

Informática Gráfica

Presentación

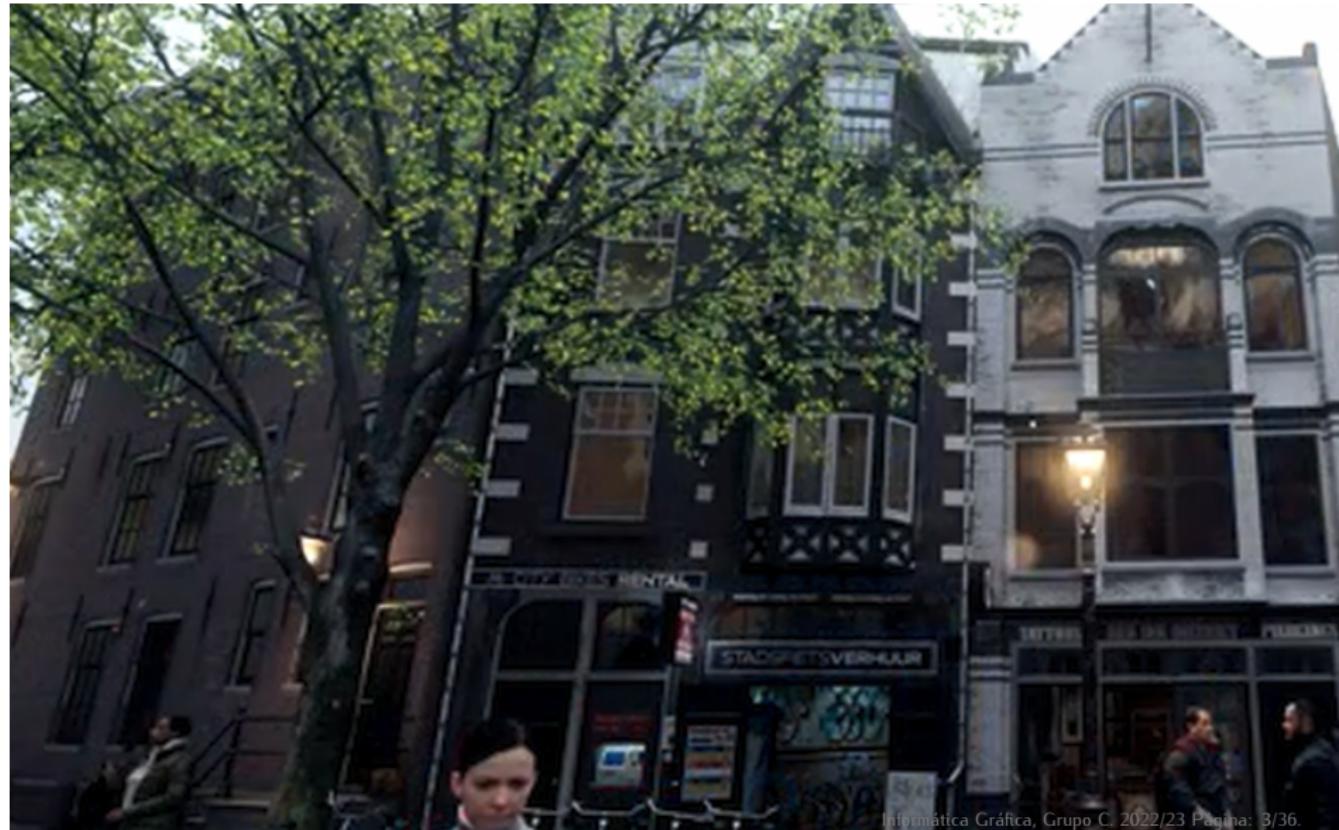
Juan Carlos Torres
Grupos C y D

Dpt. Lenguajes y Sistemas Informáticos
ETSI Informática y de Telecomunicación
Universidad de Granada

Curso 2024-25

Presentación

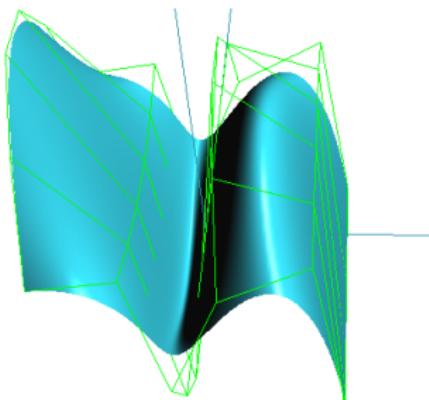
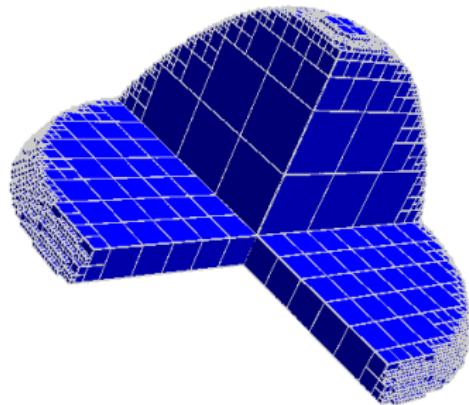
Informática Gráfica



Informática Gráfica

La Informática Gráfica es la parte de la Informática que se ocupa del procesamiento de información geométrica y visual.

- Representación de información (Modelado).
- Visualización (Rendering).
- Interacción
- Simulación



Informática Gráfica

La Informática Gráfica es la parte de la Informática que se ocupa del procesamiento de información geométrica y visual.

- Representación de información (Modelado).
- **Visualización (Rendering).**
- Interacción
- Simulación



LEON 7
Render .V3

Informática Gráfica

La Informática Gráfica es la parte de la Informática que se ocupa del procesamiento de información geométrica y visual.

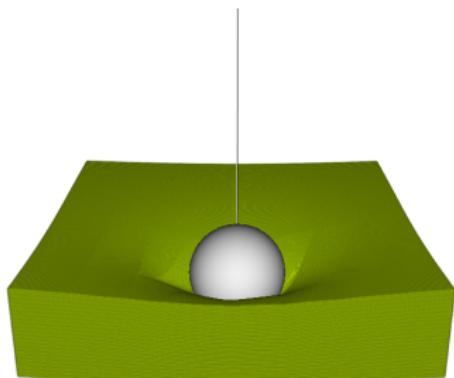
- Representación de información (Modelado).
- Visualización (Rendering).
- **Interacción**
- Simulación



Informática Gráfica

La Informática Gráfica es la parte de la Informática que se ocupa del procesamiento de información geométrica y visual.

- Representación de información (Modelado).
- Visualización (Rendering).
- Interacción
- Simulación



Aplicaciones

- Videojuegos.
- Animación y cine.
- Diseño.
- Visualización científica.
- Realidad virtual, mixta y aumentada.



Objetivos de la asignatura

- Conocer los fundamentos de la Informática Gráfica.
- Saber diseñar y utilizar las estructuras de datos más adecuadas para representar un modelo geométrico.
- Conocer los fundamentos de la visualización 3D
- Saber diseñar e implementar programas gráficos interactivos.

Ser capaz de desarrollar programas simples que visualicen, procesen y editen interactivamente un modelo geométrico, usando OpenGL.

Profesorado

(teoría C y D, prácticas C1)

profesor Juan Carlos Torres Cantero
despacho ETSIIT, planta 3^a, desp. 35
e-mail jctorres@ugr.es

(prácticas C2)

profesor Germán Arroyo Moreno
despacho ETSIIT, planta 3^a, desp. 31
e-mail arroyo@ugr.es

(prácticas D1 y D2)

profesor Alberto Jesús Durán López
despacho Edificio Auxiliar Despacho 1.4
e-mail albduranlopez@ugr.es

- 1 Introducción.** Informática Gráfica. Programación de aplicaciones gráficas interactivas.
- 2 Modelado de objetos.** Modelos geométricos. Mallas poligonales. Transformaciones geométricas. Instanciación. Grafos de escena. Representación y edición de mallas poligonales.
- 3 Visualización.** Cámara. Iluminación local y sombreado. Implementación de iluminación y sombreado mediante una biblioteca de programación gráfica. Texturas.
- 4 Interacción.** Sistemas interactivos. Eventos. Posicionamiento. Selección. Simulación física. Animación. Dispositivos de interacción.
- 5 Modelado y visualización avanzados.** Programación del cauce gráfico. Representación de escenas complejas. Ray Tracing.

Prácticas

- 1 Programación con biblioteca de programación gráfica.
- 2 Representación de modelos poligonales.
- 3 Modelos jerárquicos.
- 4 Iluminación y texturas.
- 5 Interacción.
- 6 Proyecto.

Evaluación

- Examen Teórico-Práctico (50 %)
- Evaluación de Prácticas (50 %)

Se aprueba la asignatura con una calificación final igual o superior a 5 y mas de 4 en cada prueba.

Teoría: Examen de ejercicios.

Prácticas: Las prácticas 2 a 5 puntúan 1. El proyecto 6 puntos, se evalúa en el laboratorio entre la Navidad y el examen de teoría. La práctica 1 no se evalúa.

Bibliografía online

- David J. Eck: Introduction to Computer Graphics. Version 1.4, August 2023.
☞ <https://math.hws.edu/graphicsbook/>
- John Collomosse: Fundamentals of Computer Graphics - CM20219 Lecture Notes University of Bath, UK
☞ <http://personal.ee.surrey.ac.uk/Personal/J.Collomosse/pubs/cm20219.pdf>
- D. Hearn, M.P. Baker: Gráficos por Computadora con OpenGL. Pearson, 2006.
☞ <https://ingenieriadymedida.files.wordpress.com/2013/12/graficosporcomputadora.pdf>
- D. Shreiner et al: OpenGL(R) Programming Guide: The red book, Version 2.1., 2008.
- D. Shreiner et al.: OpenGL Programming Guide 8 Edition. The Official Guide to Learning OpenGL, Version 4.3
☞ <https://www.cs.utexas.edu/users/fussell/courses/cs354/handouts/Addison.Wesley OpenGL Programming Guide 8th Edition.pdf>
- J. de Vries: Learn OpenGL-Graphics Programming.
☞ https://learnopengl.com/book/book_pdf.pdf
- F. Dunn, I. Parberry: 3D Math Primer for Graphics and Game Development. Wordware Publishing, Inc. 2002.
☞ <https://tfetimes.com/wp-content/uploads/2015/04/F.Dunn-I.Parberry-3D-Math-Primer-for-Graphics-and-Game-Development-WW-Publishing-Inc.-2002.pdf>

Metodología

Prácticas

Cinco primeras prácticas guiadas con objetivos muy concretos.

Proyecto libre.

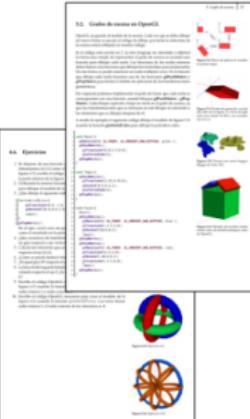
Teoría

Semana n

Resolución de dudas y realización de ejercicios de la lección n-1

Motivación y explicación de la lección n

Estudiar lección n



Semana n+1

Resolución de dudas y realización de ejercicios de la lección n

Motivación y explicación de la lección n+1

Temporización

Lecciones:

- 1 Introducción
- 2 Modelos geométricos
- 3 Transformaciones geométricas
- 4 Grafos de escena
- 5 Iluminación
- 6 Texturas
- 7 Sistemas interactivos: Selección
- 8 Lectura de posiciones
- 9 Transformación de visualización
- 10 Conectividad en mallas
- 11 Animación y simulación
- 12 Realidad Virtual y Aumentada
- 13 Visualización Realista
- 14 Modelado de Sólidos y Volúmenes

Introducción

OpenGL

- OpenGL es una interfaz software al hardware gráfico (API).
- Contiene más de 150 funciones para generar gráficos interactivos.
- Está diseñada para ser implementada en cauce de forma eficiente.
- Es independiente del hardware.
- No contiene funciones para manejar ventanas, procesar eventos de entrada ni describir objetos a alto nivel.
- OpenGL permite dibujar (renderizar) primitivas simples.
- Los programas que utilizan OpenGL deben gestionar la representación de su modelo geométrico (modo inmediato).

API Gráfica

- OpenGL hace que las aplicaciones sean independientes del hardware.
- Para gestionar ventanas y eventos de entrada se deben usar librerías auxiliares, que pueden o no ser dependientes de la máquina.
- Existen implementaciones de OpenGL para todas las plataformas existentes.
- Utiliza las capacidades de aceleración de las tarjetas gráficas.
- Realiza por software las funciones no disponibles en la tarjeta.

Historia

- 1980: Los programas gráficos se escribían para hardware específico.
- 1988: Silicon Graphics inc. era líder en estaciones gráficas. Sus sistemas usaban IRIS GL, que era propiedad de Silicon Graphics.
- 1990: Otras empresas empiezan a desarrollar hardware gráfico (SUN, IBM, HP), usando PHIGS (una API con modelo retenido).
- 1991: Silicon Graphics decide abrir su API para aumentar su influencia en el mercado, creando OpenGL.
- 1992: Silicon Graphics crea el OpenGL architectural review board (ARB), en la que también participan Microsoft, IBM, DEC e Intel.
- El ARB se encarga de establecer la especificación de OpenGL.
- Actualmente 9 empresas tienen voto en el ARB.
- No todas las optimizaciones de las tarjetas se pueden acceder a través de OpenGL.
- Los fabricantes suelen incluir extensiones a OpenGL para sus tarjetas.
- La última versión de OpenGL es la 4.5 de agosto de 2014.

Bibliotecas: GLU

OpenGL se distribuye junto con la librería GLU (OpenGL Utility Library). Esta biblioteca contiene:

- Funciones de dibujo de geometrías complejas (superficies, polígonos concavos).
- Funciones para control de proyecciones.

El nombre de las funciones de GLU comienza con `glu`.

Bibliotecas: glut

Glut es una librería auxiliar (OpenGL Utility Toolkit) que permite:

- Gestión de ventanas.
- Gestión de eventos de entrada.
- Dibujo de primitivas complejas (esferas, conos, toroides, teteras).
- Gestión de menús.

Es simple y existen implementaciones para todas las plataformas. La funciones comienzan con glut.

OpenGL

Un programa que usa OpenGL contiene código para dibujar cada frame.

Este código se puede ejecutar en la CPU, enviando el resultado de cada instrucción de dibujo a la GPU, o en la GPU. En este último caso se envía a la GPU una descripción de la escena, que solo se debe reenviar cada vez que se cambia la escena.

Estos dos modos de funcionamiento se conocen como

- Modo inmediato.
- Modo diferido.

En cualquier caso parte del procesamiento se realiza en la GPU. El código que lo hace puede estar prefijado o ser programable (dependiendo de la GPU y de la versión de OpenGL).

Instalación

Para trabajar con OpenGL hay que instalar las siguientes librerías con sus paquetes de desarrollo:

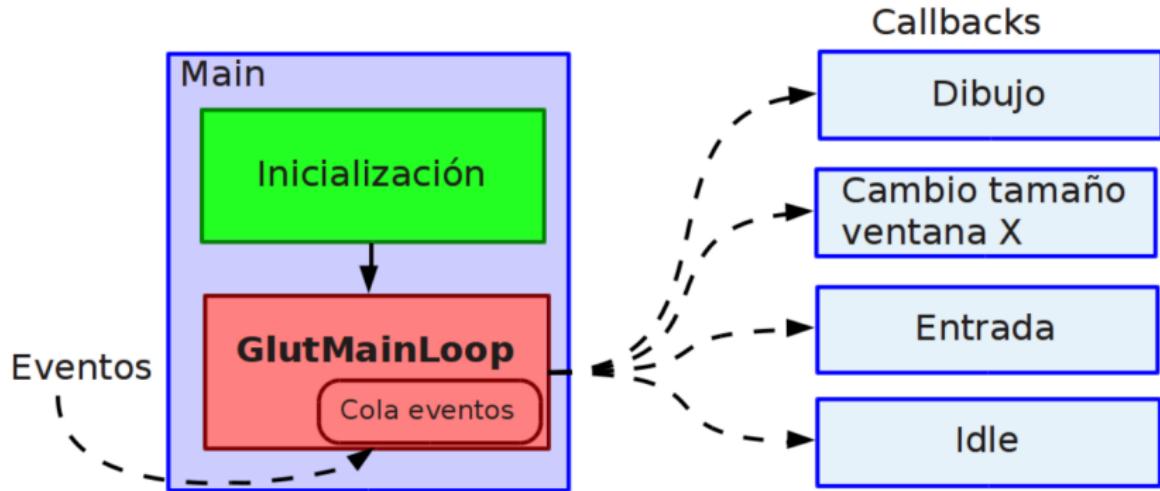
- libgl
- libglu
- libglut

Los nombres concretos de cada paquete dependerán de la distribución y la configuración de la máquina. Normalmente las librerías estarán instaladas y solo será necesario instalar los paquetes de desarrollo.

Para compilar usa

```
gcc fuente.c -lglut -lGLU -lGL -o ejecutable
```

Procesamiento de eventos

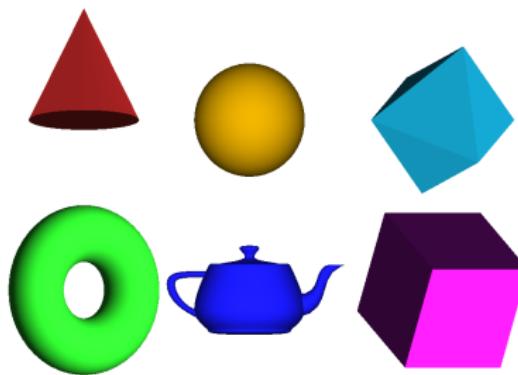


```
int main( int argc, char *argv[] )
{
    glutInit( &argc, argv );
    glutInitDisplayMode( GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH );
    glutCreateWindow( "IG: cubo" );
    glutDisplayFunc( Dibuja );
    glutReshapeFunc( Ventana );
    glutIdleFunc(idle);
    glEnable( GL_LIGHTING );
    glEnable( GL_LIGHT0 );
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
    return 0;
}
```

Primitivas de glut y GLU

Glut incluye funciones para dibujar objetos (cubos, esferas, toros, conos, teteras). Estos objetos se descomponen internamente en triángulos.

GLU incluye funciones para dibujar curvas y superficies (cuádricas y NURBS), y para descomponer (teselar) polígonos concavos.



```
glutSolidTorus(0.5,3,24,32); // dibuja un toro sólido con radios 0.5 y 3  
glutSolidCone(1,1,10,20); // dibuja un cono sólido con radio 1 y altura 10
```

Transformaciones geométricas

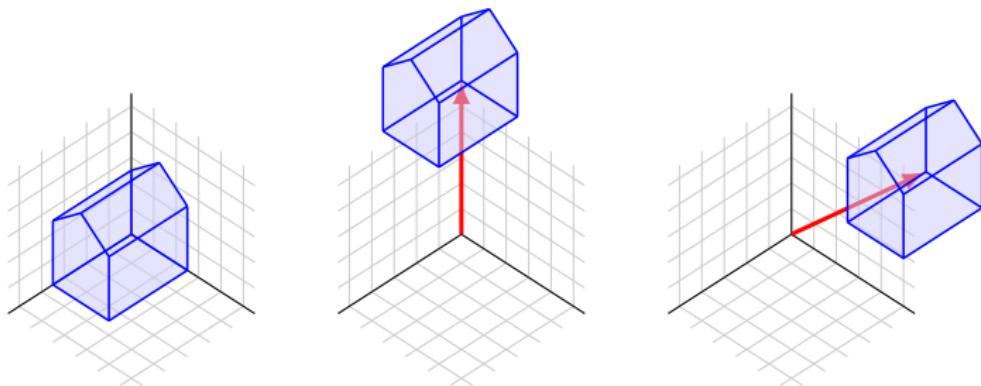
Las transformaciones geométricas son funciones que modifican las coordenadas de los puntos. Existen varias transformaciones geométricas simples que son muy útiles en Informática Gráfica para la definición de escenas y animaciones:

- **Traslación:** desplazar todos los puntos del espacio de igual forma, es decir, en la misma dirección y la misma distancia.
- **Escalado:** estrechar o alargar las figuras en una o varias direcciones.
- **Rotación:** rotar los puntos un ángulo en torno a un eje

Transformación de traslación en 3D

Si $\mathbf{d} = (d_x, d_y, d_z)$ es una tupla cualquier de \mathbb{E}_3 , la transformación de **traslación** $T_{\mathbf{d}}$ en \mathbb{E}_3 es una transformación geométrica que desplaza cualquier punto $\mathbf{p} = (x, y, z)$ sumándole \mathbf{d} , es decir:

$$T_{\mathbf{d}}(\mathbf{p}) \equiv \mathbf{p} + \mathbf{d} = (x, y, z) + (d_x, d_y, d_z) = (x + d_x, y + d_y, z + d_z)$$



$$\mathbf{d} = (0, 0, 0)$$

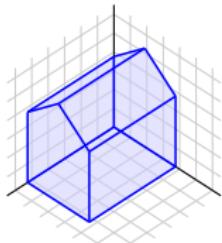
$$\mathbf{d} = (0, 1.2, 0)$$

$$\mathbf{d} = (0.7, 0.6, -0.5)$$

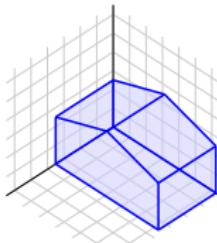
Transformación de escalado en 3D

Si $\mathbf{s} = (s_x, s_y, s_z)$, entonces una transformación de **escalado** $E_{\mathbf{s}}$ en \mathbb{E}_3 es una transformación geométrica que usa multiplicación componente a componente por \mathbf{s} , es decir para cualquier punto $\mathbf{p} = (x, y, z)$ se cumple:

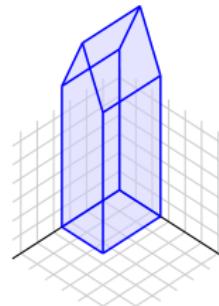
$$E_{\mathbf{s}}(\mathbf{p}) \equiv (s_x x, s_y y, s_z z)$$



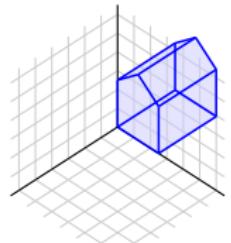
$\mathbf{s} = (1,5,1,5,1,5)$
uniforme
 $s_x = s_y = s_z$



$\mathbf{s} = (2,5,1,1)$
no uniforme
 $s_x \neq s_y = s_z$



$\mathbf{s} = (1,3,1)$
no uniforme
 $s_y \neq s_x = s_z$

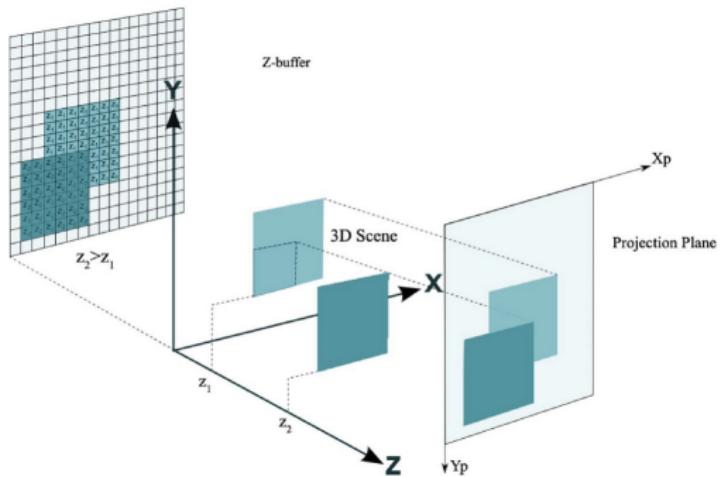


$\mathbf{s} = (1,1,-1)$
espejo
 $s_z < 0$

Z-Buffer

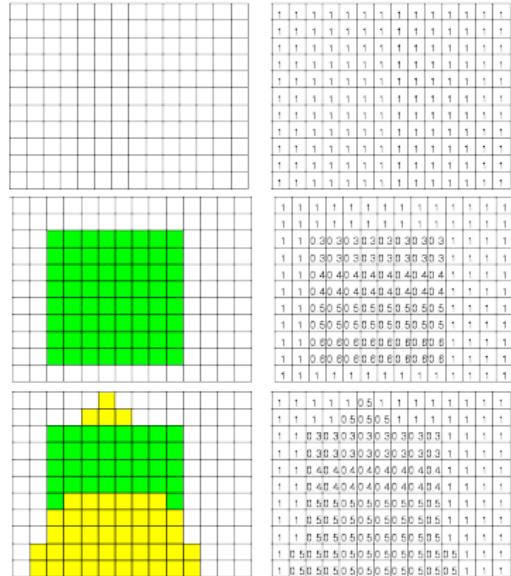
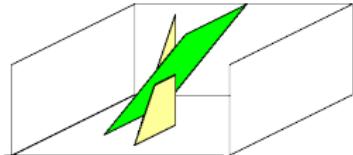
El algoritmo de Z-Buffer es el método usado para eliminar las partes ocultas

Para cada pixel que se va a dibujar se calcula su profundidad respecto a la cámara y solo se dibuja si no se ha dibujado previamente un objeto más próximo en ese pixel.



Z-Buffer

```
color canvas[resx,resy]
int zbuffer[resx,resy]=Zfar
for each primitive
    Rasterizar
        for each fragment
            if zbuffer[x,y] > fragment.depth
                zbuffer[x,y] = fragment.depth
                canvas[x,y] = fragment.color
```

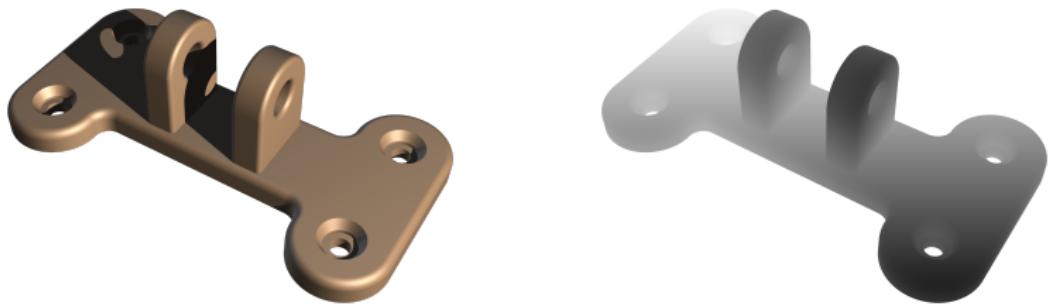


Z-Buffer

Las profundidades se transforman a un intervalo

$$z' = \frac{Z_{far} + Z_{near} - 2\frac{Z_{far}Z_{near}}{Z}}{Z_{far} - Z_{near}}$$

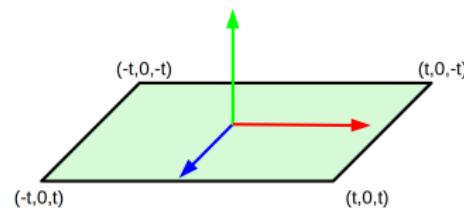
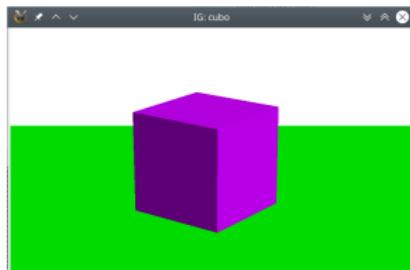
y se discretizan (para almacenarlas como enteros sin signo).



Ejemplo simple

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <GL/glut.h>
float roty=30.0;

void plano( float t ){//Construye un cuadrado horizontal
    glBegin( GL_QUADS );
        glNormal3f( 0.0, 1.0, 0.0 );
        glVertex3f( t, 0, t );
        glVertex3f( t, 0, -t );
        glVertex3f( -t, 0, -t );
        glVertex3f( -t, 0, t );
    glEnd();
}
```



Ejemplo simple

```
void Dibuja( ){
    float pos[4] = {5.0, 5.0, 10.0, 0.0 };
    float morado[4]={0.8,0,1,1}, verde[4]={0,1,0,1};
    glClearColor(1,1,1,1); // Fondo blanco
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    glLoadIdentity();
    glTranslatef(-0.5,-0.5,-100);
    glLightfv( GL_LIGHT0, GL_POSITION, pos );
    glRotatef( 20, 1.0, 0.0, 0.0 );
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, verde );
    plano(30);
    glRotatef( roty, 0.0, 1.0, 0.0 );
    glTranslatef(0,5,0);
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, morado );
    glutSolidCube(10);
    glutSwapBuffers();
}
```

Ejemplo simple

```
void Ventana(GLsizei ancho,GLsizei alto) {
    float D=ancho; if(D<alto) D=alto;
    glViewport(0,0,ancho,alto); //fija el area de dibujo
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-ancho/D,ancho/D,-alto/D,alto/D,5,250);
    glMatrixMode(GL_MODELVIEW);
}

void idle(){
    roty +=0.15;
    glutPostRedisplay();
}
```

Ejemplo simple

```
int main( int argc, char *argv[] ) {
    glutInit( &argc, argv );
    glutInitDisplayMode( GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH );
    glutCreateWindow( "IG: cubo" );
    glutDisplayFunc( Dibuja );
    glutReshapeFunc( Ventana );
    glutIdleFunc( idle );
    glEnable( GL_LIGHTING );
    glEnable( GL_LIGHT0 );
    glEnable( GL_DEPTH_TEST );
    glutMainLoop();
    return 0;
}
```