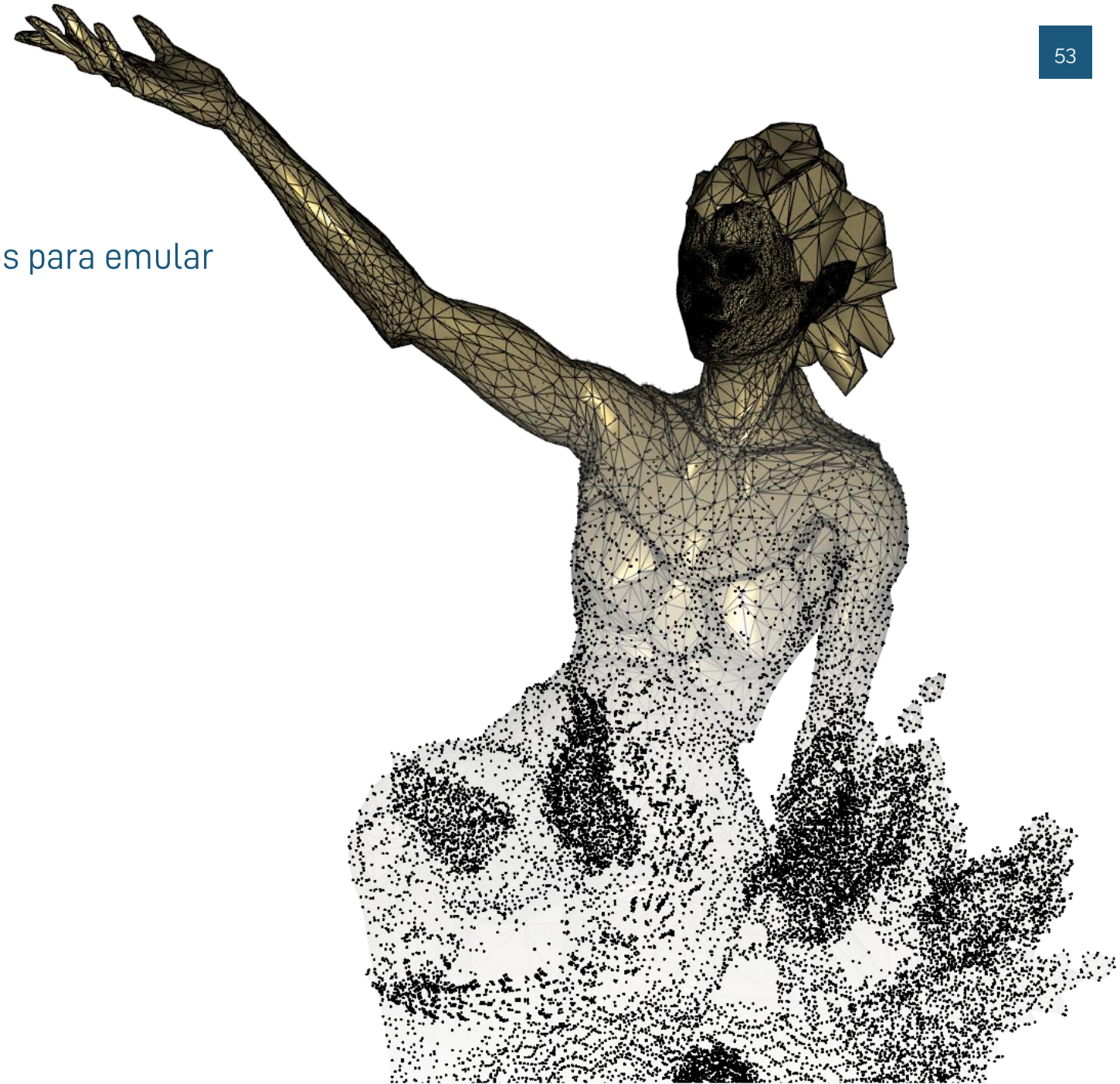
- **Ejercicio A.**
  - `glutSolidSphere(radius, nlong, nlat);`
  - `glutSolidCone(rbase, altura, nlong, nlat);`
  - `glutSolidTorus(rcircle, raxial, nconcen, nsec);`
  - `glutSolidCube(edge);`
  - `...`

  - `glutWireSphere(radius, nlong, nlat);`
  - `glutWireCone(rbase, altura, nlong, nlat);`
  - `glutWireTorus(rcircle, raxial, nconcen, nsec);`
  - `glutWireCube(edge);`
  - `...`



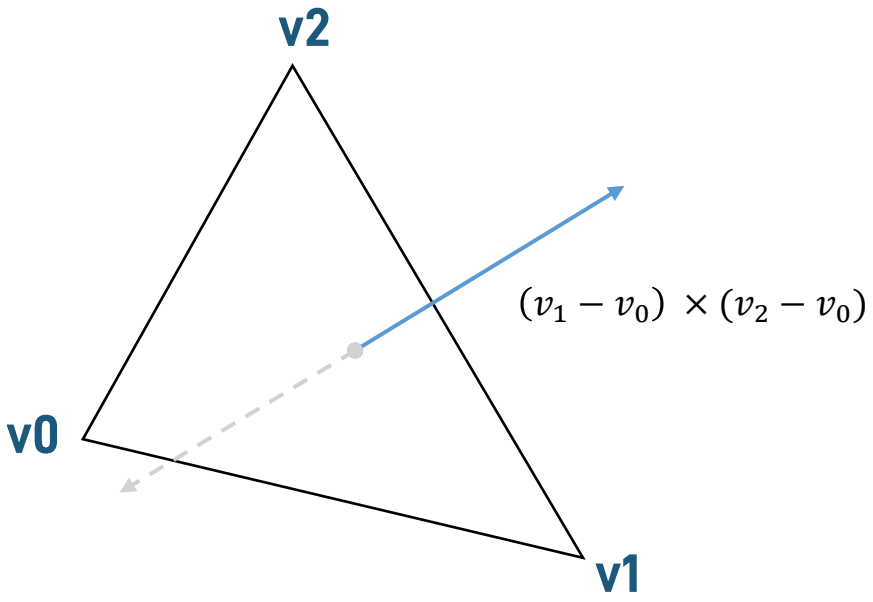
Ejercicios

- **Ejercicio B.**
  - Construcción de una malla de triángulos para emular el comportamiento de `gluCylinder()`.
  - Vértices y vectores normales.



Ejercicios

- **Ejercicio B.**
  - Construcción de una malla de triángulos para emular el comportamiento de `gluCylinder()`.
  - Vértices y vectores normales.
  - ***Counterclockwise order*:**
    - Contrario a la rotación de las agujas de un reloj: v_0, v_1, v_2.



Ejercicios

- **Representación de mallas de triángulos.**

position

float x 3 x n

x_1 y_1 z_1	x_2 y_2 z_2	x_3 y_3 z_3	x_4 y_4 z_4	...

normal

float x 3 x n

x_1 y_1 z_1	x_2 y_2 z_2	x_3 y_3 z_3	x_4 y_4 z_4	...

Ejercicios

- **Representación de mallas de triángulos.**

position

float x 3 x n

x_1	y_1	z_1	x_2	y_2	z_2	...

normal

float x 3 x n

x_1	y_1	z_1	x_2	y_2	z_2	...

Ejercicios

- **Ejercicio B.**
  - Representación de mallas de triángulos en OpenGL.

```
glBegin(GL_TRIANGLES);
    glVertex3f(x_p1, y_p1, z_p1);
    glVertex3f(x_p2, y_p2, z_p2);
    glVertex3f(x_p3, y_p3, z_p3);
    glVertex3f(x_p4, y_p4, z_p4);
    glVertex3f(x_p5, y_p5, z_p5);
    glVertex3f(x_p6, y_p6, z_p6);
glEnd();
```

Ejercicios

- **Ejercicio B.**
  - Representación de mallas de polígonos en OpenGL.

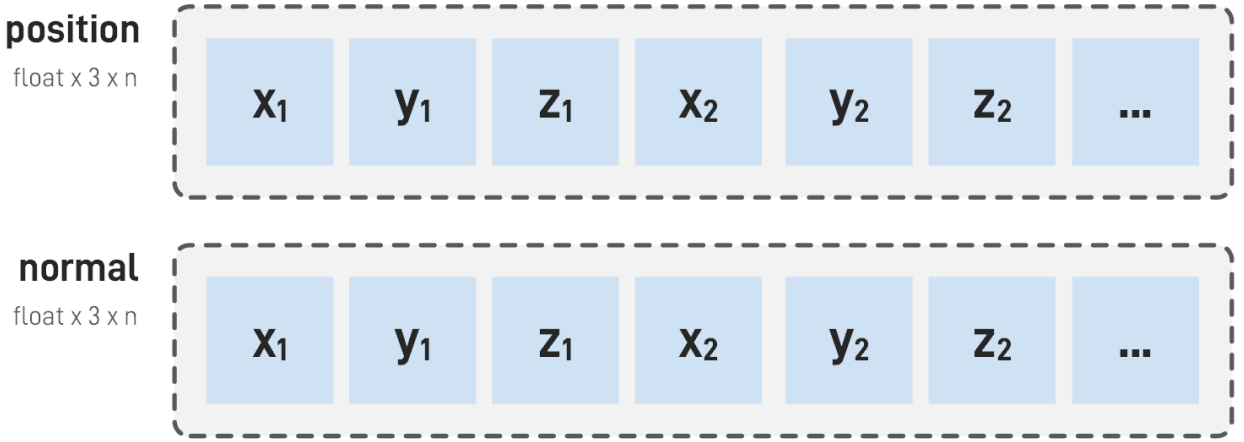
```
glBegin(GL_POLYGON);
    glVertex3f(x_p1, y_p1, z_p1);
    glVertex3f(x_p2, y_p2, z_p2);
    glVertex3f(x_p3, y_p3, z_p3);
    glVertex3f(x_p4, y_p4, z_p4);
glEnd();
```

Ejercicios

- **Ejercicio B.**
  - Representación de mallas de triángulos en OpenGL.
    - **Vertex array**: menor número de llamadas de dibujo -> más eficiente.

```
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(dimensiones, tipo_coordenada, offset, array_pointer);
```

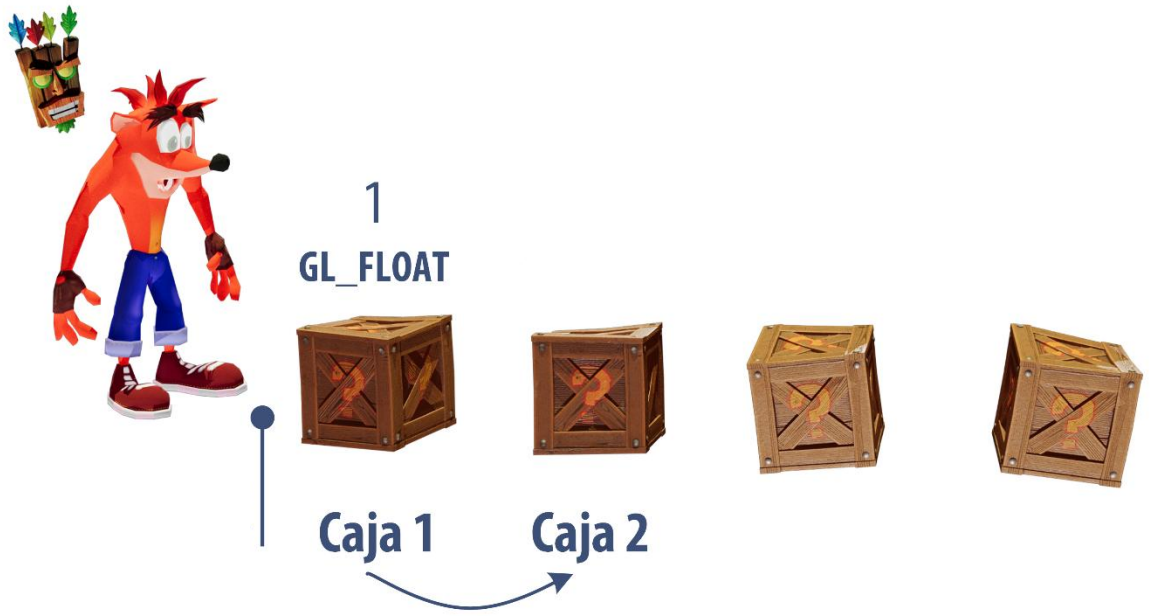
    - ¿2D, 3D?
    - GL_INT, GL_FLOAT...
    - *Offset* entre dos vértices consecutivos.
    - Puntero a *array*.



Ejercicios

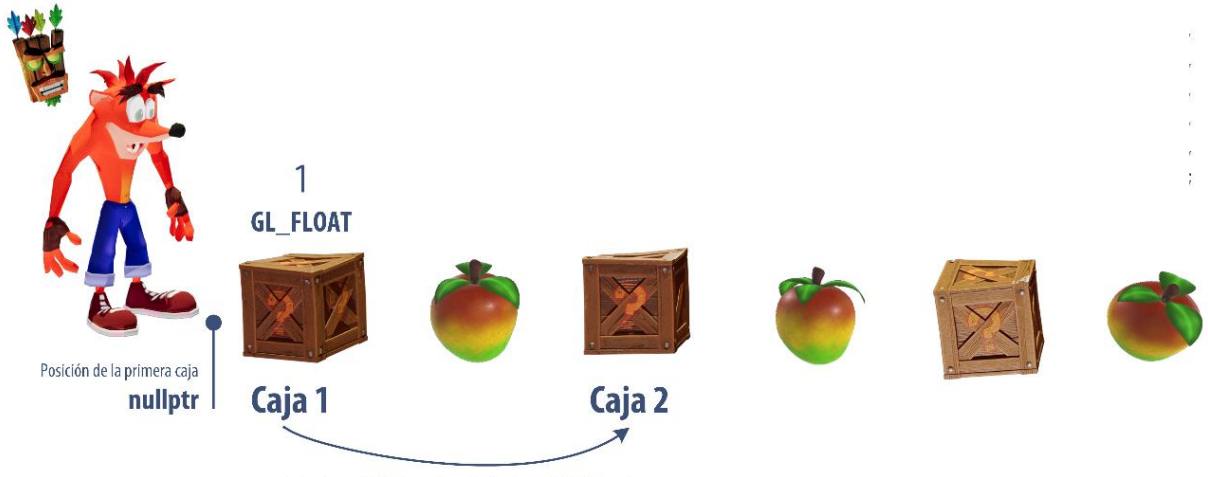
- **Ejercicio B.**
  - Representación de mallas de triángulos en OpenGL.
    - **Vertex array**: menor número de llamadas de dibujo -> más eficiente.

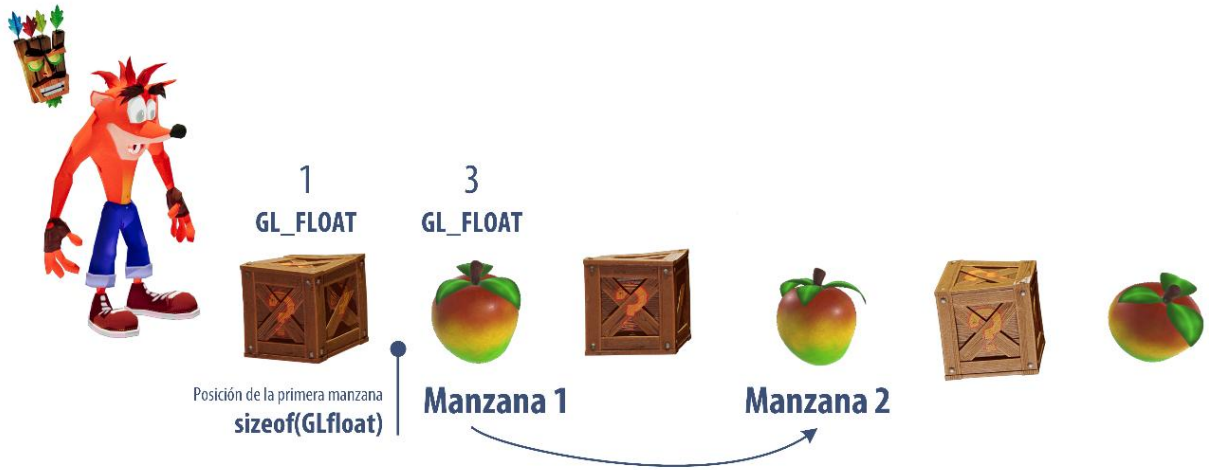
```
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(dimensiones, tipo_coordenada, offset, array_pointer);
```



Ejercicios

- **Ejercicio B.**
  - Representación de mallas de triángulos en OpenGL.
    - **Vertex array**: menor número de llamadas de dibujo -> más eficiente.





Ejercicios

- **Ejercicio B.**
  - Representación de mallas de triángulos en OpenGL.
    - **Vertex array**: menor número de llamadas de dibujo -> más eficiente.

```
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(dimensiones, tipo_coordenada, offset, array_pointer);
glDrawElements(primitive, num_indices, formato_indice, array_pointer);
```

    - GL_TRIANGLES, GL_QUADS, etc.
    - GL_UNSIGNED_SHORT, GL_UNSIGNED_INT, GL_UNSIGNED_BYTE, etc.

Ejercicios

- **Ejercicio B.**
  - Representación de mallas de triángulos en OpenGL.
    - **Vertex array**: menor número de llamadas de dibujo -> más eficiente.

```
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(dimensiones, tipo_coordenada, offset, array_pointer);
glDrawElements(primitive, num_indices, formato_indice, array_pointer);
glDisableClientState(GL_VERTEX_ARRAY);
```

Ejercicios

- **Ejercicio B.**
  - Representación de mallas de triángulos en OpenGL.
    - **Vertex array**: menor número de llamadas de dibujo -> más eficiente.

```
GLfloat vertices[][3] = {
        {0.0, 1.0, 0.0}, {0.0, 0.0, 0.0}, {1.0, 0.0, 0.0}, {1.0, 1.0, 0.0}, {2.0, 0.0, -1.0}
};
GLubyte indices[] = { 0,1,2, 0,2,3, 3,2,4 };

glEnableClientState(GL_VERTEX_ARRAY);
        glVertexPointer(3, GL_FLOAT, 0, vertices);
        glDrawElements(GL_TRIANGLES, 9, GL_UNSIGNED_BYTE, indices);
glDisableClientState(GL_VERTEX_ARRAY);
```

Ejercicios

- **Ejercicio B.**
  - Representación de mallas de triángulos en OpenGL.

```
glNormal3f(n_x, n_y, n_z);
glBegin(GL_TRIANGLES);
    glVertex3f(x_p1, y_p1, z_p1);
    glVertex3f(x_p2, y_p2, z_p2);
    glVertex3f(x_p3, y_p3, z_p3);
glEnd();
```

Ejercicios

- **Ejercicio B.**
  - Representación de mallas de triángulos en OpenGL.

```
glBegin(GL_TRIANGLES);
    glNormal3f(n_x1, n_y1, n_z1);
    glVertex3f(x_p1, y_p1, z_p1);
    glNormal3f(n_x2, n_y2, n_z2);
    glVertex3f(x_p2, y_p2, z_p2);
    glNormal3f(n_x3, n_y3, n_z3);
    glVertex3f(x_p3, y_p3, z_p3);
glEnd();
```

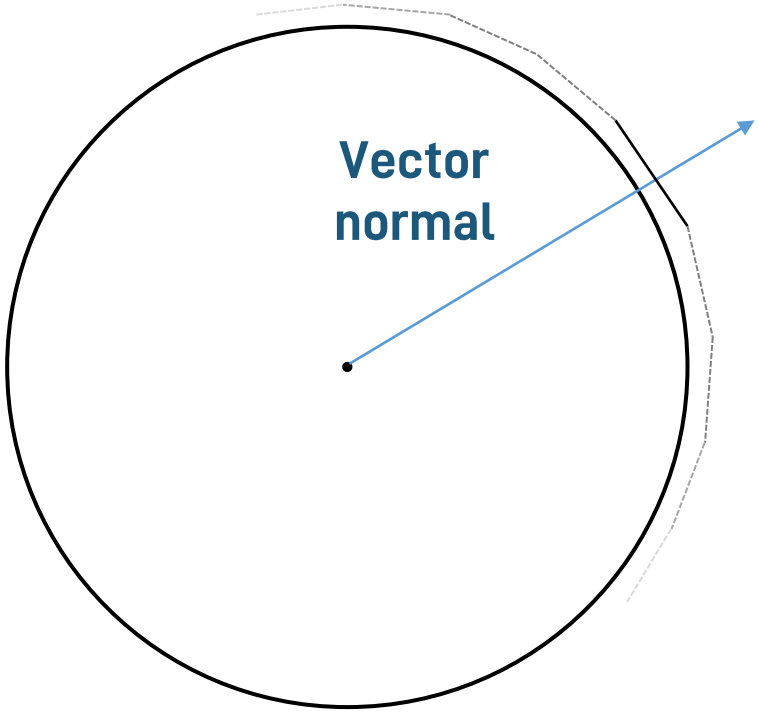
Ejercicios

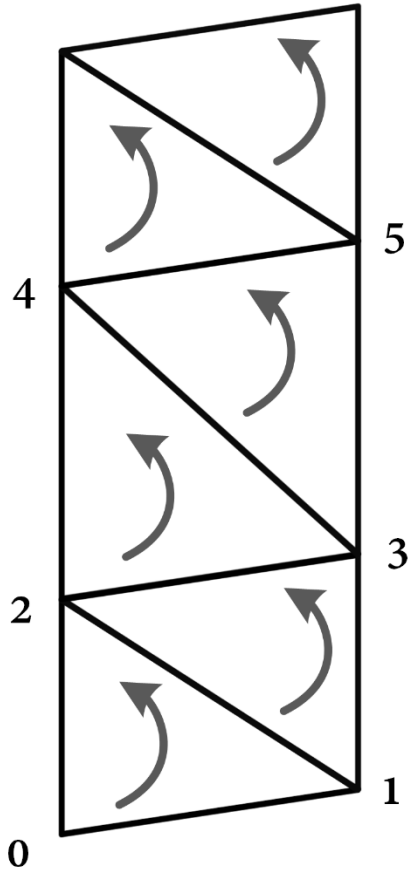
Ejercicio B.

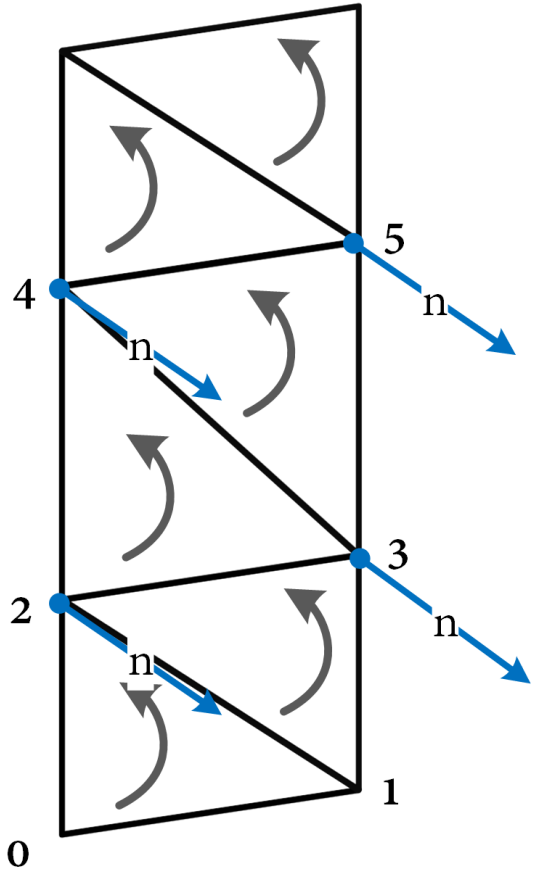
```
GLfloat vertices[][3] = {
        {0.0, 1.0, 0.0}, {0.0, 0.0, 0.0}, {1.0, 0.0, 0.0}, {1.0, 1.0, 0.0}, {2.0, 0.0, -1.0}
};
GLfloat normal_vect[][3] = {
        {0.0, 0.0, 1.0}, {0.0, 0.0, 1.0}, {0.0, 0.0, 1.0}, {0.0, 0.0, -1.0},{1.0, 0.0, 0.0}
};
GLubyte indices[] = { 0,1,2, 0,2,3, 3,2,4 };

glEnableClientState(GL_VERTEX_ARRAY);
        glVertexPointer(3, GL_FLOAT, 0, vertices);
        glEnableClientState(GL_NORMAL_ARRAY);
        glNormalPointer(GL_FLOAT, 0, normal_vect);
        glDrawElements(GL_TRIANGLES, 9, GL_UNSIGNED_BYTE, indices);
glDisableClientState(GL_VERTEX_ARRAY);
glDisableClientState(GL_NORMAL_ARRAY);
```

Ejercicios

- Ejercicio B y C.

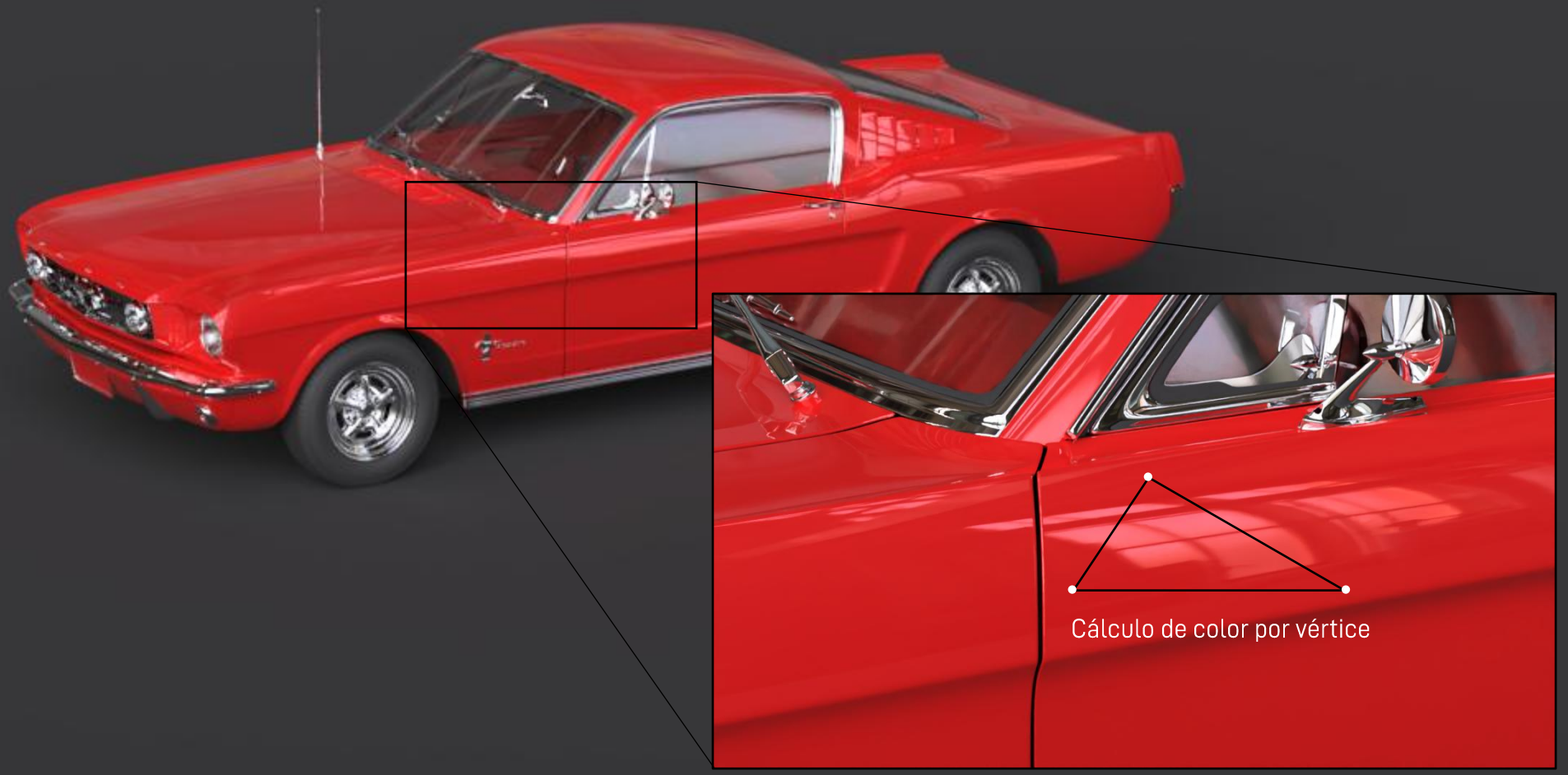






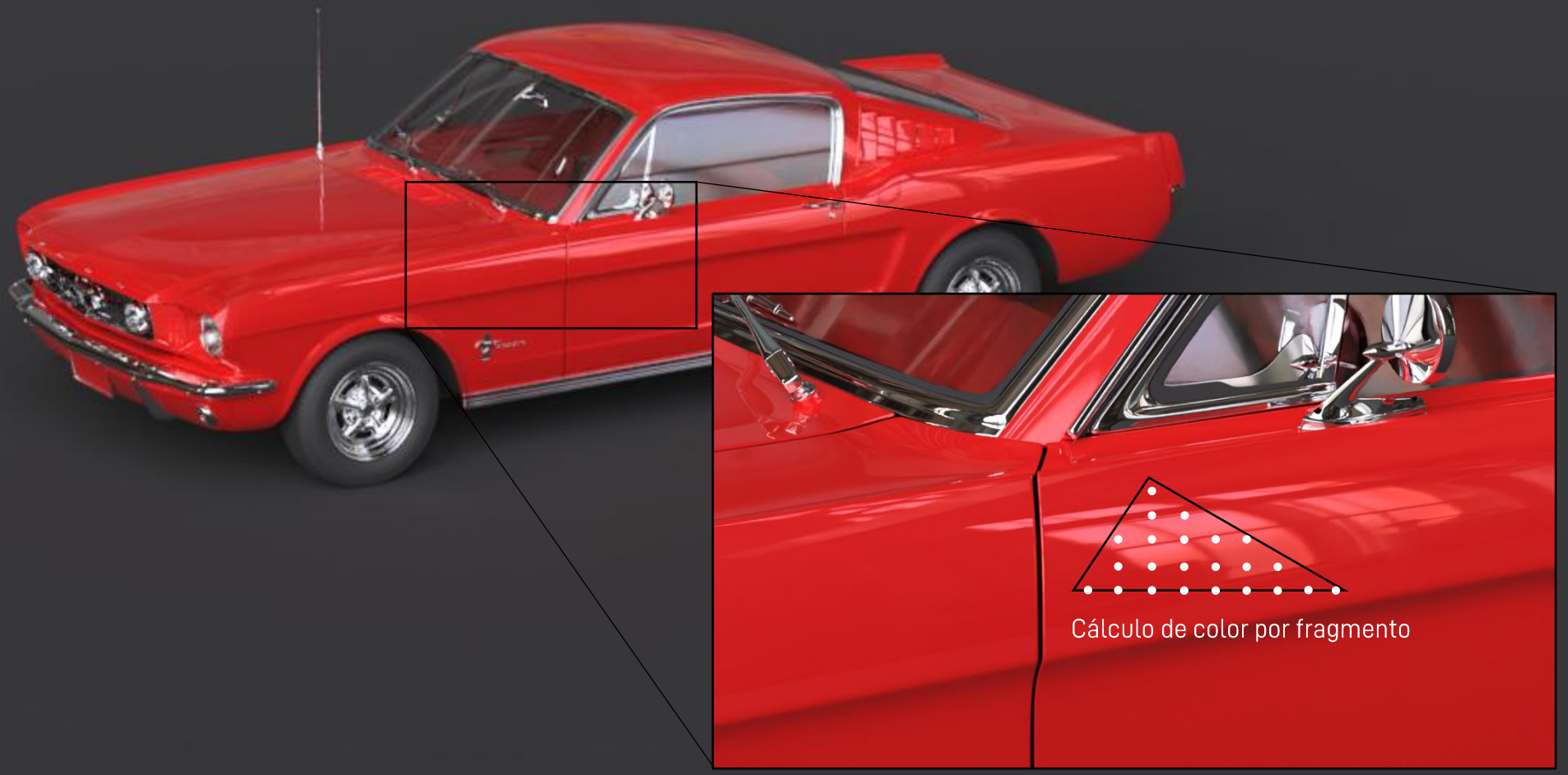
Ejercicios

- Gouraud shading: **GL_FLAT**.



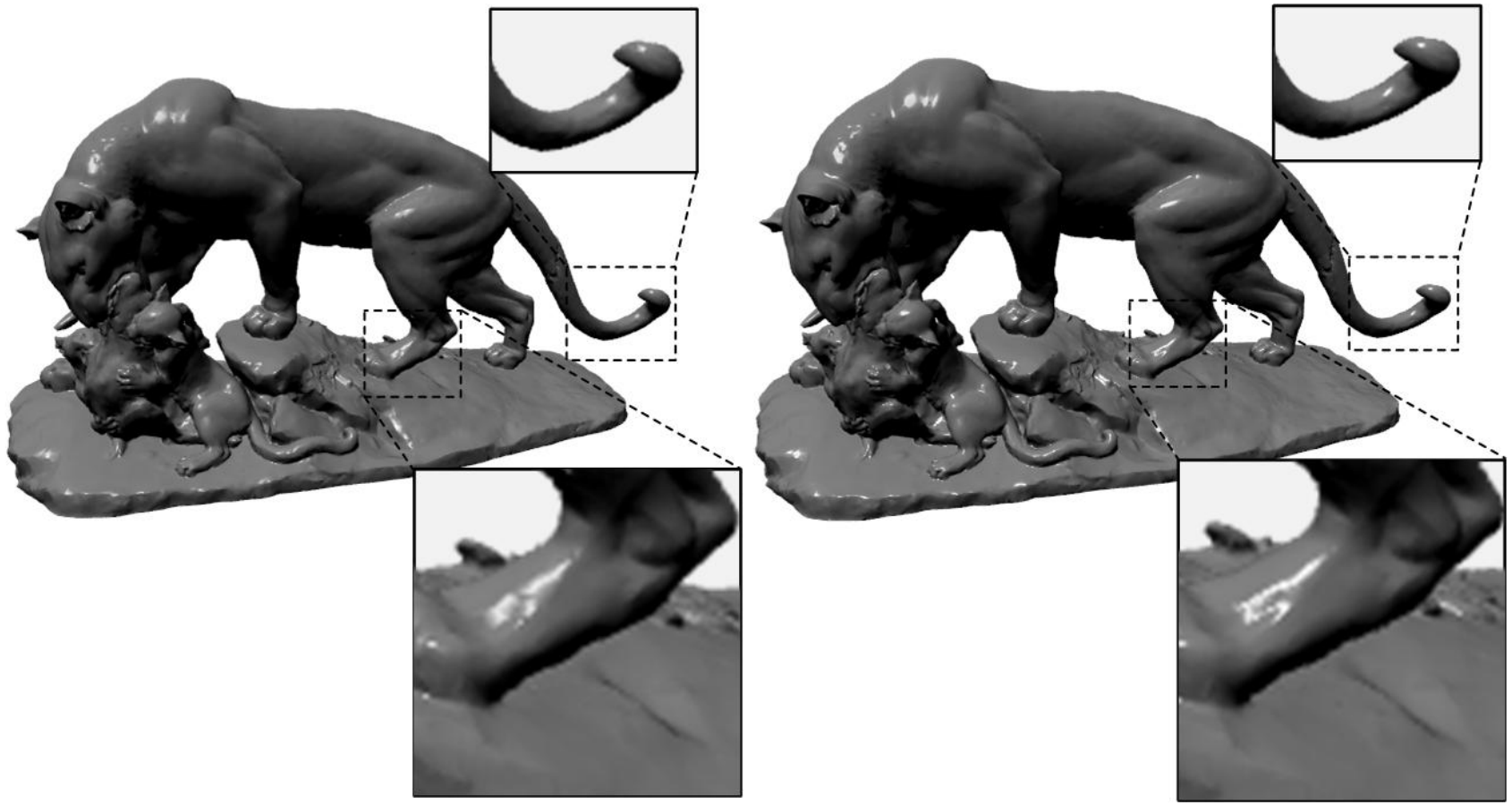
Ejercicios

- Phong shading: **GL_SMOOTH**.



Ejercicios

- **Gouraud vs Phong shading.**
  - Existen menos diferencias entre ambos métodos cuando la malla tiene muchos triángulos...



Ejercicios

- **Gouraud vs Phong shading.**
  - … y hay muchas diferencias cuando la malla tiene muy pocos triángulos.

Práctica 3b. Grafos de escena

Curso académico 2023-2024

Informática Gráfica y Visualización

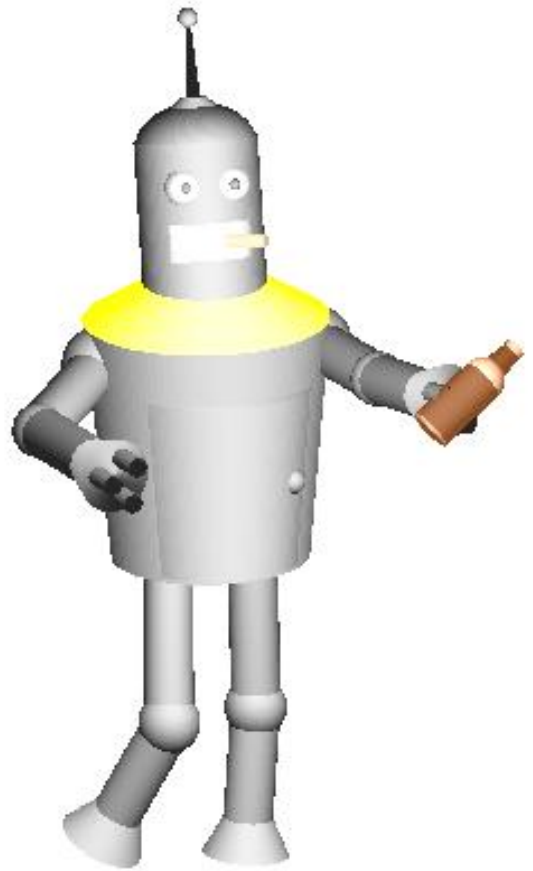
Grado en Ingeniería Informática

Estructura del proyecto

- **pr3b:** main.
- **igvInterfaz:** gestión de eventos y ventana.
- **igvEscena:** gestiona contenido presente en la escena.
- **igvPunto3D**: declaración de objetos tipo punto y vector.
- **igvCamara**: configuración de cámara.

Ejercicios

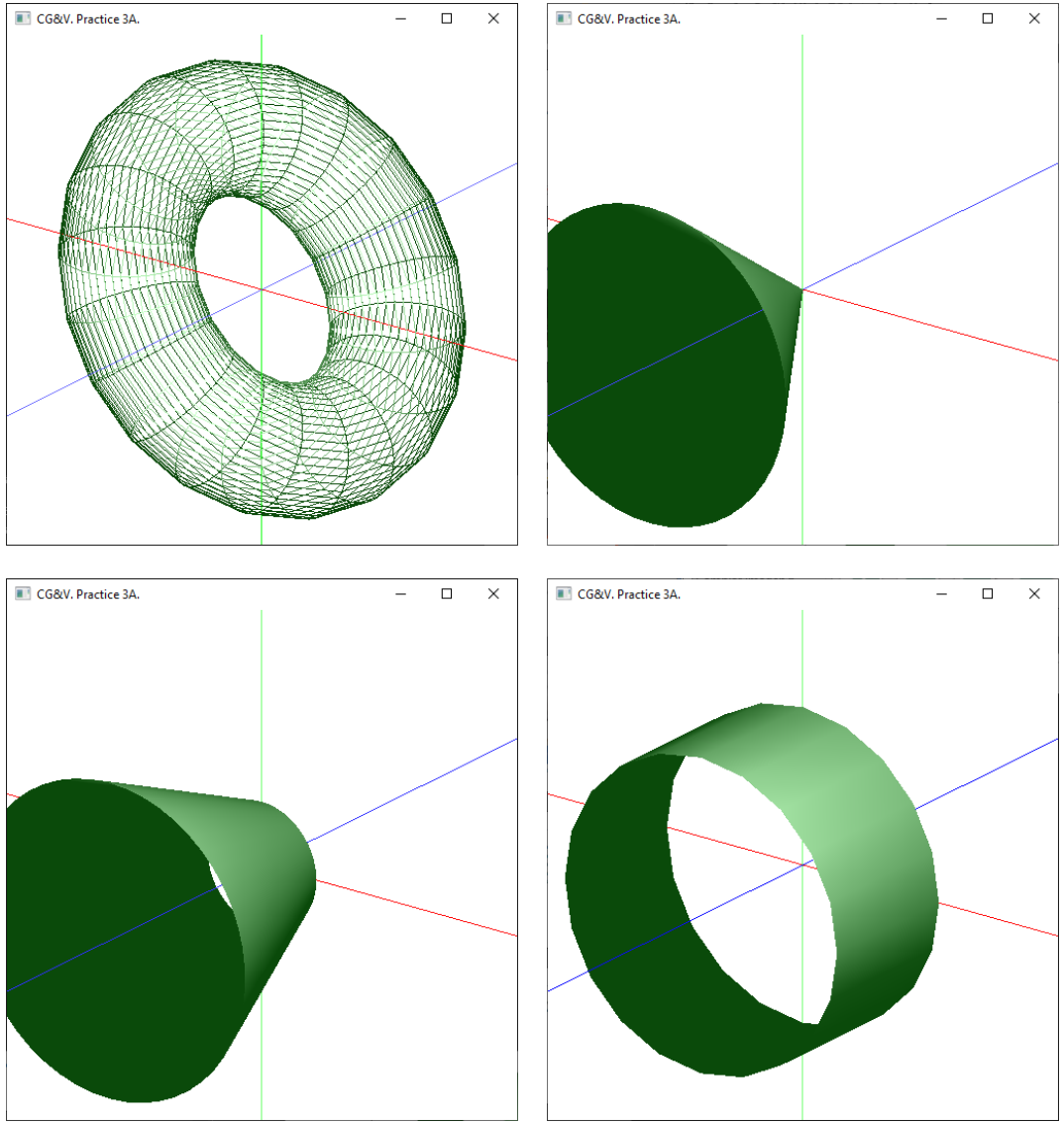
- **Ejercicio A.**
  - Definir grafo de escena con 4 niveles de nodos.
  - 3 grados de libertad.
    - 3 movimientos en elementos diferentes de la estructura.
  - Documentación de jerarquía, transformaciones y teclas asociadas.
  - Visualización de modelo diseñado.
    - *igvEscena::visualizar()*



Ejercicios

- **Ejercicio B.**
  - Visualización de modelo diseñado.
    - **Primitivas de OpenGL.**

```
glutSolidSphere(radius, nlong, nlat);
glutSolidCone(rbase, altura, nlong, nlat);
glutSolidTorus(rcircle, raxial, nconcen, nsec);
glutSolidCube(edge);
...

glutWireSphere(radius, nlong, nlat);
glutWireCone(rbase, altura, nlong, nlat);
glutWireTorus(rcircle, raxial, nconcen, nsec);
glutWireCube(edge);
...
```



Ejercicios

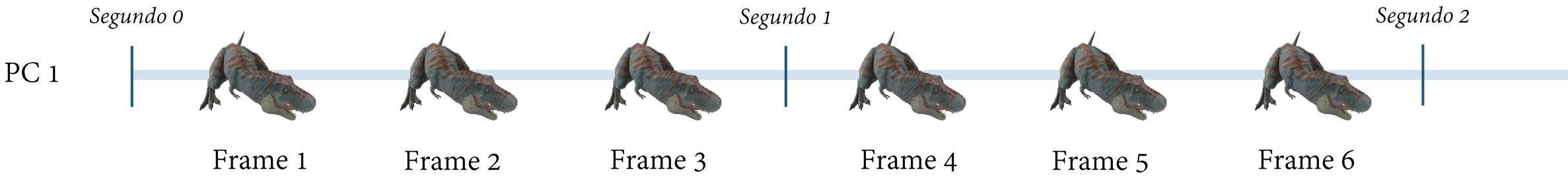
- **Ejercicio B.**
  - Visualización de modelo diseñado.
    - **Modelos externos (.obj, .stl, .fbx, .gltf, etc.)**
      - Contienen vértices, normales e índices de triángulos.
    - OBJ Loader, Assimp, etc.
      - *vcpkg install assimp*
      - *vcpkg install tinyobjloader*
      - *OBJ Loader (header file en Github).*

Ejercicios

- **Ejercicio B.**
  - Visualización de modelo diseñado.
    - **Modelos externos (.obj, .stl, .fbx, .gltf, etc.)**
      - Contienen vértices, normales e índices de triángulos.

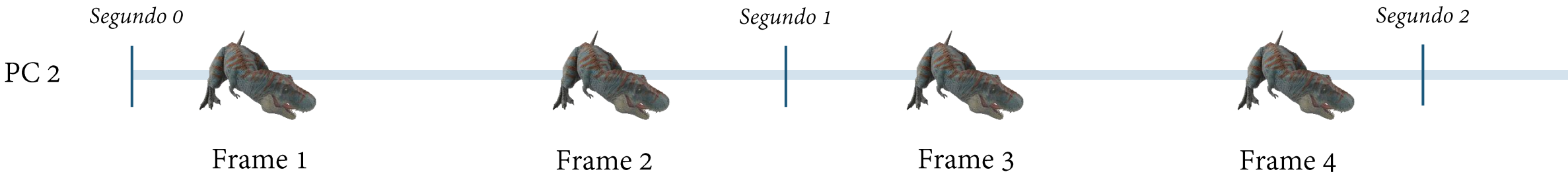
Ejercicios

- **Ejercicio C.**
  - Modificación interactiva de grados de libertad con teclado.
    - Por ejemplo, utilizando b/B.
- **Ejercicio D.**
  - Animación automática.
    - No todos los equipos funcionan con una misma tasa de *frames*.



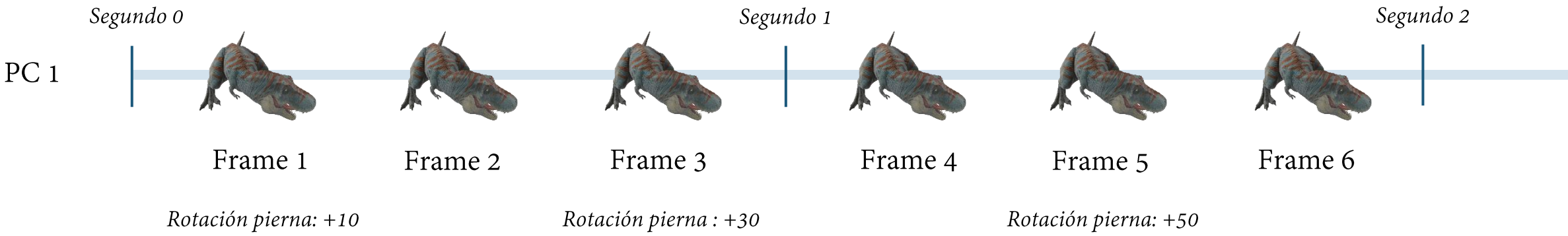
Ejercicios

- **Ejercicio C.**
  - Modificación interactiva de grados de libertad con teclado.
    - Por ejemplo, utilizando b/B.
- **Ejercicio D.**
  - Animación automática.
    - No todos los equipos funcionan con una misma tasa de *frames*.



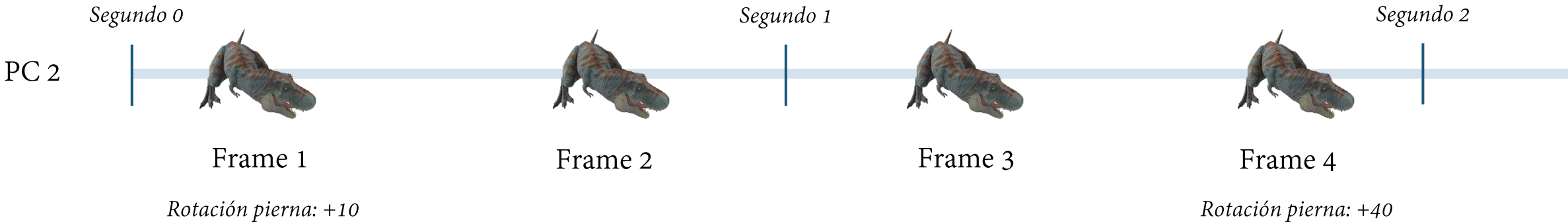
Ejercicios

- **Ejercicio C.**
  - Modificación interactiva de grados de libertad con teclado.
    - Por ejemplo, utilizando b/B.
- **Ejercicio D.**
  - Animación automática.
    - No todos los equipos funcionan con una misma tasa de *frames*.



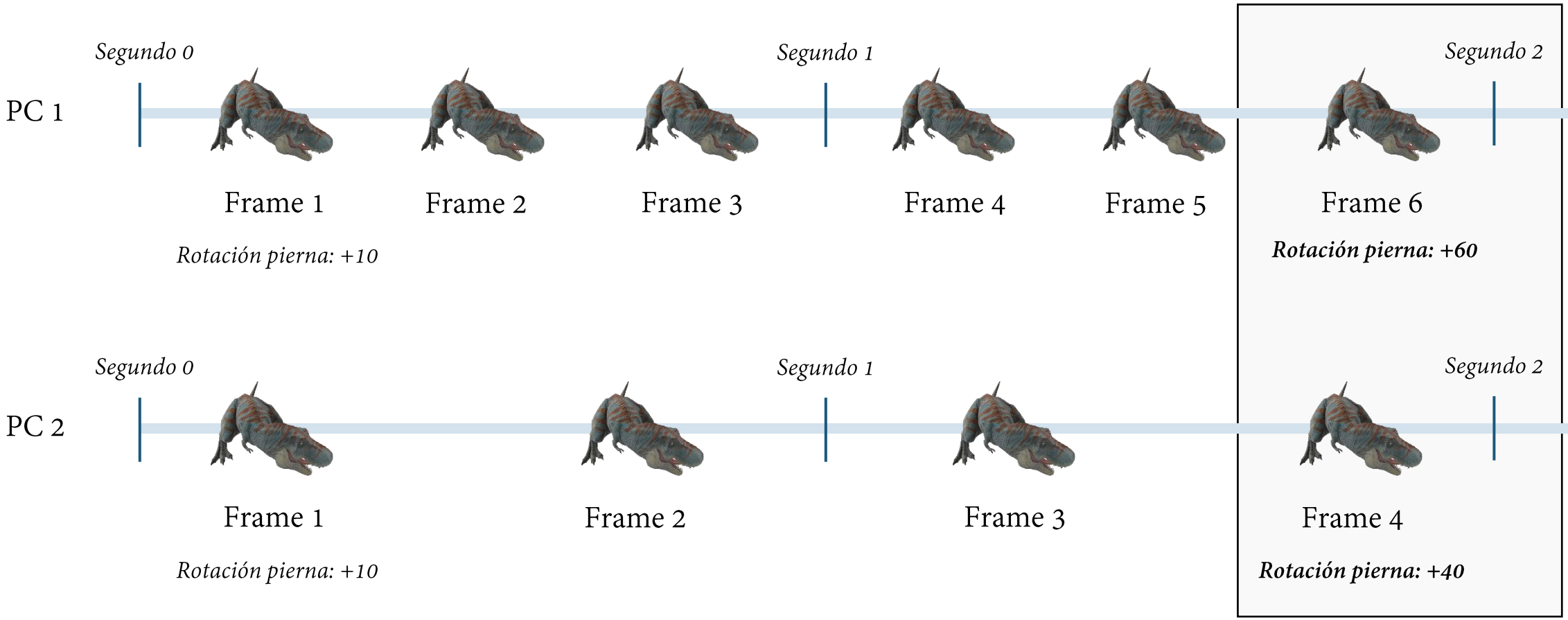
Ejercicios

- **Ejercicio C.**
  - Modificación interactiva de grados de libertad con teclado.
    - Por ejemplo, utilizando b/B.
- **Ejercicio D.**
  - Animación automática.
    - No todos los equipos funcionan con una misma tasa de *frames*.



Ejercicios

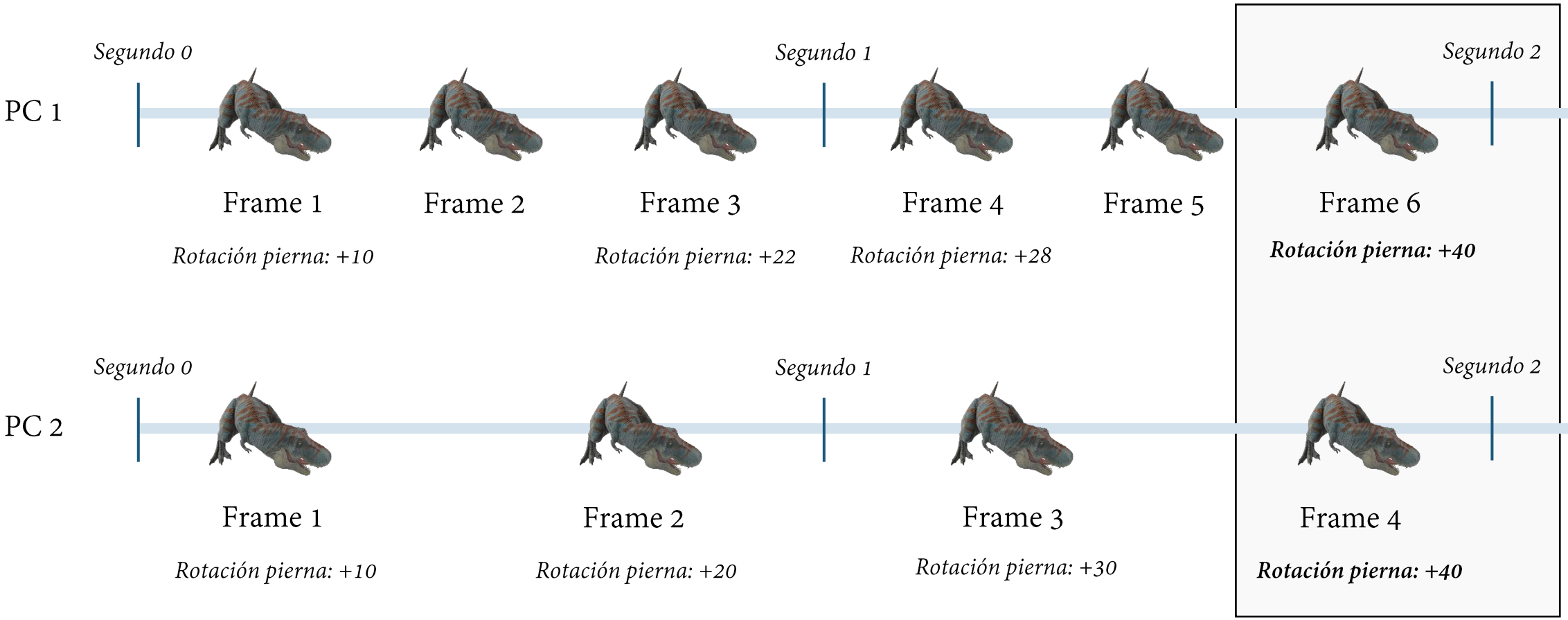
- **Ejercicio D.**
  - Animación automática.
    - No todos los equipos funcionan con una misma tasa de *frames*.



Ejercicios

Ejercicio D.

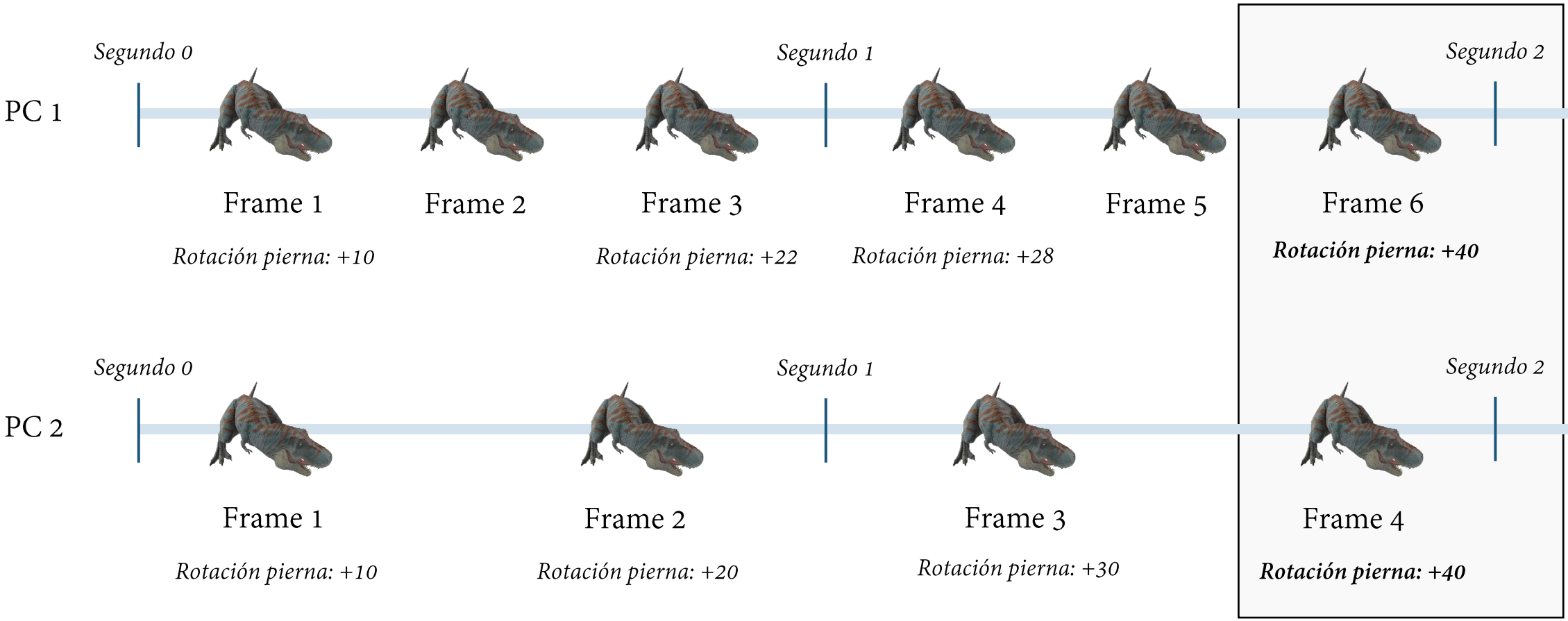
*Rotación: Ángulo actual + Ángulo * Tiempo transcurrido*



Ejercicios

Ejercicio D.

glutGet(GLUT_ELAPSED_TIME);



Práctica 3c. Selección

Curso académico 2023-2024

Informática Gráfica y Visualización

Grado en Ingeniería Informática

Estructura del proyecto

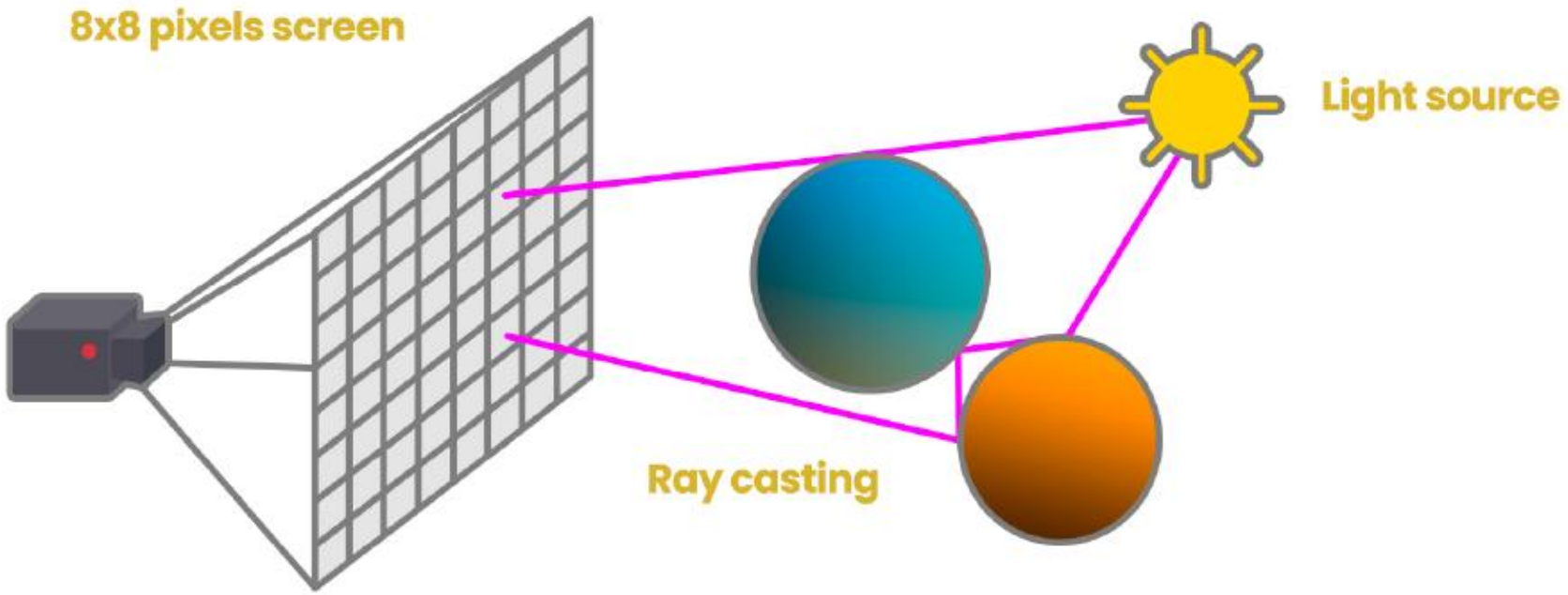
- **pr3c:** main.
- **igvInterfaz:** gestión de eventos y ventana.
- **igvEscena:** gestiona modelos en la escena y su dibujado.
- **igvPunto3D**: declaración de objetos tipo punto y vector.
- **igvCamara**: configuración de cámara.
- **igvCaja**: modelado de cada caja de la práctica 2.

Métodos de selección

Intersección de rayo

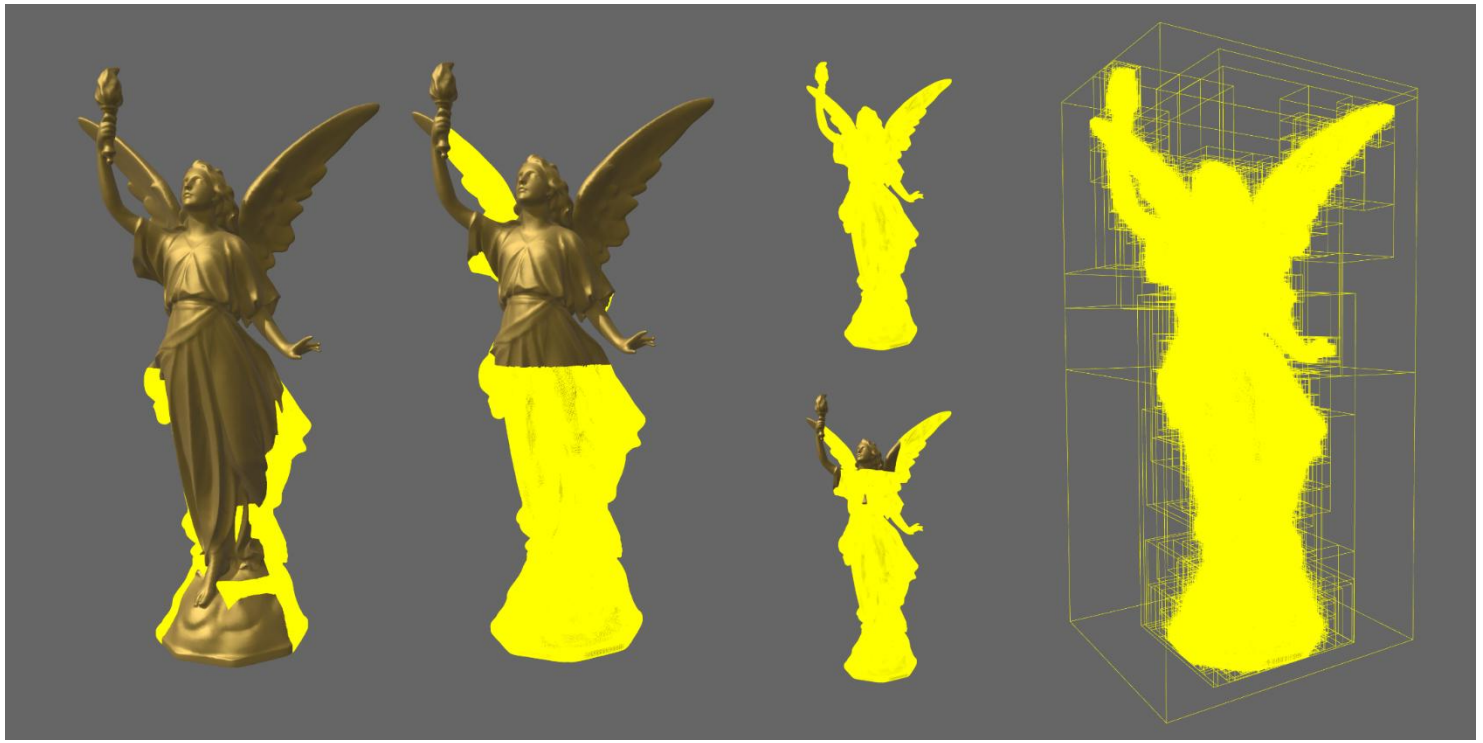
- **Eficiencia:** ●●●●●
- **Complejidad:** ●●●●●
- **Limitaciones:** X

*Requiere, no obstante, indexar la escena en estructuras de datos espaciales. La estructura más eficiente es la Jerarquía de Volúmenes Envolventes (Boundary Volume Hierarchy, BVH).



Métodos de selección

- Intersección de rayo
  - Eficiencia: ●●●●●
  - Complejidad: ●●●●●
  - Limitaciones: X



Métodos de selección

Intersección de rayo

- Se emplea para iluminar escenas mediante trazado de rayos.

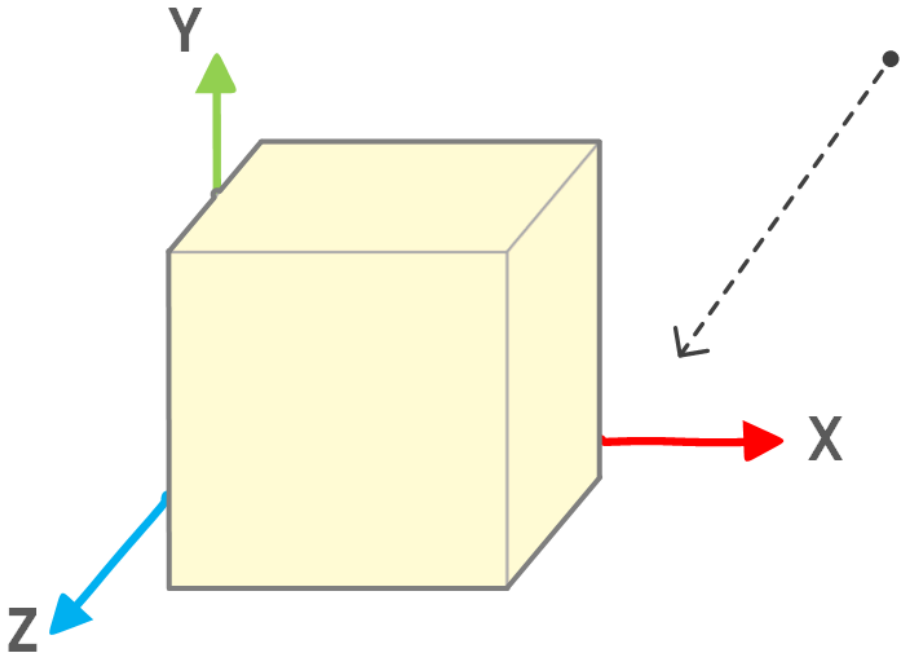
Métodos de selección

Bounding box

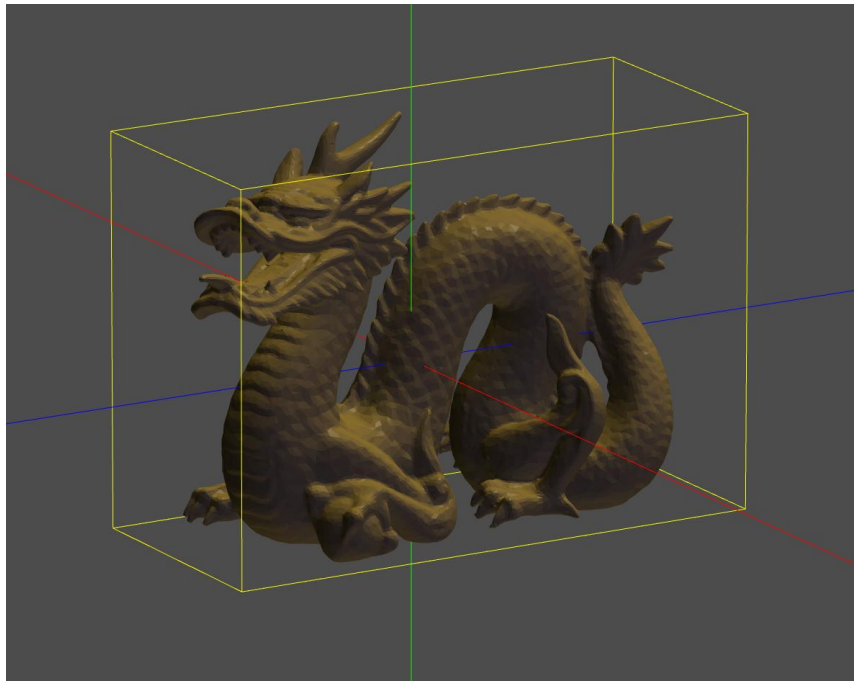
- **Eficiencia:** ●●●○○
- **Complejidad:** ●●●○○
- **Limitaciones:** **Muy poca precisión. Solapamiento de cajas envolventes**

*La eficiencia dependerá del tipo de objeto envolvente. Con una caja será muy eficiente, pero no con una malla de triángulos (por ejemplo, una envolvente convexa).

*Ídem para complejidad. Depende del objeto envolvente.



Axis-aligned bounding box (AABB)



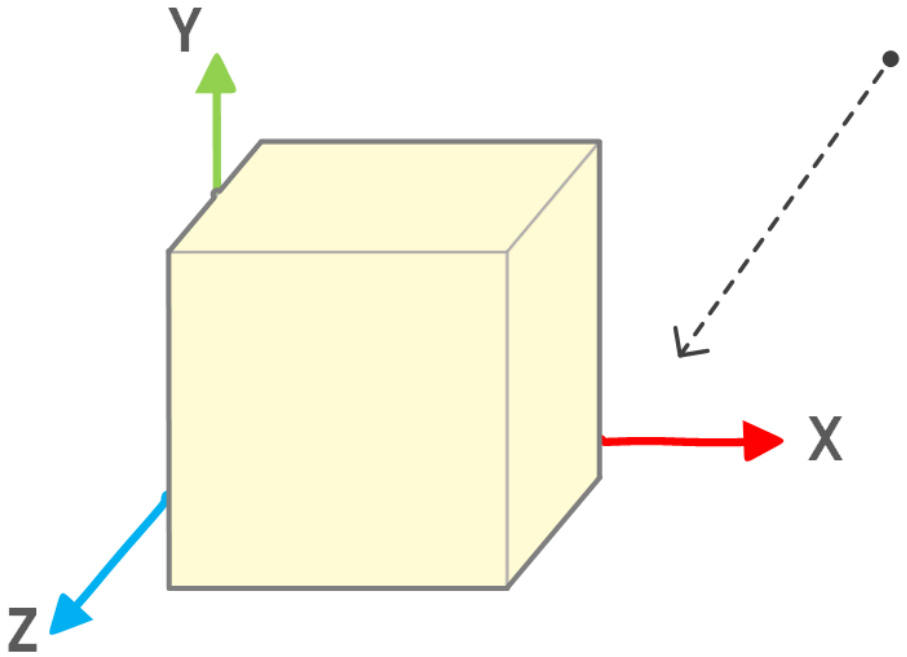
Métodos de selección

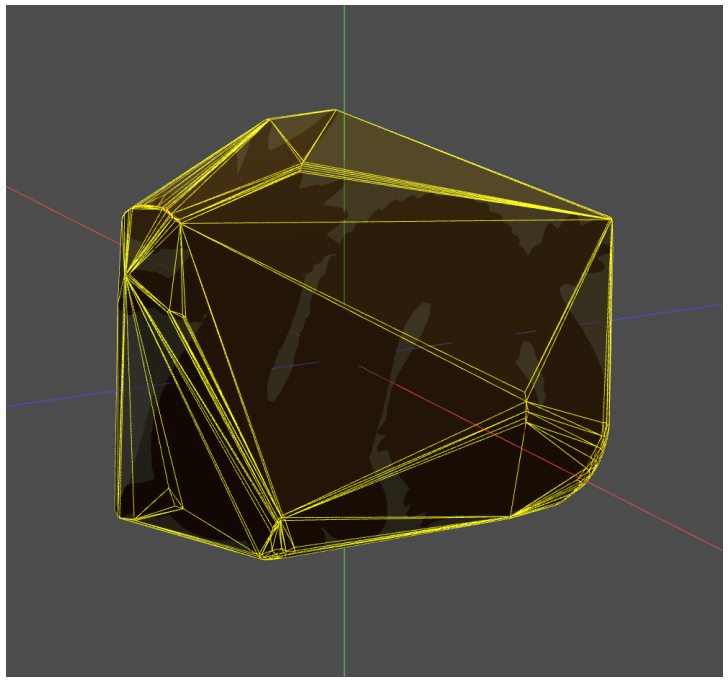
Bounding box

- **Eficiencia:** ●●●○○
- **Complejidad:** ●●●○○
- **Limitaciones:** **Muy poca precisión. Solapamiento de cajas envolventes**

*La eficiencia dependerá del tipo de objeto envolvente. Con una caja será muy eficiente, pero no con una malla de triángulos (por ejemplo, una envolvente convexa).

*Ídem para complejidad. Depende del objeto envolvente.





Envolvente convexa (convex hull)

Métodos de selección

Lista de impactos

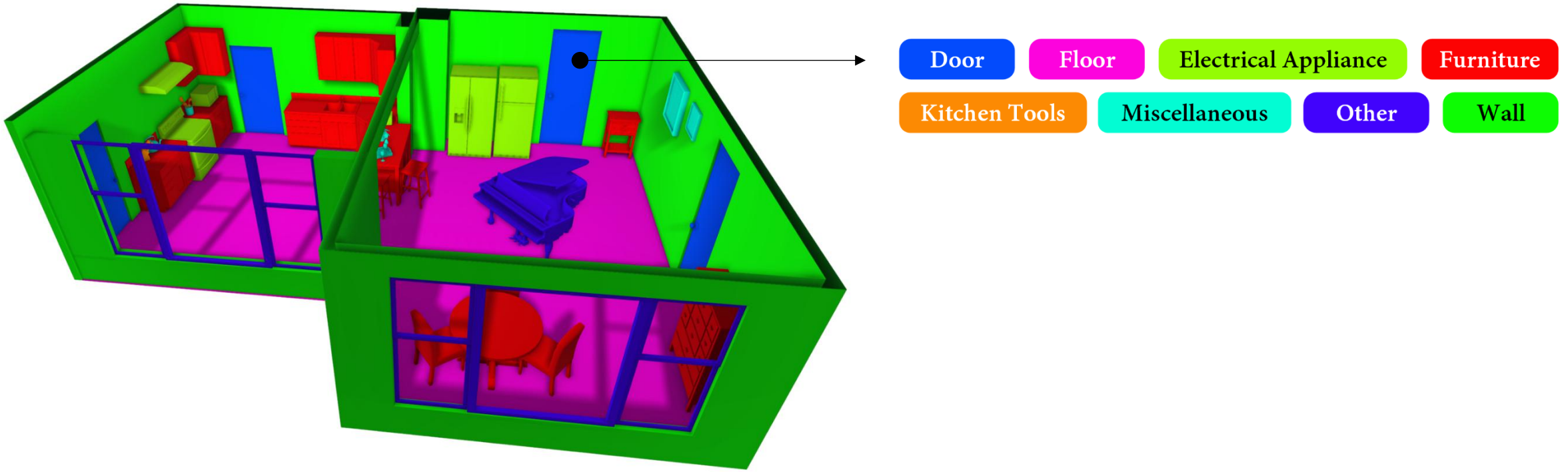
- **Eficiencia:** ●●○○○
- **Complejidad:** ●●●○○
- **Limitaciones:** Redibujado. Devuelve una lista y no el impacto directo. Mejor con soporte de librería

Mano: z de 0.2 a 0.8

Fondo: z de 2 a 2.3

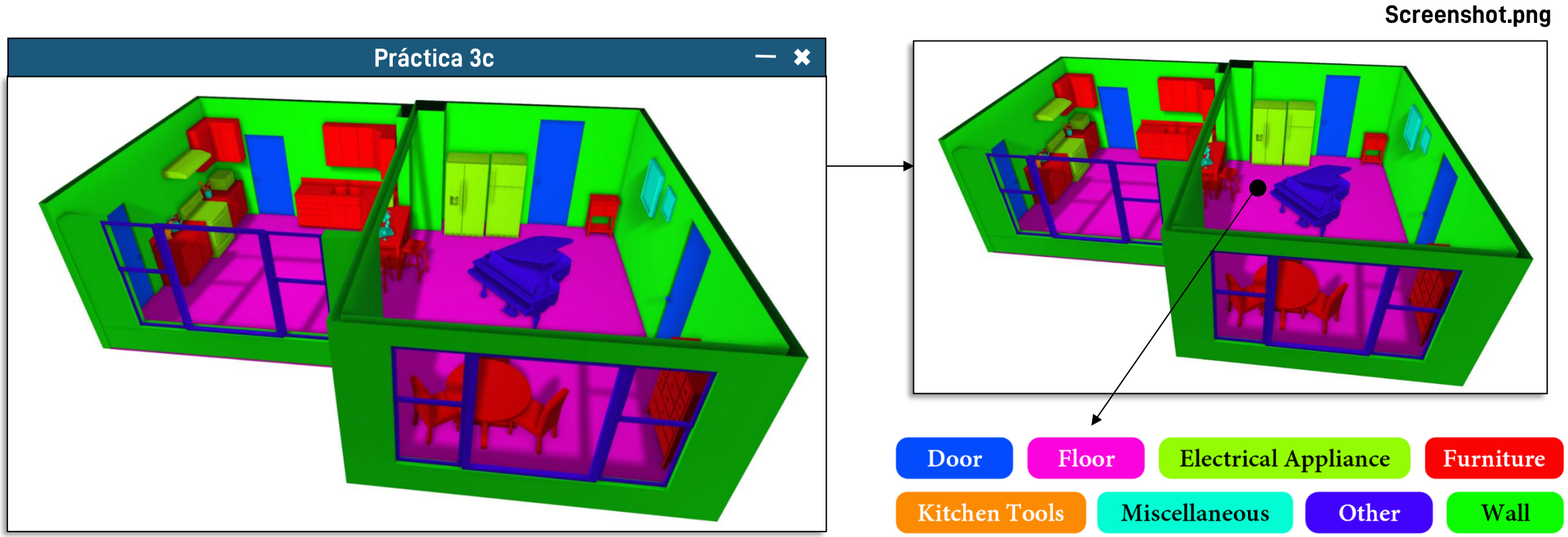
Métodos de selección

- **Color buffer**
  - **Eficiencia:** ●●●○○
  - **Complejidad:** ●●○○○
  - **Limitaciones:** Límite de objetos (256^3, R x G x B). Requiere redibujado



Métodos de selección

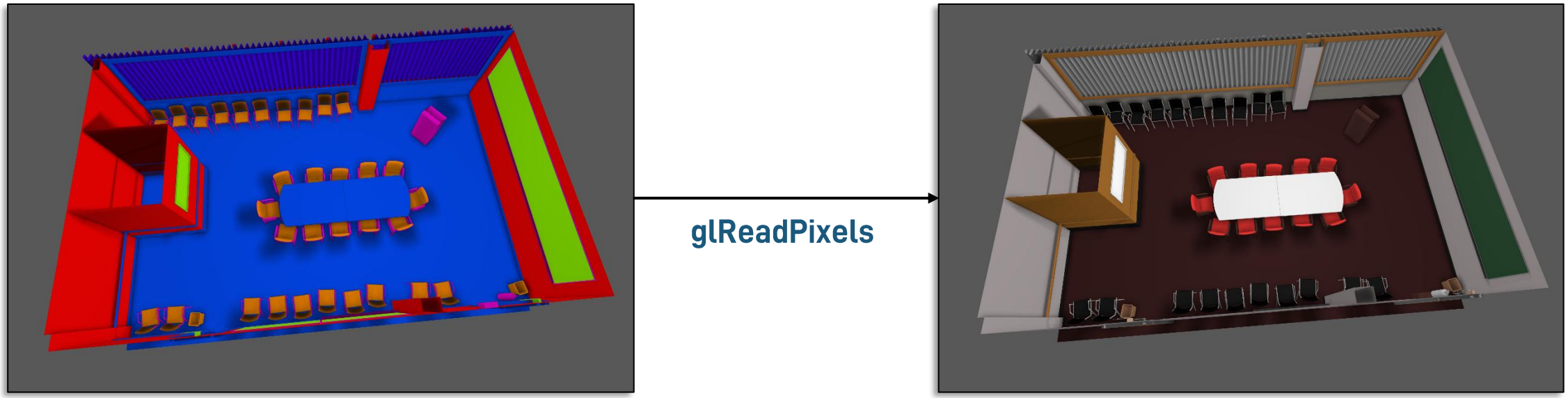
- **Color buffer**
  - Eficiencia: ●●●○○
  - Complejidad: ●●○○○
  - Limitaciones: Límite de objetos (256^3, R x G x B). Requiere redibujado



Métodos de selección

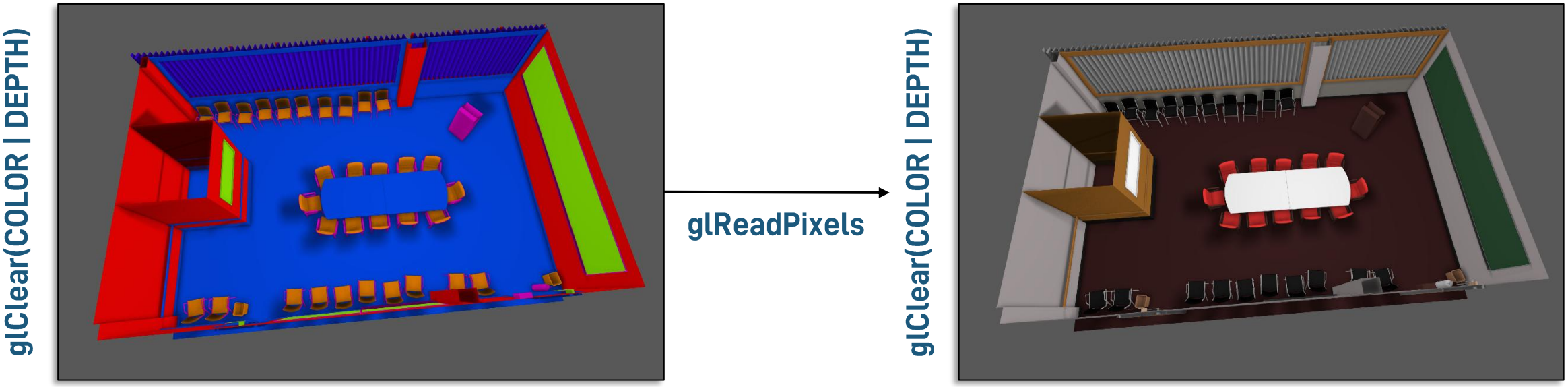
- **Color buffer**
  - **Eficiencia:** ●●●○○
  - **Complejidad:** ●●○○○
  - **Limitaciones:** Límite de objetos (256^3, R x G x B). Requiere redibujado

Modo selección



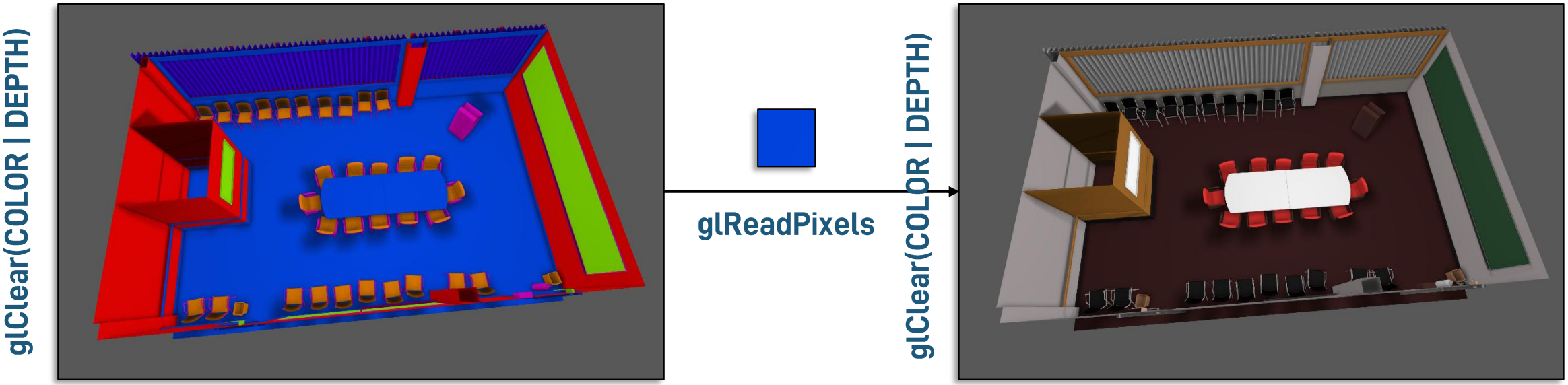
Métodos de selección

- **Color buffer**
  - **Eficiencia:** ●●●○○
  - **Complejidad:** ●●○○○
  - **Limitaciones:** Límite de objetos (256^3, R x G x B). Requiere redibujado



Métodos de selección

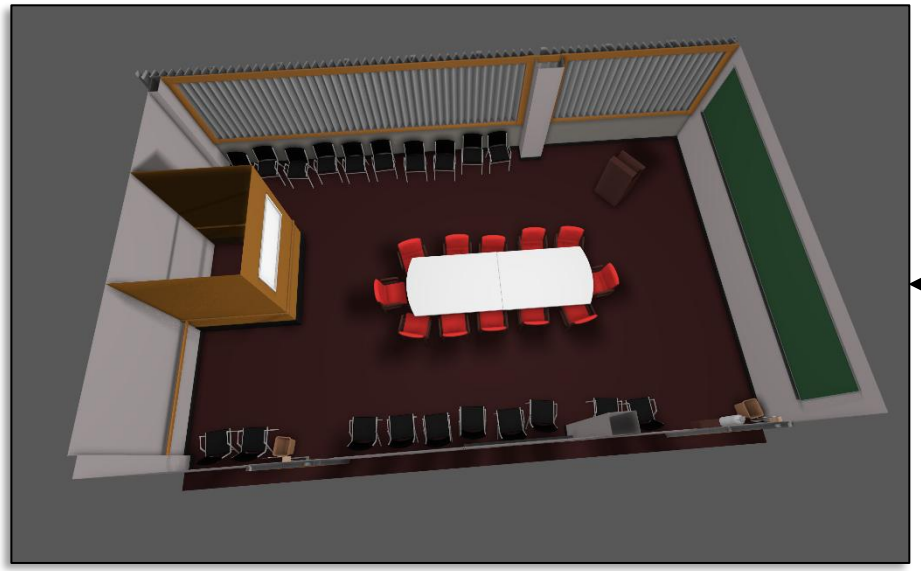
- **Color buffer**
  - **Eficiencia:** ●●●○○
  - **Complejidad:** ●●○○○
  - **Limitaciones:** Límite de objetos (256^3, R x G x B). Requiere redibujado



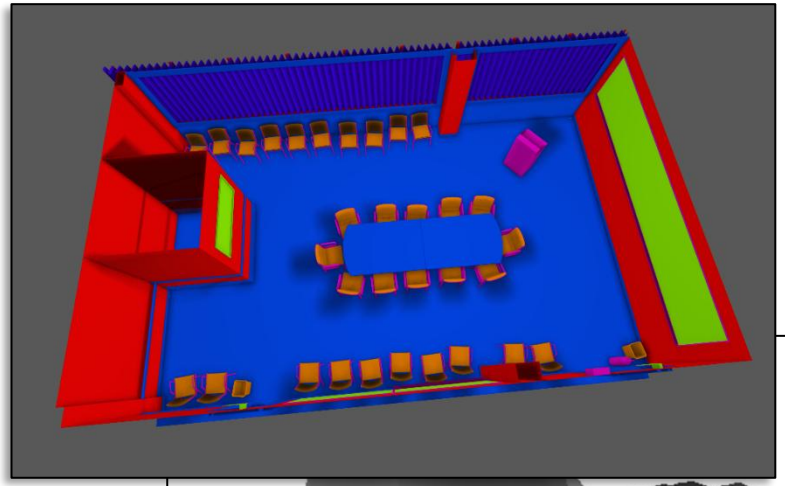
Métodos de selección

Color buffer

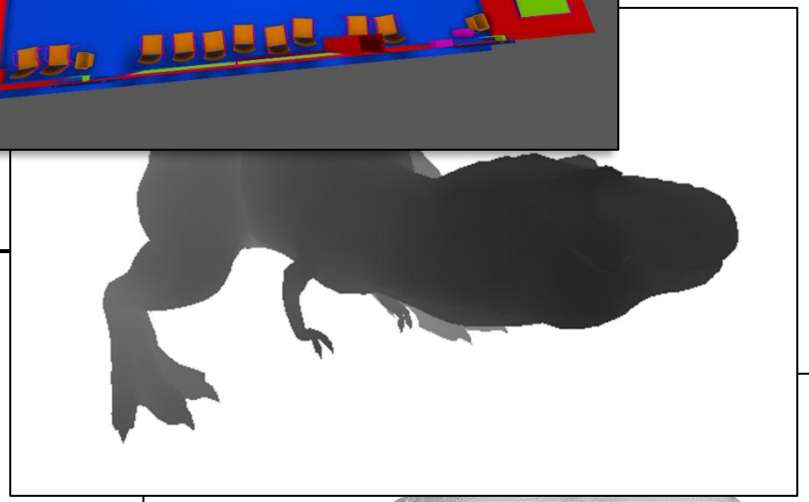
Framebuffer 0 (Default, GLUT)



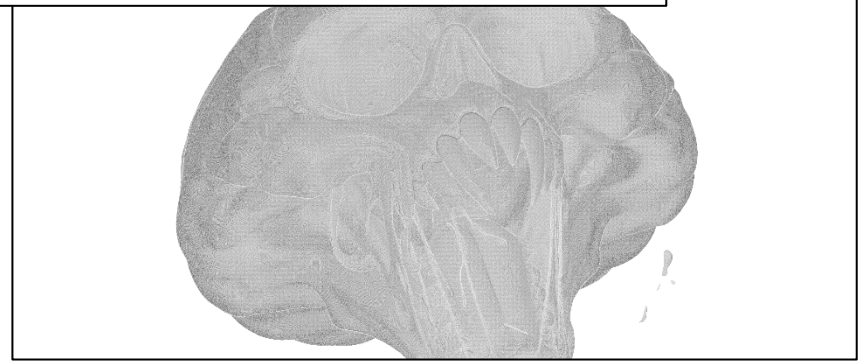
Framebuffer 1



Framebuffer 2



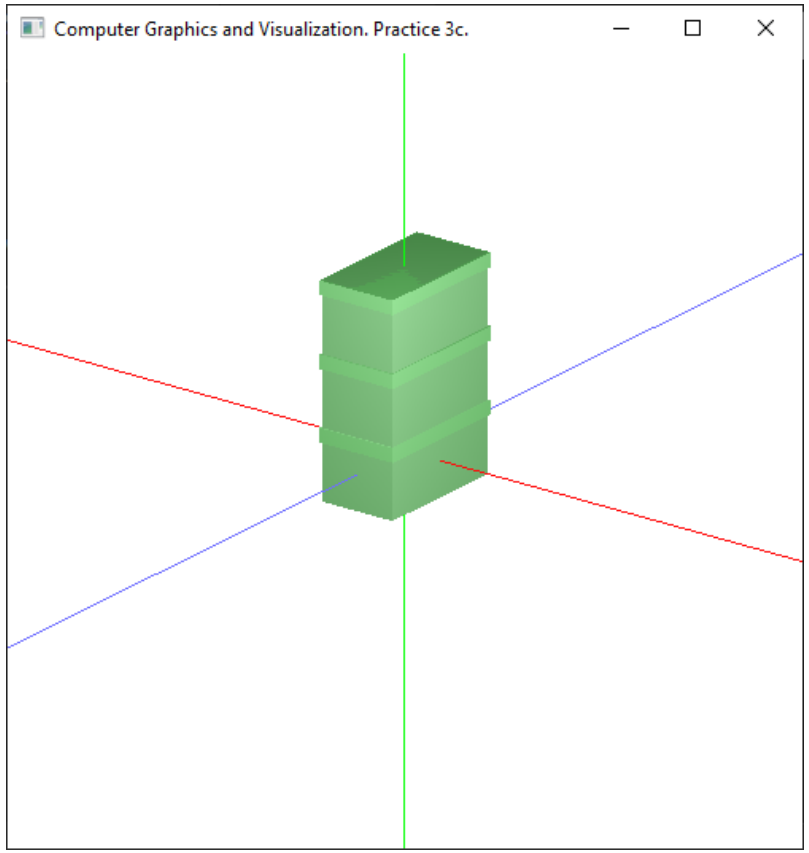
Framebuffer 3



Podemos tener varios *buffers* no visibles en paralelo. En nuestro caso, solo utilizaremos uno que almacena el color del objeto, que además, servirá de identificador del mismo.

Ejercicios

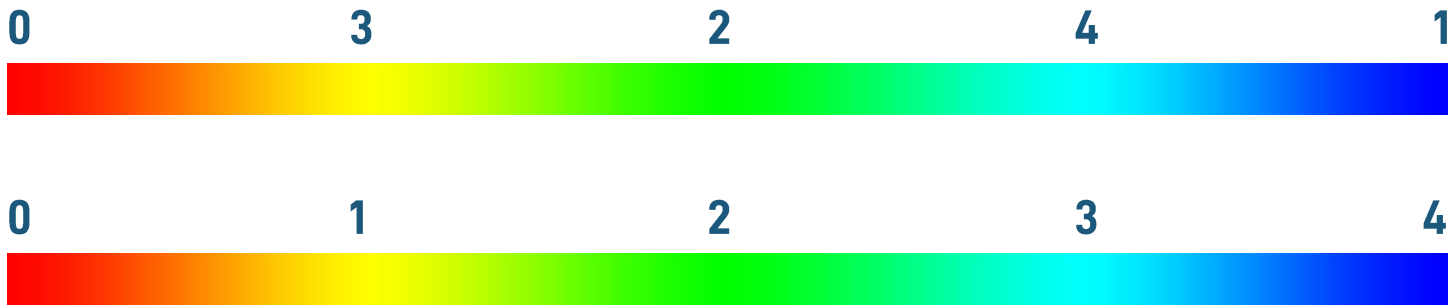
- **A. Selección por color**
  - **`igvInterfaz:modo`**, (¿usuario haciendo click y moviendo ratón?)
  - **`igvInterfaz:cursorX/cursorY`**
  - **`igvInterfaz:objeto_seleccionado`**
  - **`igvInterfaz:boton_retenido`**



Ejercicios

- **A. Selección por color**
  - **`igvInterfaz:modo`,** (¿usuario haciendo click y moviendo ratón?)
  - **`igvInterfaz:cursorX/cursorY`**
  - **`igvInterfaz:objeto_seleccionado`**
  - **`igvInterfaz:boton_retenido`**

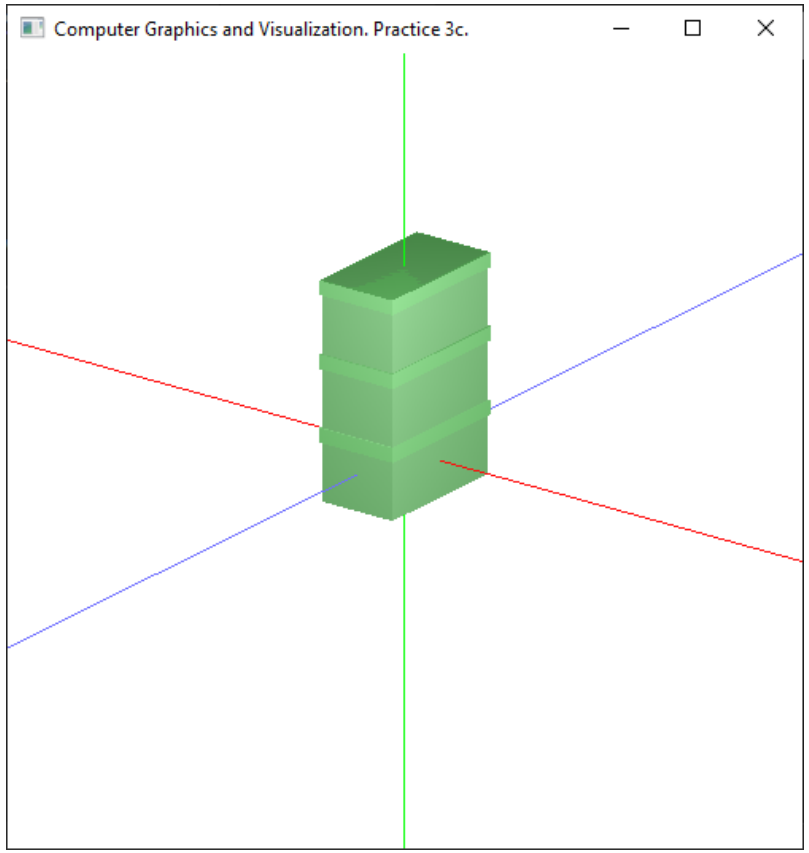
  - **Objetos con color diferente.**
    - Hay muchas maneras de asignar colores diferentes a cada objeto.



Ejercicios

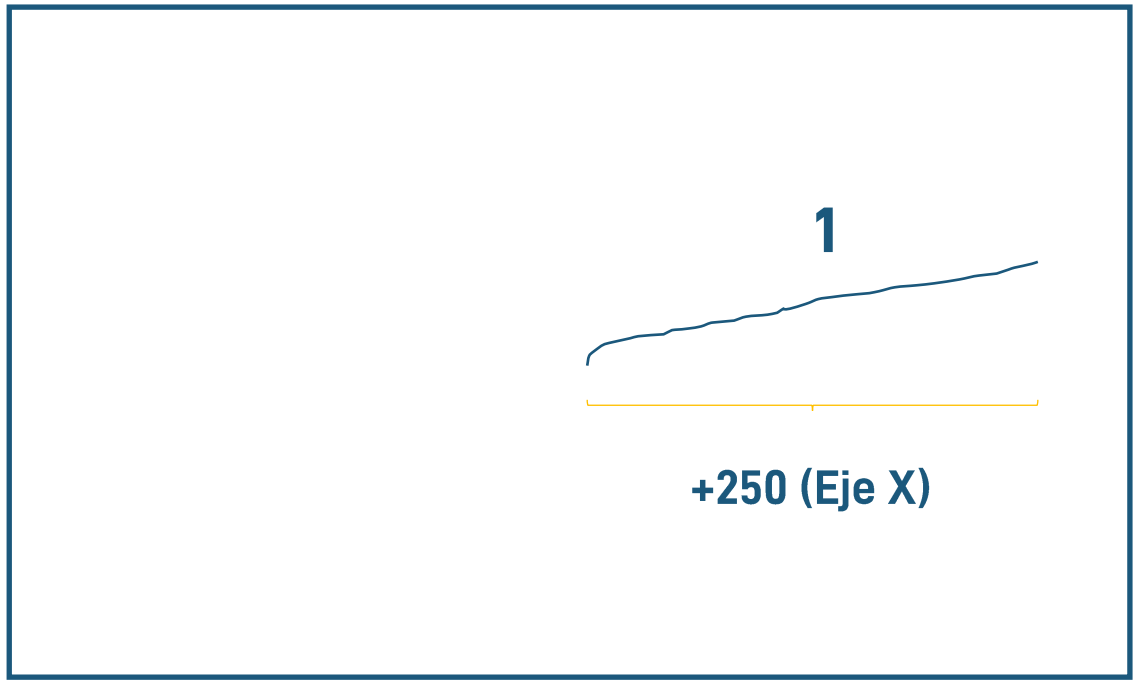
- **A. Selección por color**
  - **`igvInterfaz:modo`,** (¿usuario haciendo click y moviendo ratón?)
  - **`igvInterfaz:cursorX/cursorY`**
  - **`igvInterfaz:objeto_seleccionado`**
  - **`igvInterfaz:boton_retenido`**

  - **Objetos con color diferente.**
  - **Dos dibujados si se hace click.**
  - **Comportamiento de `igvCaja`.**



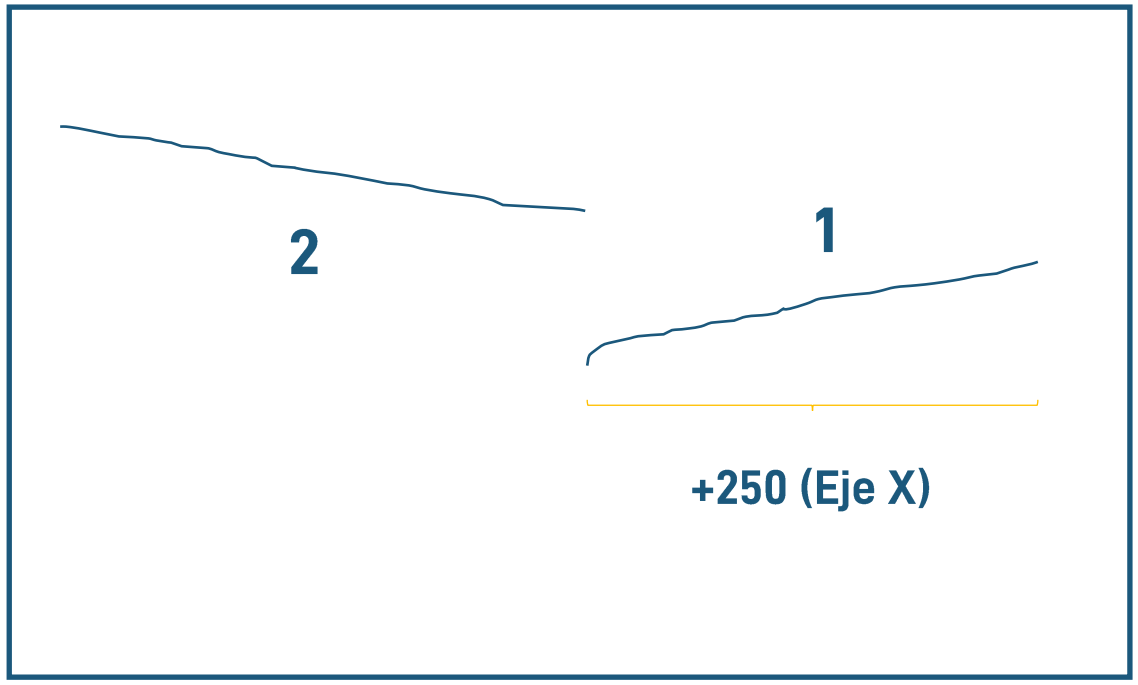
Ejercicios

- **B. Rotación**
  - **Controlar dirección de movimiento de ratón con `glutMotionFunc`.**
    - Posición anterior (si la hay) y actual.
  - **Rotación en torno al eje Y del objeto seleccionado (si existe).**



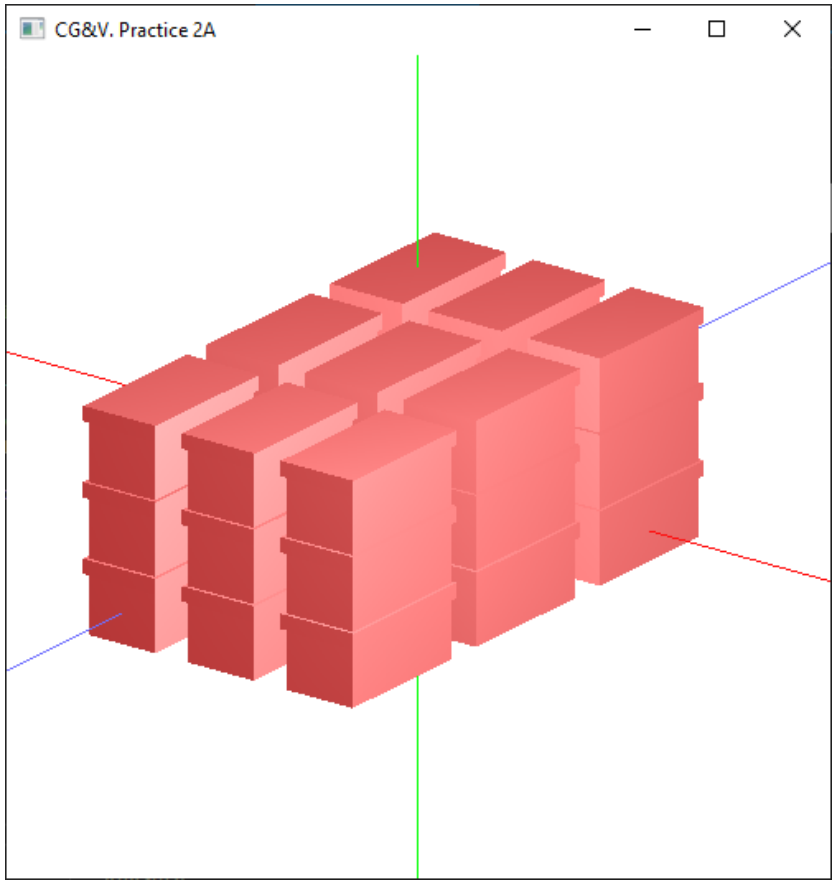
Ejercicios

- **B. Rotación**
  - **Controlar dirección de movimiento de ratón con `glutMotionFunc`.**
    - Posición anterior (si la hay) y actual.
  - **Rotación en torno al eje Y del objeto seleccionado (si existe).**



Ejercicios

- C. Modelado de matriz
  - Generar cajas en forma de matriz, como en la práctica 2.



Práctica 4. Iluminación y texturas

Curso académico 2023-2024

Informática Gráfica y Visualización

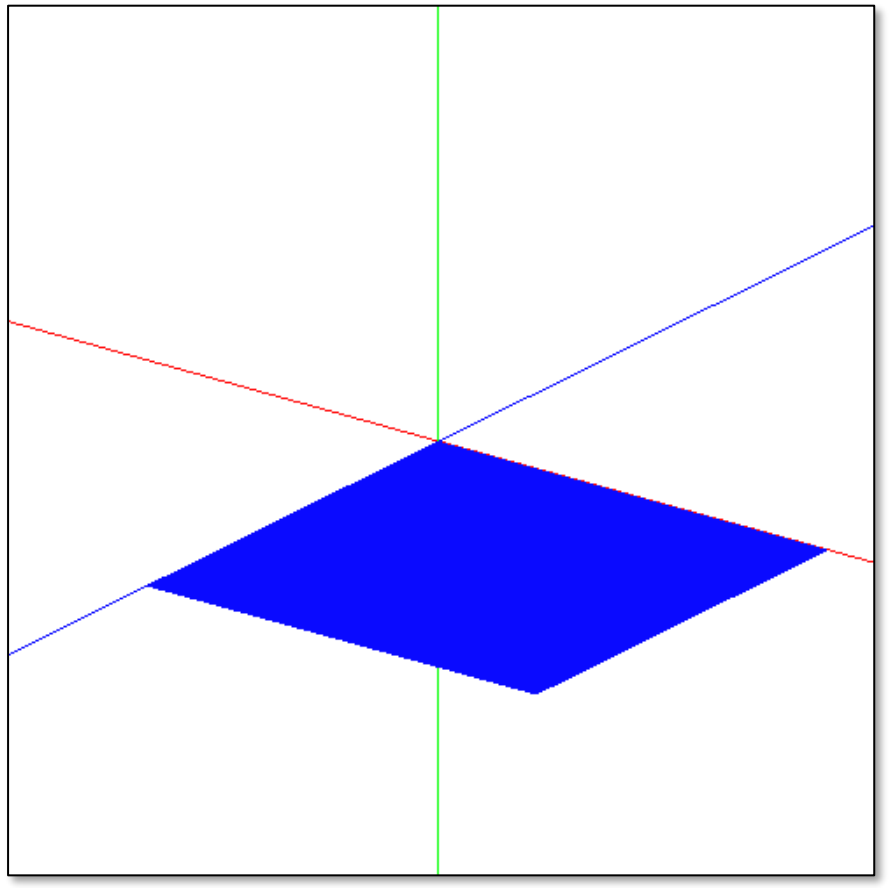
Grado en Ingeniería Informática

Estructura del proyecto

- **pr3c:** main.
- **igvInterfaz**: gestión de eventos y ventana.
- **igvEscena**: gestiona modelos en la escena y su representación.
- **igvPunto3D**: declaración de objetos de tipo punto y vector.
- **igvCamara**: configuración de cámara.
- **igvColor**: gestión de colores en la escena.
- **igvFuenteLuz**: iluminación de la escena.
- **igvMaterial**: material para modelar la representación de superficies en la escena.
- **igvTextura**: carga de imágenes y transferencia de texturas a la tarjeta gráfica.

Iluminación

- **Iluminación básica:**
  - $L_{rgb} * M_{rgb} * (L \cdot N)$
    - *Material emission*: M_{rgb}.
    - *Light emission*: L_{rgb}.
    - Vector normal de un punto de la superficie: N.
    - Posición de la luz y un punto sobre la superficie: $L = P_L - P_M$.



Iluminación

- **Iluminación avanzada:**
  - $L(x, w_o) = \int^{\lambda} \int^{\Omega} L_i(x, w_i) \cdot f(x, w_i, w_o) \cdot \cos(N, L_i)\, dw_i$

Iluminación

Iluminación avanzada:

- $L(x, w_o) = L(x, w_o) + \int^{\lambda} \int^{\Omega} L_i(x, w_i) \cdot f(x, w_i, w_o) \cdot \cos(N, L_i)\, dw_i$
  - Es imposible resolverlo en tiempo real.
  - En *ray-tracing*, se aplican técnicas de Monte-Carlo para reducir el espacio a muestrear de la integral.
- Tradicionalmente, se han utilizado otras funciones alternativas que simulan un *rendering* realista de manera más eficiente:
  - Cook-Torrance.
  - Phong.
  - etc.

Iluminación ambiente — Iluminación difusa — Iluminación especular

$$L(x, w_o) = \boxed{L_A M_A} + \boxed{L_D \cdot M_D \cdot (N \cdot L)} + \boxed{L_S \cdot M_S \cdot (N \cdot V)^{ns}}$$

Iluminación

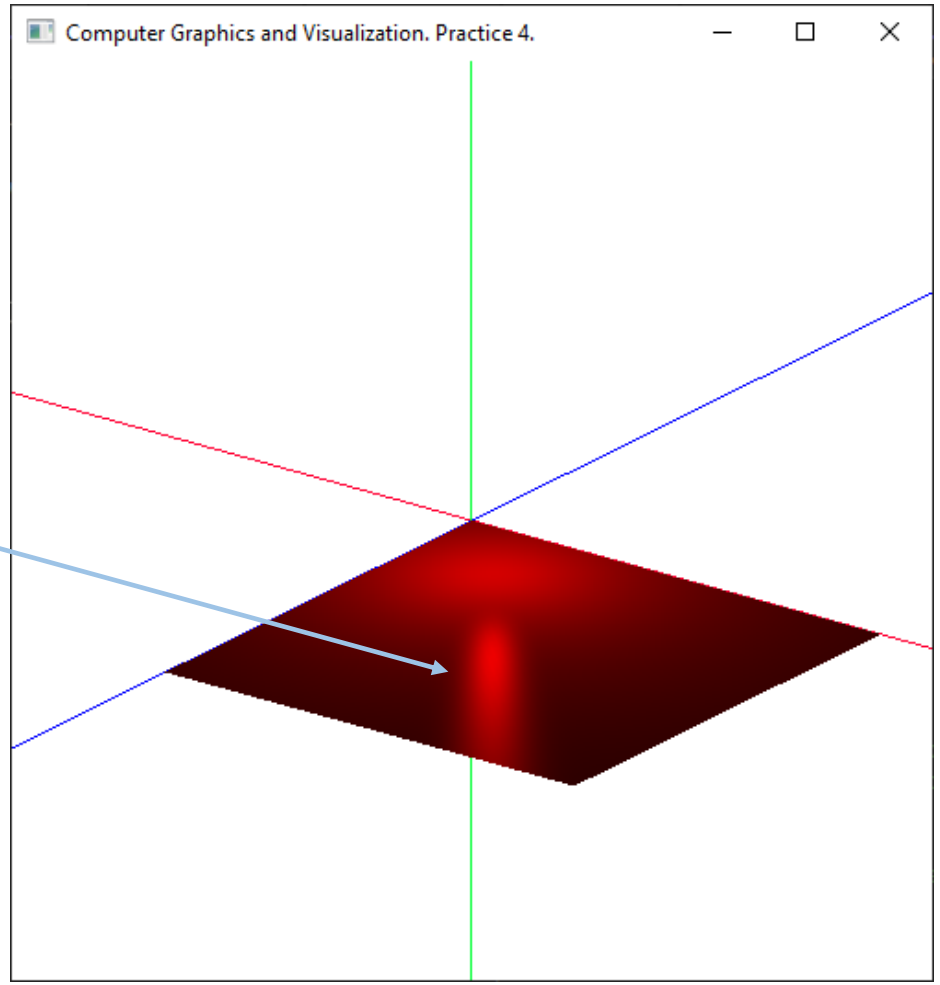
- **Modelos de iluminación:**
  - ¿Cómo incide la luz sobre una superficie?
    - **Luz puntual**: fuente de luz muy localizada.
      - Cercana a la escena.
      - Se podría representar como una esfera.

Iluminación

- **Modelos de iluminación:**
  - ¿Cómo incide la luz sobre una superficie?
    - Luz puntual: fuente de luz muy localizada.
    - **Luz direccional**: siempre sigue la misma dirección.
      - Fuente de luz en el infinito.
      - Se considera siempre la misma dirección, omitiendo los posibles cambios de dirección que pudieran existir a lo largo de una superficie muy extensa.
        - Se encuentra tan lejos la fuente de luz, que estos cambios son despreciables.
      - Luz solar.

Iluminación

- **Modelos de iluminación:**
  - Parámetros de una luz básica:
    - GL_POSITION
    - GL_AMBIENT
    - GL_DIFFUSE
    - GL_SPECULAR



glLightf(GL_LIGHT_x, attribute, value)

glMaterialfv(GL_FRONT, attribute, value)

Iluminación

Modelos de iluminación:

- Definición de una superficie con mayor nivel de detalle.

Iluminación

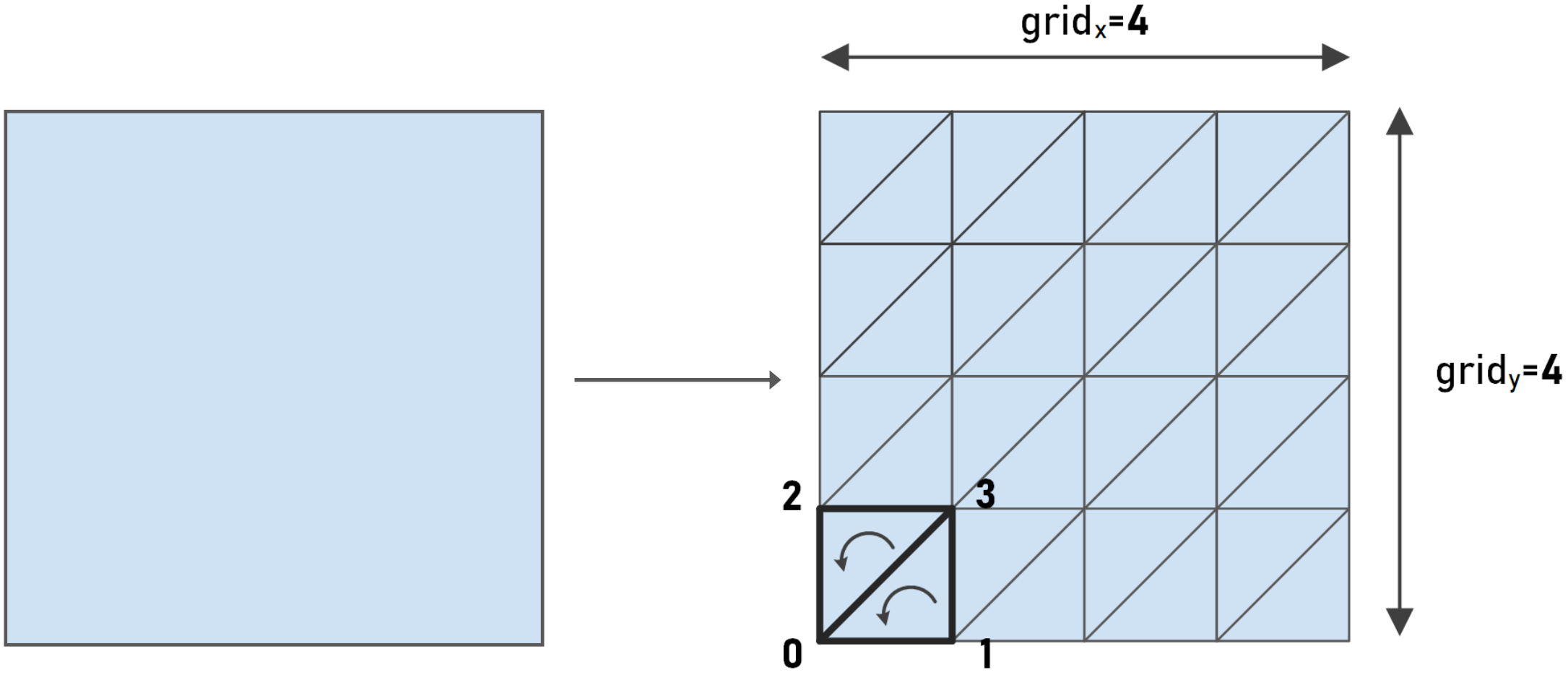
Modelos de iluminación:

- Definición de una superficie con mayor nivel de detalle.

Iluminación

Modelos de iluminación:

- Definición de una superficie con mayor nivel de detalle.



Iluminación

Modelos de iluminación:

- Definición de una superficie con mayor nivel de detalle.

```
glEnableClientState(GL_VERTEX_ARRAY);
        glVertexPointer(3, GL_FLOAT, 0, vertices);
        glEnableClientState(GL_NORMAL_ARRAY);
        glNormalPointer(GL_FLOAT, 0, normal_vect);
                glEnableClientState(GL_TEXTURE_COORD_ARRAY);
                glTexCoordPointer(...);
                        glDrawElements(GL_TRIANGLES, 9, GL_UNSIGNED_BYTE, indices);
glDisableClientState(GL_VERTEX_ARRAY);
glDisableClientState(GL_NORMAL_ARRAY);
glDisableClientState(GL_TEXTURE_COORD_ARRAY);
```

Iluminación

Modelos de iluminación:

- Definición de una superficie con mayor nivel de detalle.

```
glBegin(GL_...);
    glTexCoord2d(0, 1);
    glNormal3f(n_x1, n_y1, n_z1);
    glVertex3f(x_p1, y_p1, z_p1);
    glNormal3f(n_x2, n_y2, n_z2);
    glVertex3f(x_p2, y_p2, z_p2);
    glNormal3f(n_x3, n_y3, n_z3);
    glVertex3f(x_p3, y_p3, z_p3);
    glNormal3f(n_x4, n_y4, n_z4);
    glVertex3f(x_p4, y_p4, z_p4);
glEnd();
```

Iluminación

- **Modelos de iluminación:**
  - ¿Cómo incide la luz sobre una superficie?
    - **Luz spotlight**: luz puntual limitada a un cono.
      - Foco.

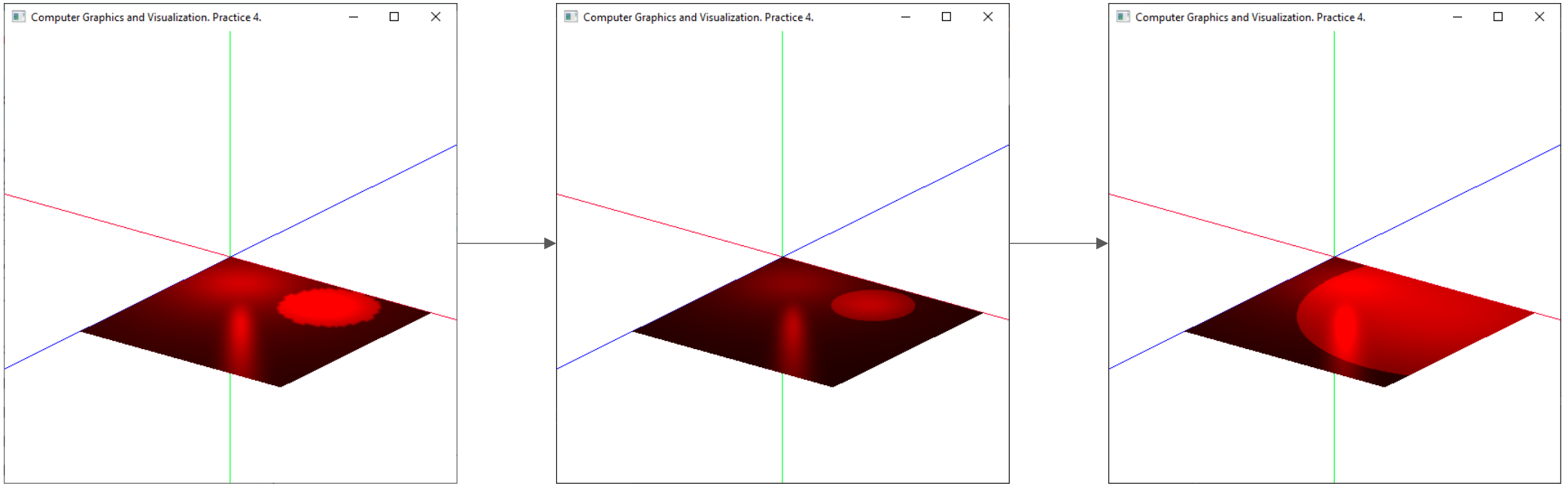
Iluminación

- **Modelos de iluminación:**
  - ¿Cómo incide la luz sobre una superficie?
    - **Luz spotlight**: luz puntual limitada a un cono.
      - Foco.
      - *Cutoff*: ángulo de corte.
      - *Falloff*: disminución de energía en la frontera.

Iluminación

Modelos de iluminación:

- Parámetros de una luz *spotlight*:
  - GL_SPOT_DIRECTION
  - GL_SPOT_CUTOFF
  - GL_SPOT_EXPONENT



Iluminación

Modelos de iluminación:

- OpenGL es una máquina de estados.
- Una vez se activa la luz, esta se tiene en cuenta en los siguientes *frames*.
  - **glEnable(id)**
  - **glDisable(id)**

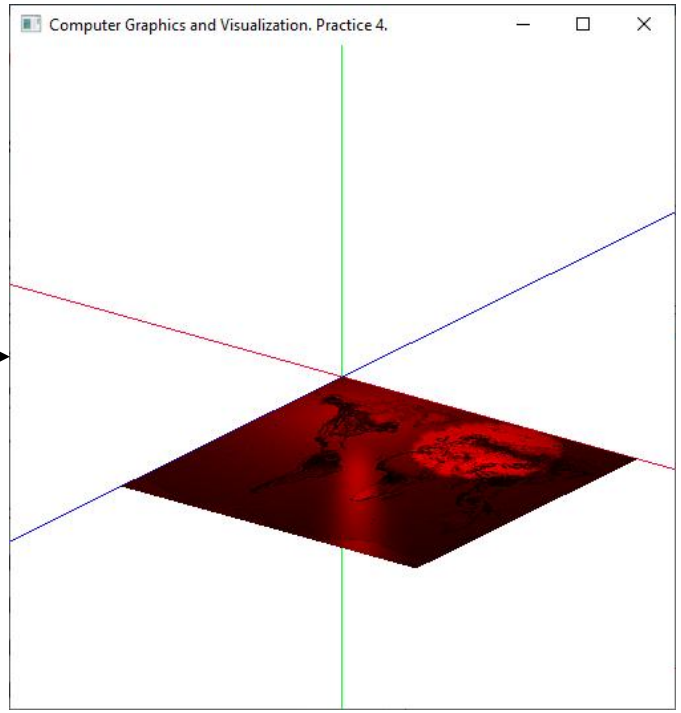
Iluminación

- **Modelos de iluminación:**
  - Otros parámetros de una luz:
    - GL_CONSTANT_ATTENUATION
    - GL_LINEAR_EQUATION
    - GL_QUADRATIC_ATTENUATION

$$\frac{1}{c_1 + c_2 \cdot d^2 + c_3 \cdot d^3}$$

Texturas

- **Definición de texturas:**
  - Generación: **glGenTextures**
  - *Bind*: **glBindTexture**
  - Parámetros: **glTexParameteri**
  - Envío de textura a GPU: **glTexImage2D**



Texturas

- **Definición de texturas:**
  - Generación: **glGenTextures**
  - *Bind*: **glBindTexture**
  - Parámetros: **glTexParameteri**
  - Envío de textura a GPU: **glTexImage2D**

Textures objects and parameters (open.gl)

Texturas

- **Definición de texturas:**
  - Generación: **glGenTextures**
  - *Bind*: **glBindTexture**
  - Parámetros: **glTexParameteri**
  - Envío de textura a GPU: **glTexImage2D**

Textures objects and parameters (open.gl)