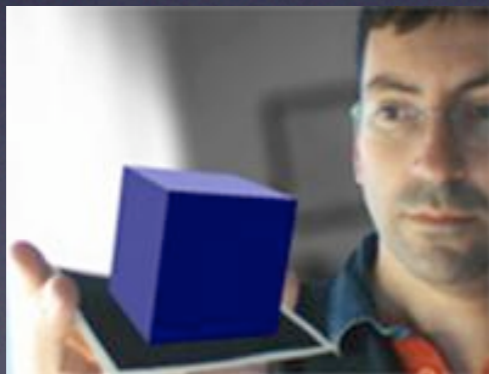


Gráficos por Computador

3D con processing



Jordi Linares i Pellicer

Escola Politècnica Superior d'Alcoi

Dep. de Sistemes Informàtics i Computació

jlinares@dsic.upv.es

<http://www.dsic.upv.es/~jlinares>

3D con processing

- *processing* dispone de dos modos de visualización 3D: P3D y OPENGL
- P3D está basado en software, OPENGL se implementa mediante OpenGL (hardware). Con algunas excepciones, los dos modos soportan el mismo conjunto de primitivas y funciones.
- Las primitivas y funciones utilizadas en 2D pueden utilizarse en 3D (puede haber alguna excepción)
- Algunas de las primitivas, como líneas (`line`), puntos (`point`), curvas y shapes (primitiva `vertex`), permiten definir 3 coordenadas: `x`, `y`, `z`
- El resto de las primitivas 2D también pueden utilizarse (con una `z=0` implícita)
- Las funciones de trazo, relleno, texto e imágenes (texturas) también pueden utilizarse en 3D

3D con processing

Transformaciones geométricas 3D

Traslación
$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Escalado
$$\begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotación eje x
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha & 0 \\ 0 & \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotación eje y
$$\begin{bmatrix} \cos \alpha & 0 & \sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotación eje z
$$\begin{bmatrix} \cos \alpha & -\sin \alpha & 0 & 0 \\ \sin \alpha & \cos \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3D con processing

Transformaciones geométricas 3D

Traslación

```
translate(tx, ty, tz)
```

Escalado

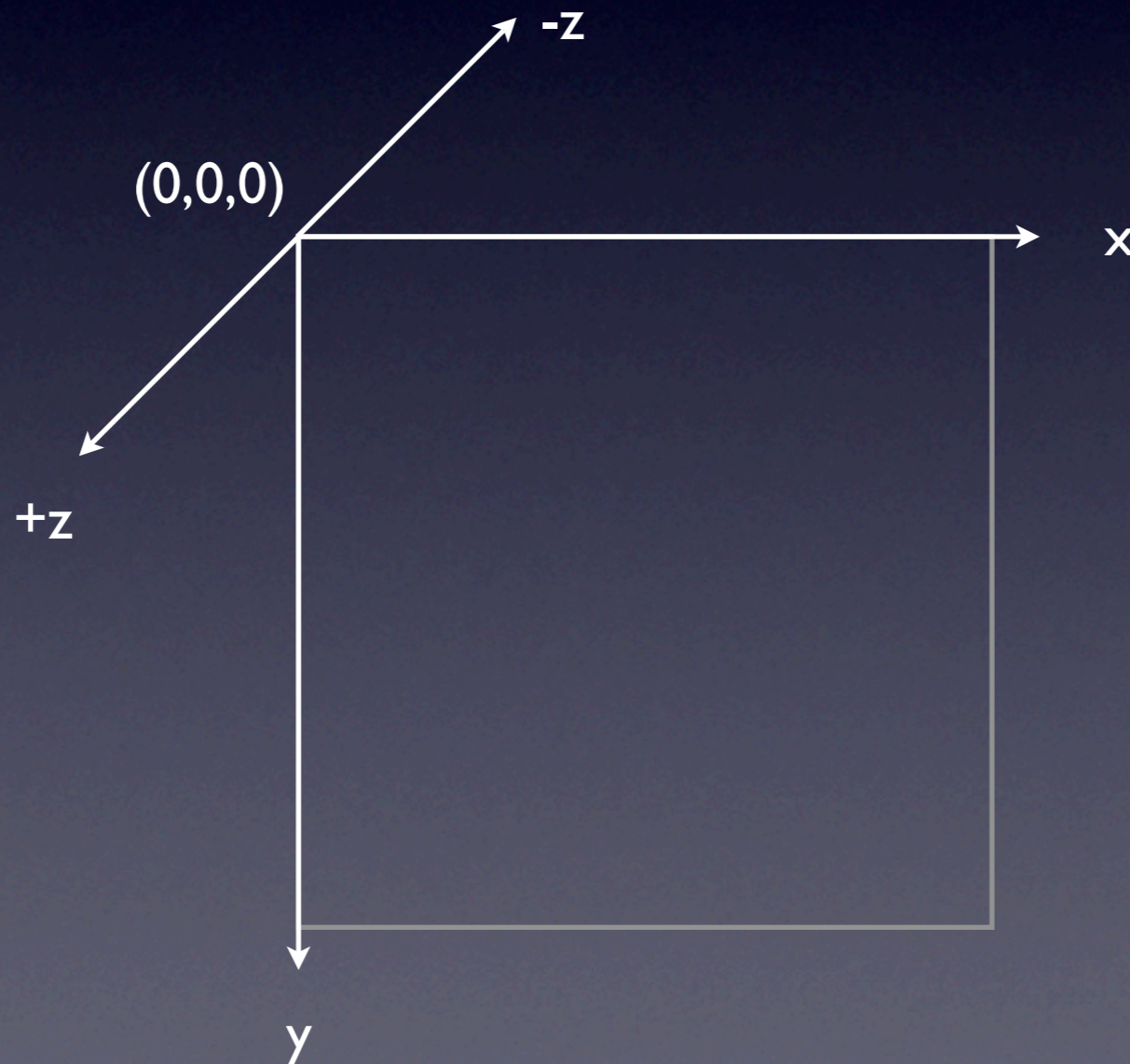
```
scale(sx, sy, sz)
```

Rotación respecto a un eje

```
rotateX(), rotateY(), rotateZ()
```

3D con processing

- Por defecto, se realiza una proyección perspectiva simple
- El $(0,0,0)$ se encuentra ubicado en la esquina superior izquierda
- $-z$ para alejar



3D con processing

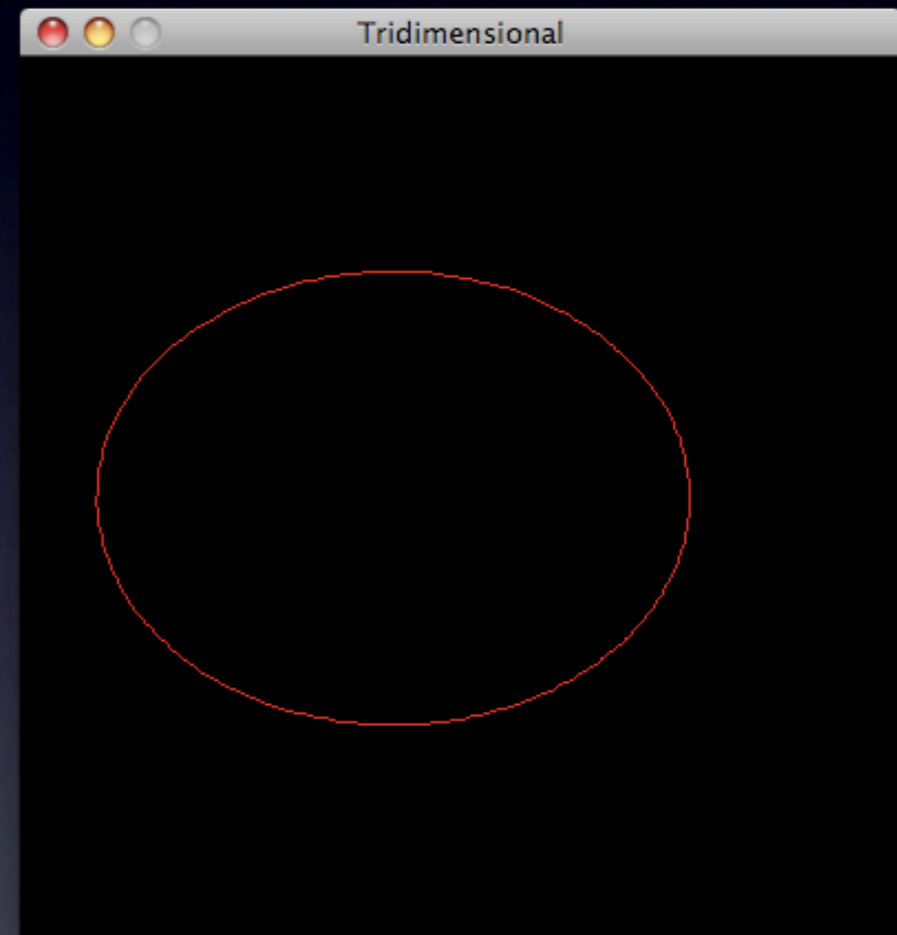
```
// Elipse rotando alrededor
// del eje y
float ang = 0.0;

void setup()
{
  size(400, 400, P3D);
  stroke(255, 0, 0);
  noFill();
}

void draw()
{
  background(0);

  // Dibujaremos centrado
  // en el (0,0,0)
  translate(width/2, height/2);

  rotateY(ang += 0.1);
  ellipse(0, 0, 300, 200);
}
```



3D con processing

```
box(ancho, alto, profundo)
```

- Dibuja un ortoedro (caja) centrada en el (0,0,0) de ancho (x), alto (y) y profundo (z) como argumentos

```
sphere(radio)
```

- Dibuja un esfera centrada en el (0,0,0) de radio suministrado como argumento
- El nivel de detalle en el que la esfera será generada se puede ajustar con la función `sphereDetail(n)`, donde n implica que los vértices serán generados cada $360^\circ/n$ (por defecto, $n = 30$)

3D con processing

```
// Cubo rotando alrededor
// de los tres ejes

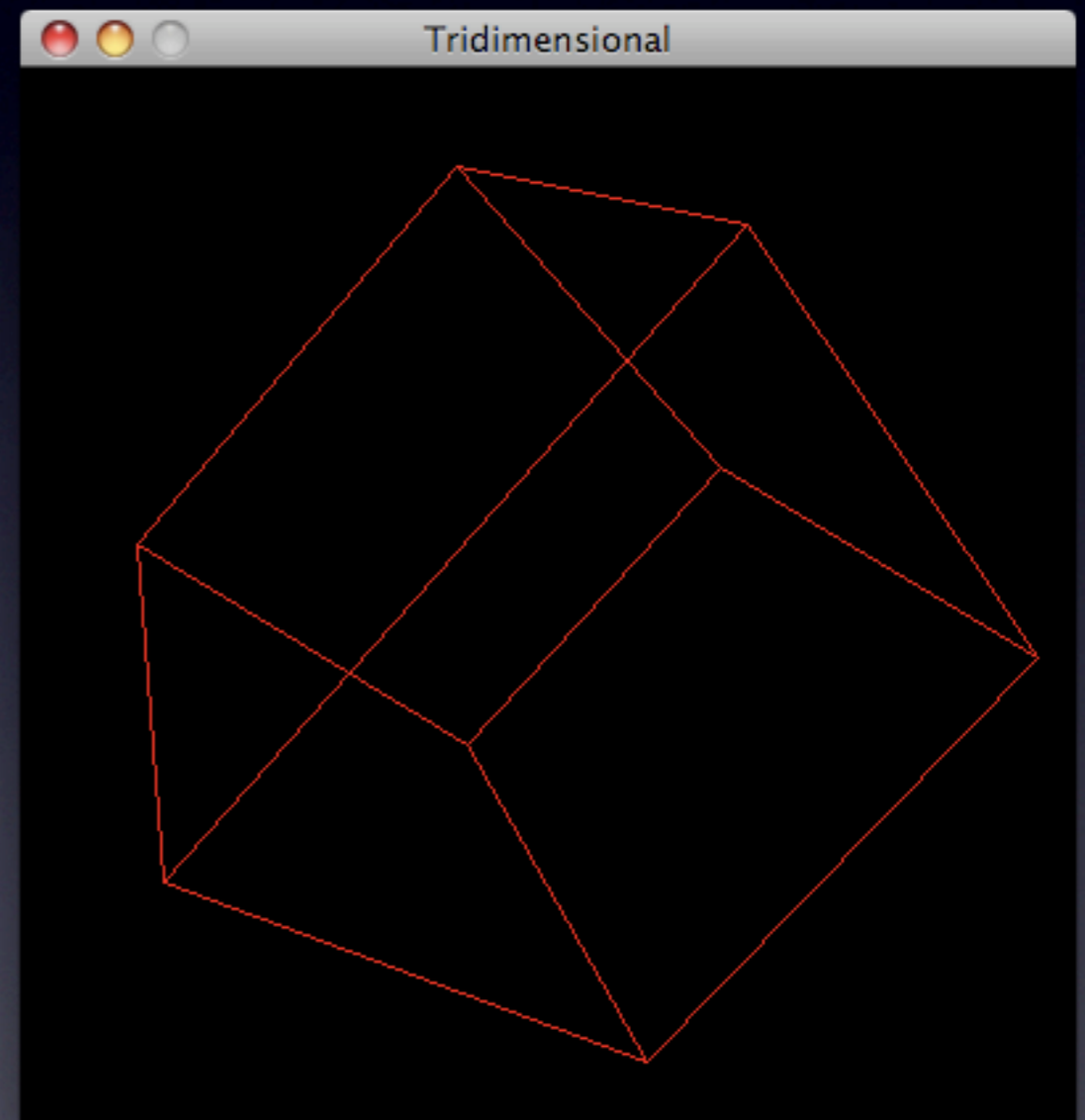
// Versión alámbrica

void setup()
{
  size(400, 400, P3D);
  stroke(255, 0, 0);
  noFill();
}

void draw()
{
  background(0);

  // Dibujaremos centrado
  // en el (0,0,0)
  translate(width/2, height/2);

  rotateX(frameCount*PI/60.0);
  rotateY(frameCount*PI/120.0);
  rotateZ(frameCount*PI/180.0);
  box(200, 200, 200);
}
```



3D con processing

```
// Cubo rotando alrededor
// de los tres ejes

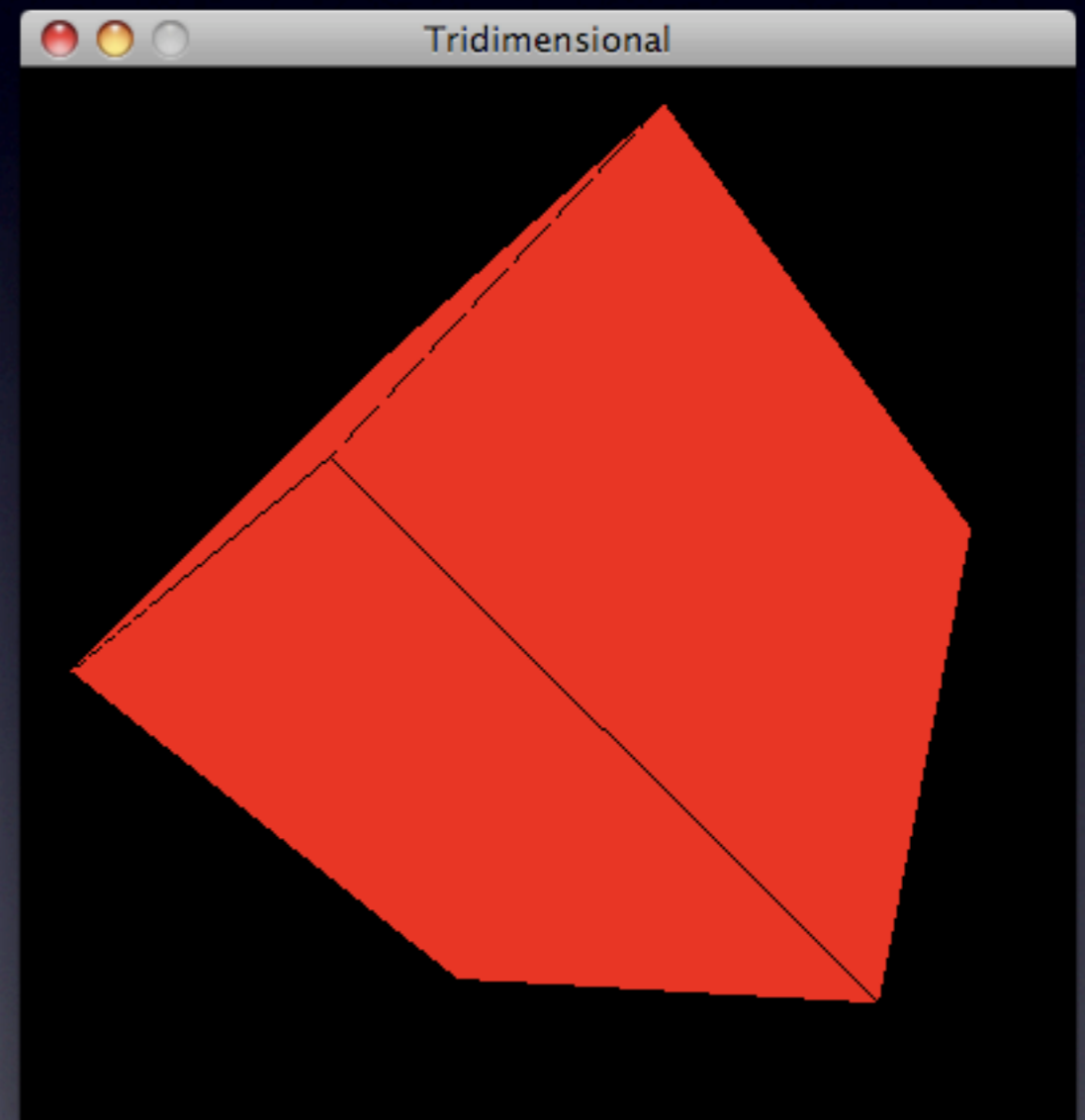
// Versión sólida

void setup()
{
  size(400, 400, P3D);
  fill(255, 0, 0);
}

void draw()
{
  background(0);

  // Dibujaremos centrado
  // en el (0,0,0)
  translate(width/2, height/2);

  rotateX(frameCount*PI/60.0);
  rotateY(frameCount*PI/120.0);
  rotateZ(frameCount*PI/180.0);
  box(200, 200, 200);
}
```



3D con processing

```
// Cubo rotando alrededor
// de los tres ejes

// Versión con iluminación básica

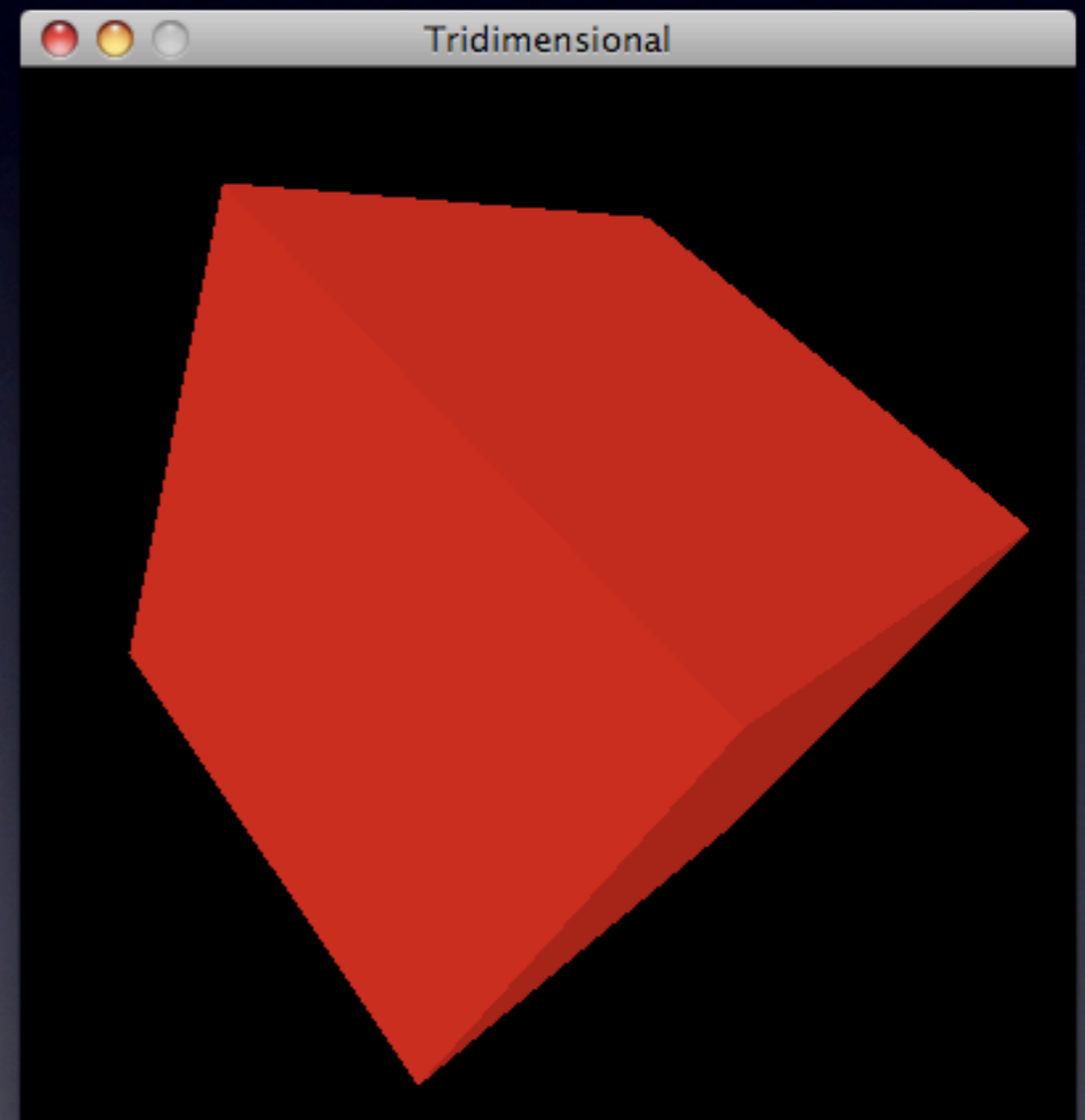
void setup()
{
  size(400, 400, P3D);
  fill(255, 0, 0);
  noStroke();
}

void draw()
{
  background(0);

  // Iluminación básica
  lights();

  // Dibujaremos centrado
  // en el (0,0,0)
  translate(width/2, height/2);

  rotateX(frameCount*PI/60.0);
  rotateY(frameCount*PI/120.0);
  rotateZ(frameCount*PI/180.0);
  box(200, 200, 200);
}
```



3D con processing

```
// Cubo interactivo
float rotX = 0.0, rotY = 0.0;
int lastX, lastY;
float distX = 0.0, distY = 0.0;

void setup(){
  size(400, 400, P3D);
  noStroke();
  fill(255, 0, 0);
}

void draw(){
  background(0);

  lights();

  translate(width/2, height/2);
  rotateX(rotX + distY);
  rotateY(rotY + distX);

  box(200, 200, 200);
}
```

```
void mousePressed()
{
  lastX = mouseX;
  lastY = mouseY;
}

void mouseDragged()
{
  distX = radians(mouseX - lastX);
  distY = radians(lastY - mouseY);
}

void mouseReleased()
{
  rotX += distY;
  rotY += distX;
  distX = distY = 0.0;
}
```

Práctica 7-1

- Modificar el programa anterior para permitir también modificar el factor de zoom, mediante el acercamiento o alejamiento del cubo
- Para ello, habrá que añadir a la traslación un desplazamiento en z
- Inicialmente este desplazamiento será 0, y deberá variar de 0 a -500 con incrementos de 10
- Para ello se puede capturar el evento de teclado `KeyPressed()` para detectar la pulsación de la tecla UP y DOWN

3D con processing

```
// Lo hacemos con OpenGL (hardware)
// Necesitamos hacer el import
import processing.opengl.*;

float rotX = 0.0, rotY = 0.0;
int lastX, lastY;
float distX = 0.0, distY = 0.0;

// Textura
PImage foto;

void setup(){
  size(400, 400, OPENGL);
  noStroke();
  foto = loadImage("foto.jpg");

  // Vamos a referirnos a las coordenadas de la
  // textura de (0,0) a (1,1)
  textureMode(NORMALIZED);
}

void draw(){
  background(0);

  translate(width/2, height/2);
  rotateX(rotX + distY);
  rotateY(rotY + distX);

  // Hacemos el cubo de 200 x 200 x 200
  // Lo dibujamos de -1 a 1
  scale(100, 100, 100);

  beginShape(QUADS);
  texture(foto);
```

```
// Proporcionamos los vértices
// 3D de cada cara del cubo.
// Los dos valores finales
// son las coordenadas de la
// textura que corresponde
// al vértice en cuestión

// +Z "front" face
vertex(-1, -1, 1, 0, 0);
vertex( 1, -1, 1, 1, 0);
vertex( 1,  1, 1, 1, 1);
vertex(-1,  1, 1, 0, 1);

// -Z "back" face
vertex( 1, -1, -1, 0, 0);
vertex(-1, -1, -1, 1, 0);
vertex(-1,  1, -1, 1, 1);
vertex( 1,  1, -1, 0, 1);

// +Y "bottom" face
vertex(-1,  1,  1, 0, 0);
vertex( 1,  1,  1, 1, 0);
vertex( 1,  1, -1, 1, 1);
vertex(-1,  1, -1, 0, 1);

// -Y "top" face
vertex(-1, -1, -1, 0, 0);
vertex( 1, -1, -1, 1, 0);
vertex( 1, -1,  1, 1, 1);
vertex(-1, -1,  1, 0, 1);

// +X "right" face
vertex( 1, -1,  1, 0, 0);
vertex( 1, -1, -1, 1, 0);
vertex( 1,  1, -1, 1, 1);
vertex( 1,  1,  1, 0, 1);
```

```
// -X "left" face
vertex(-1, -1, -1, 0, 0);
vertex(-1, -1,  1, 1, 0);
vertex(-1,  1,  1, 1, 1);
vertex(-1,  1, -1, 0, 1);

endShape();
}

void mousePressed()
{
  lastX = mouseX;
  lastY = mouseY;
}

void mouseDragged()
{
  distX = radians(mouseX - lastX);
  distY = radians(lastY - mouseY);
}

void mouseReleased()
{
  rotX += distY;
  rotY += distX;
  distX = distY = 0.0;
}
```

3D con processing



3D con processing

```
import processing.opengl.*;

// Dibujo de una función 3D
float rotX = 0.0, rotY = 0.0;
int lastX, lastY;
float distX = 0.0, distY = 0.0;

// Pasos de función
int steps = 50;

// Escala z
float scaleZ = 200.0;

// Zoom z
float zoomZ = -300.0;

// Tamaño gráfica
float gX = 500.0, gY = 500.0;

void setup()
{
  size(500, 500, OPENGL);
  noFill();
}

float funcion(float x, float y)
{
  return x*x*x + y*y*y;
}

void draw()
{
  background(0);

  // Centramos resultado sobre ventana
  translate(gX/2, gY/2, zoomZ);

  // Rotaciones según ratón
  rotateY(rotY + distX);
  rotateX(rotX + distY);

  // Centramos la gráfica sobre el (0, 0);
  translate(-gX/2, -gY/2);

  // Hacemos la función que
  // cubra 400 x 400 x scaleZ
  scale(gX, gY, scaleZ);

  // Dibujamos la gráfica
  stroke(255);
  dibujarFuncion();

  // Dibujamos ejes
  stroke(255, 0, 0);
  line(0,0,0,2000,0,0);
  stroke(0,255,0);
  line(0,0,0,0,2000,0);
  stroke(0,0,255);
  line(0,0,0,0,0,2000);
}

void dibujarFuncion()
{
  float x, y, z;
  int i = 0, j = 0;
  float in_steps = 1.0 / steps;

  float[][] matriz = new float[steps+1][steps+1];

  for (y = 0.0, j = 0; y <= 1.0; y+=in_steps, j++)
    for (x = 0.0, i = 0; x <= 1.0; x+=in_steps, i++)
      matriz[i][j] = funcion(x, y);

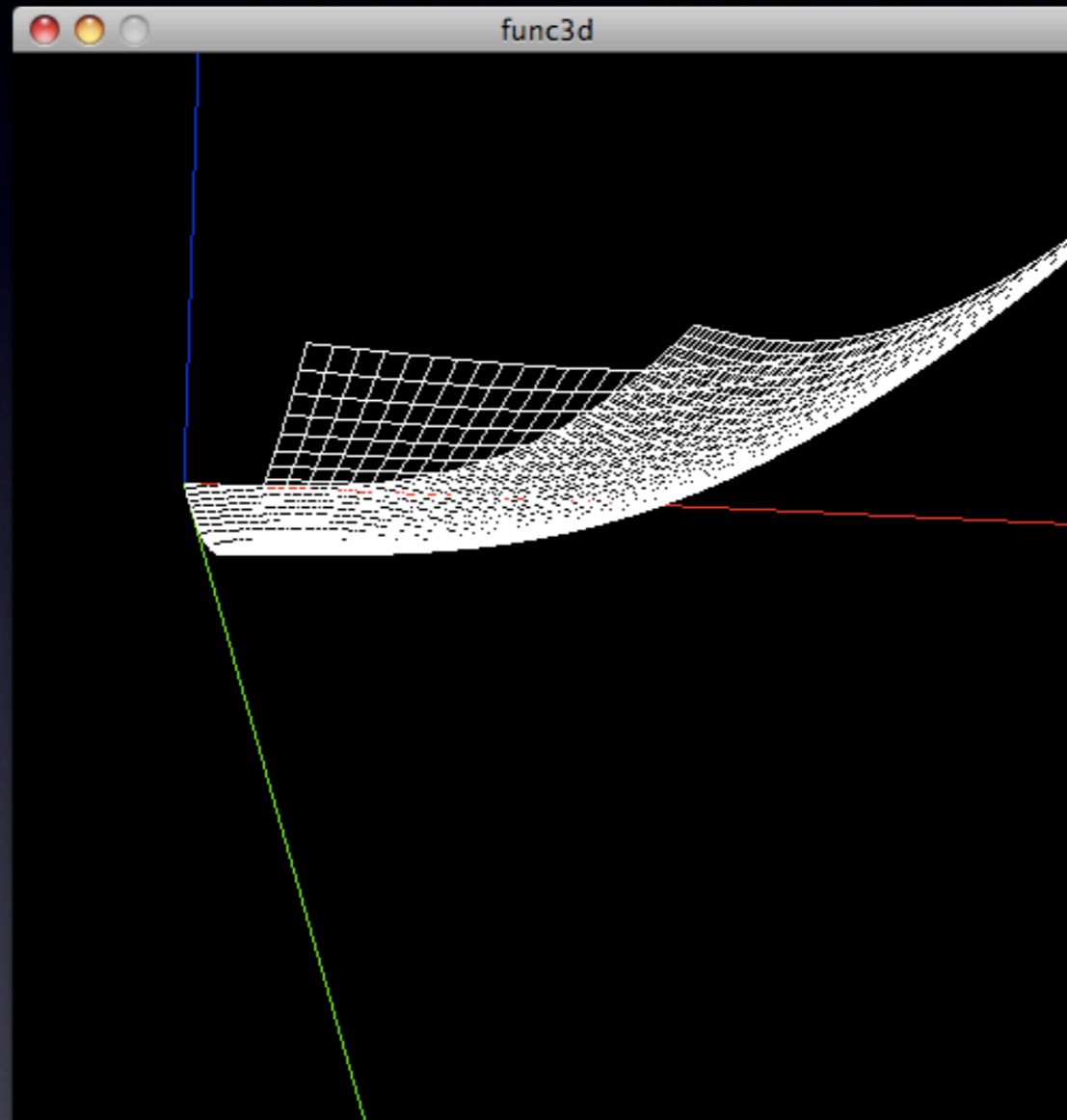
  for (j = 0, y = 0.0; j < steps; j++, y+=in_steps) {
    beginShape(QUAD_STRIP);
    for (i = 0, x = 0.0; i <= steps; i++, x+=in_steps) {
      vertex(x, y, matriz[i][j]);
      vertex(x, y + in_steps, matriz[i][j+1]);
    }
    endShape();
  }
}

void mousePressed()
{
  lastX = mouseX;
  lastY = mouseY;
}

void mouseDragged()
{
  distX = radians(mouseX - lastX);
  distY = radians(lastY - mouseY);
}

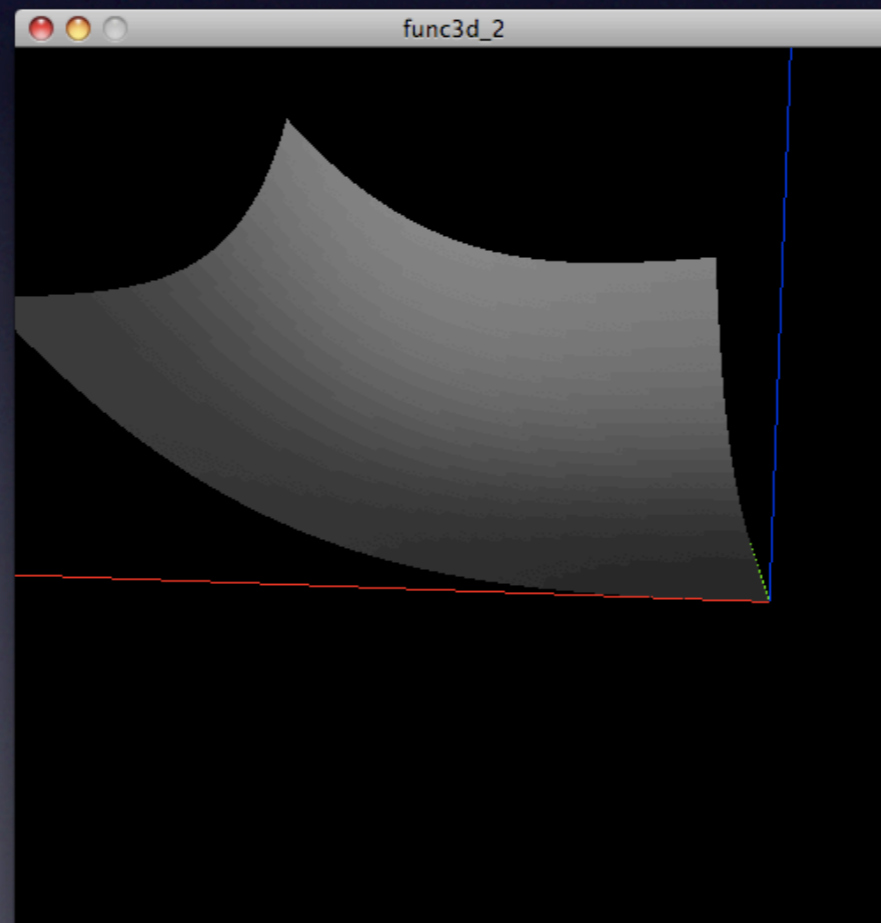
void mouseReleased()
{
  rotX += distY;
  rotY += distX;
  distX = distY = 0.0;
}
```

3D con processing



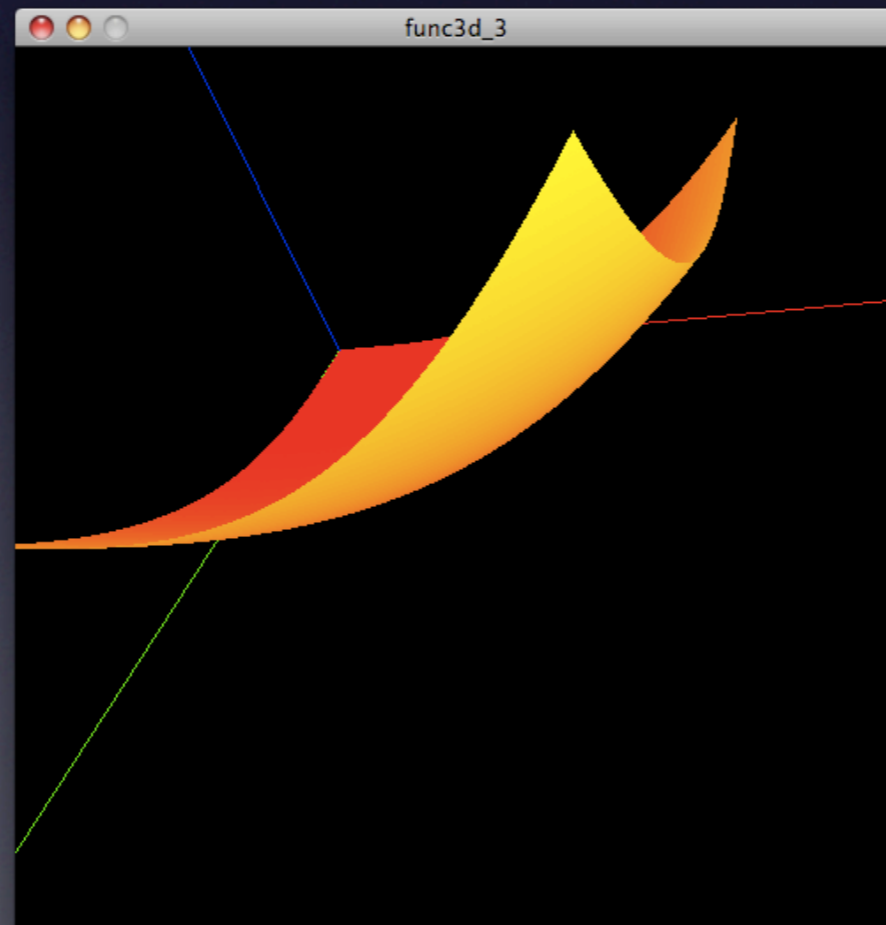
Práctica 7-2

- Modificar el programa anterior para dibujar la superficie de la función de forma sólida mediante iluminación básica (mediante `lights()` y un único `fill()`)



Práctica 7-3

- Modificar el programa anterior para dibujar la superficie mediante un gradiente de dos colores (por ejemplo, rojo valores bajos de z , amarillos valores altos)
- Usar para ello un `fill()` previo a cada `vertex()`



Práctica 7-4

- Modificar de nuevo el programa anterior para que en lugar de representar una función matemática, cargue una imagen y, tras convertirla en escala de grises, utilice estos niveles como valores a representar en la tira de cuadriláteros

