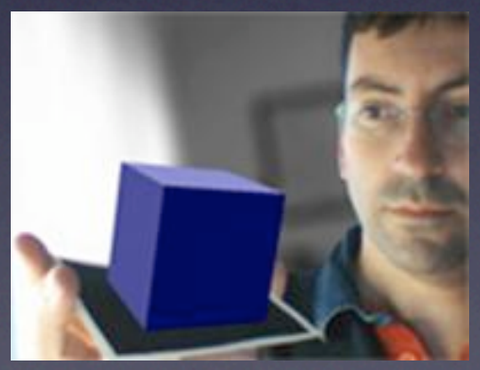
Computer Graphics

3D with 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 with processing

- *processing* offers two 3D rendering modes: P3D and OPENGL
- P3D is based on software, OPENGL is implemented using OpenGL (hardware). With some exceptions, both allow the same set of primitives and functions.
- 2D primitives and functions can be used in 3D (some exceptions might exist)
- Some of the primitives, such as lines (`line`), points (`point`), curves and shapes (`vertex` primitive), can define 3 coordinates: x, y, z
- The rest of the 2D primitives can also be used (with an implicit z=0)
- Stroke, fill, text and image (textures) functions can also be used in 3D

3D with processing

3D geometric transformations

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

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

Rotation x axis
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation y axis
$$\begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation z axis
$$\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3D with processing

3D geometric transformations

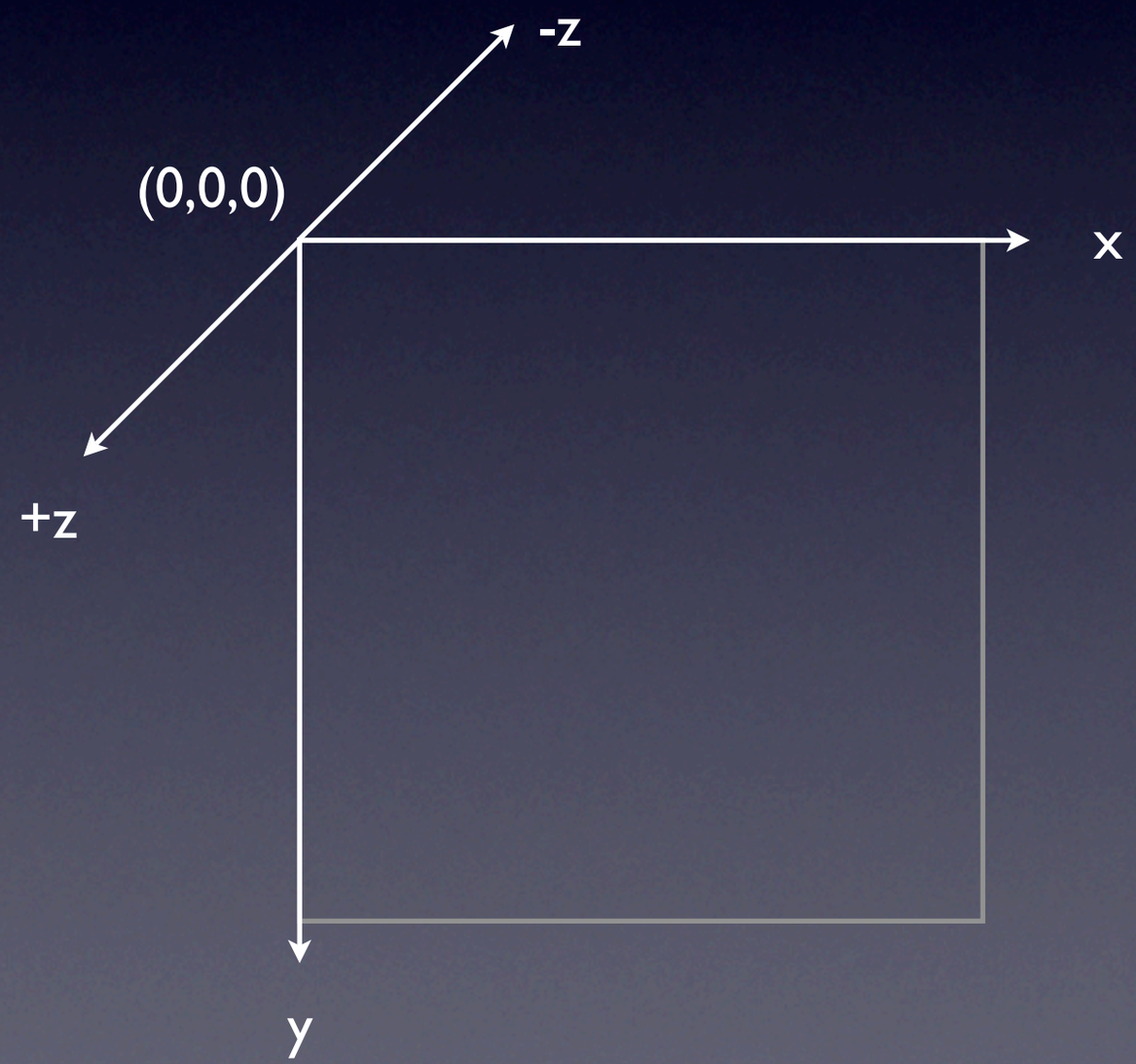
Translation
`translate(tx, ty, tz)`

Scale
`scale(sx, sy, sz)`

Rotation around an axis
`rotateX(), rotateY(), rotateZ()`

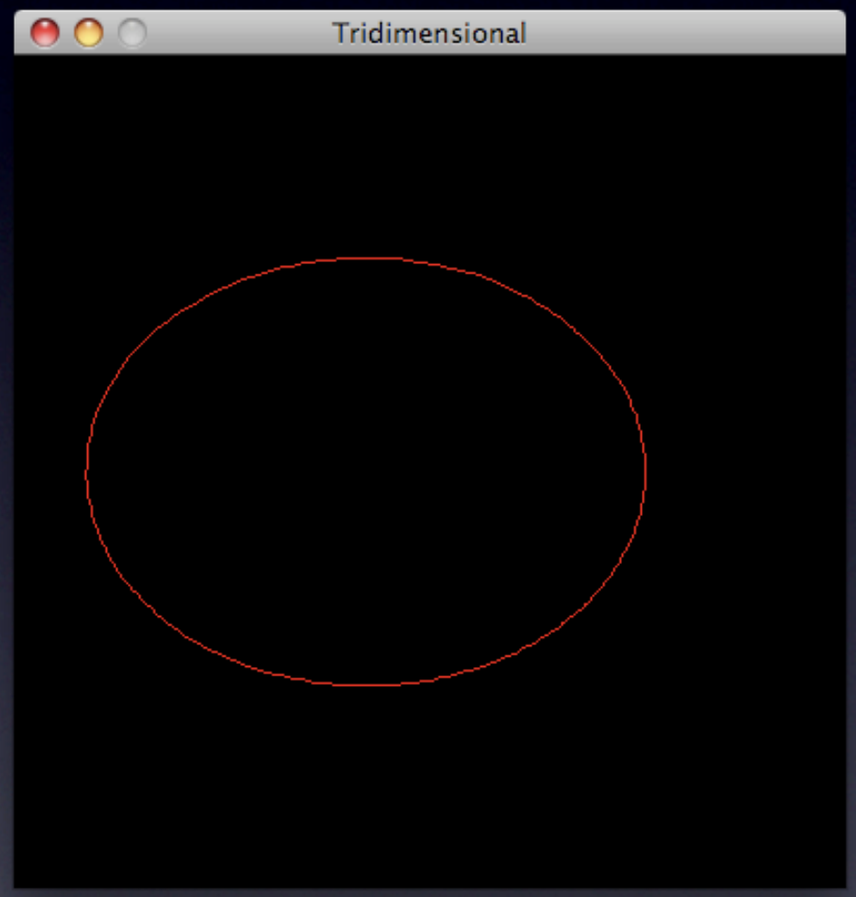
3D with processing

- By default, a perpective projection is applied
- (0,0,0) is placed in the left-top corner
- -z to move away



3D with processing

```
// An ellipse rotating
// around the y axis
float ang = 0.0;


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

void draw()
{
   background(0);

  // Drawing centered
  // (0,0,0)
  translate(width/2, height/2);

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



3D with processing

```
box(width, height, depth)
```

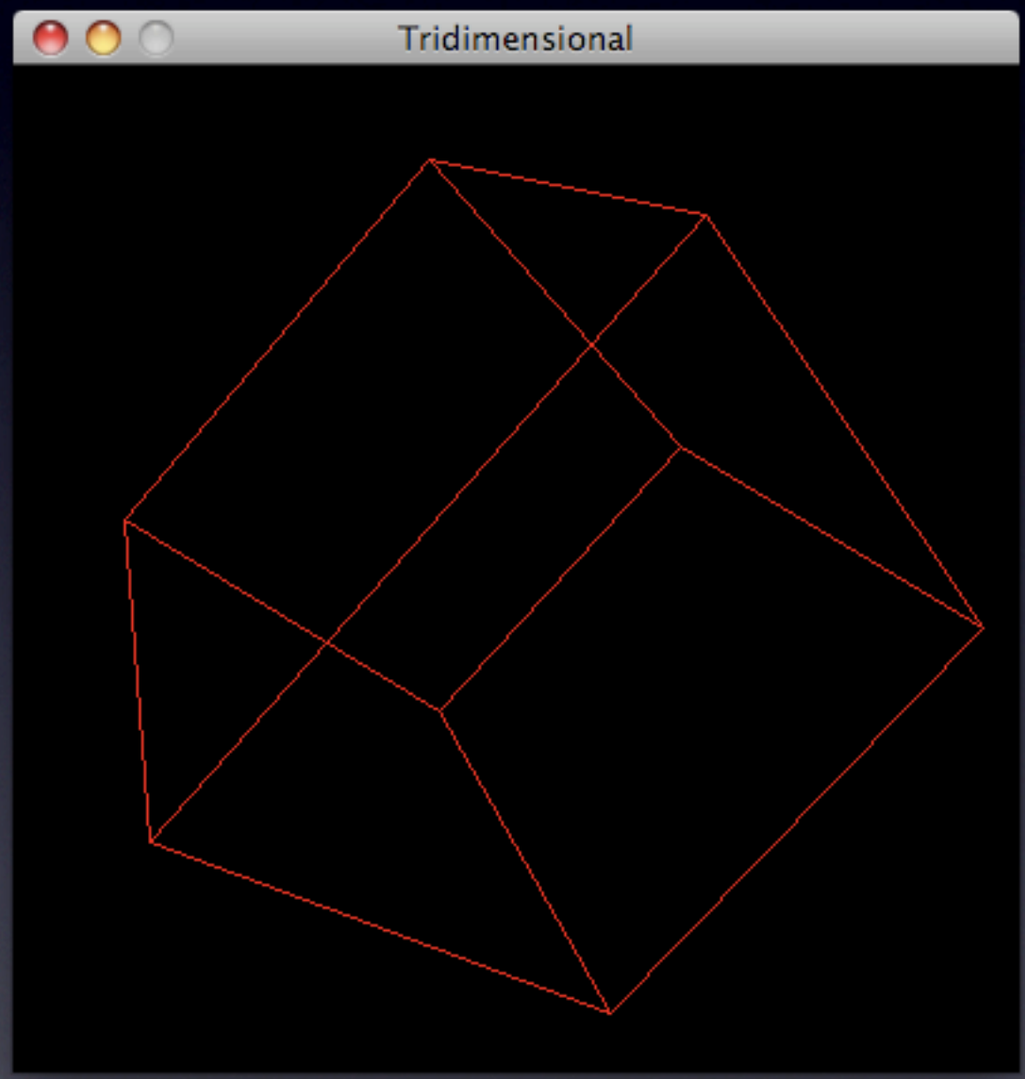
- Draws a cuboid, a right prism, centered in (0,0,0) of width (x), height (y) and depth (z) as arguments

```
sphere(radius)
```

- Draws a sphere centered in (0,0,0) with the specified radius
- The level of detail in which the sphere will be drawn can be adjusted with the `sphereDetail(n)` function, where `n` implies that vertices will be generated every `360°/n` (by default, `n = 30`)

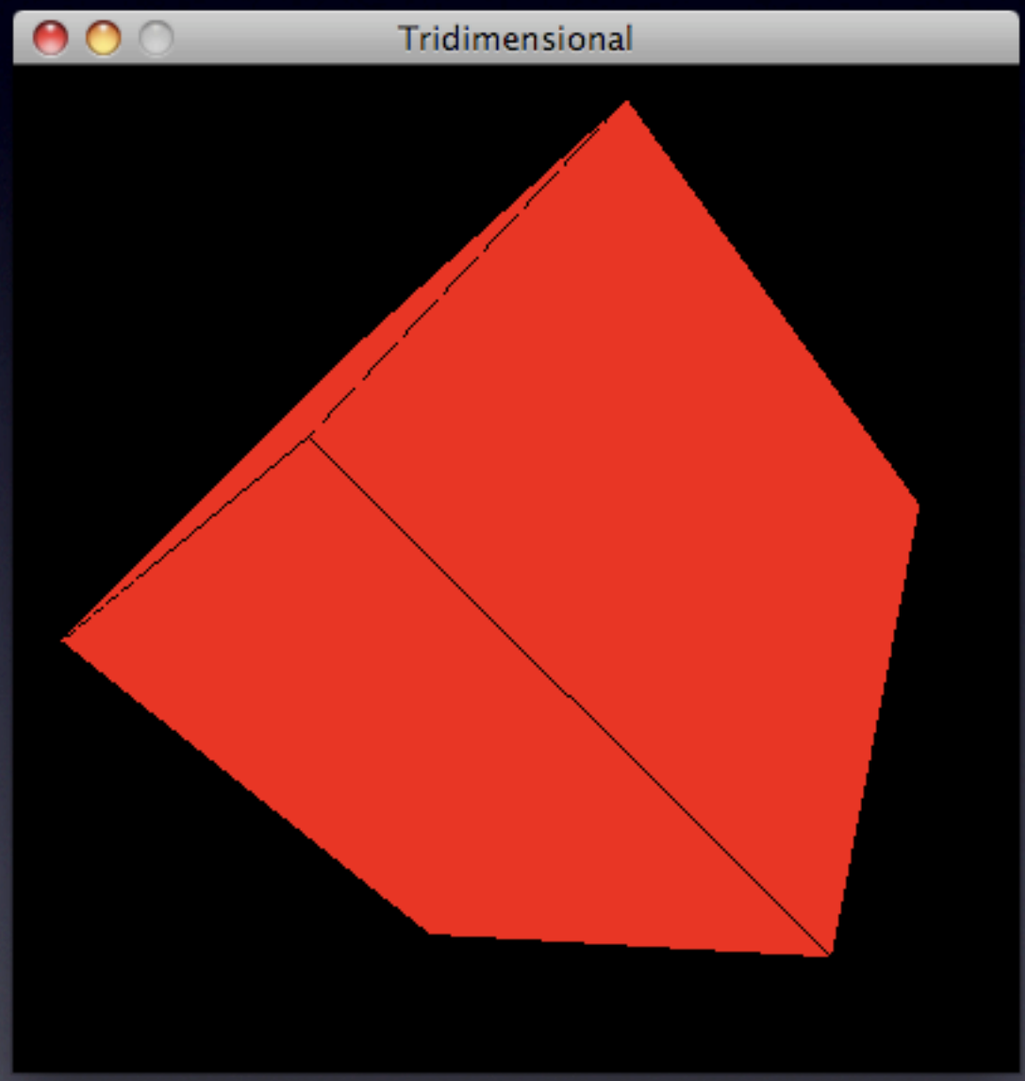
3D with processing

```
// A cube rotating
// around the three axes

// Wireframe version

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

void draw()
{
   background(0);

  // Drawing centered
  // in (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);
}
```



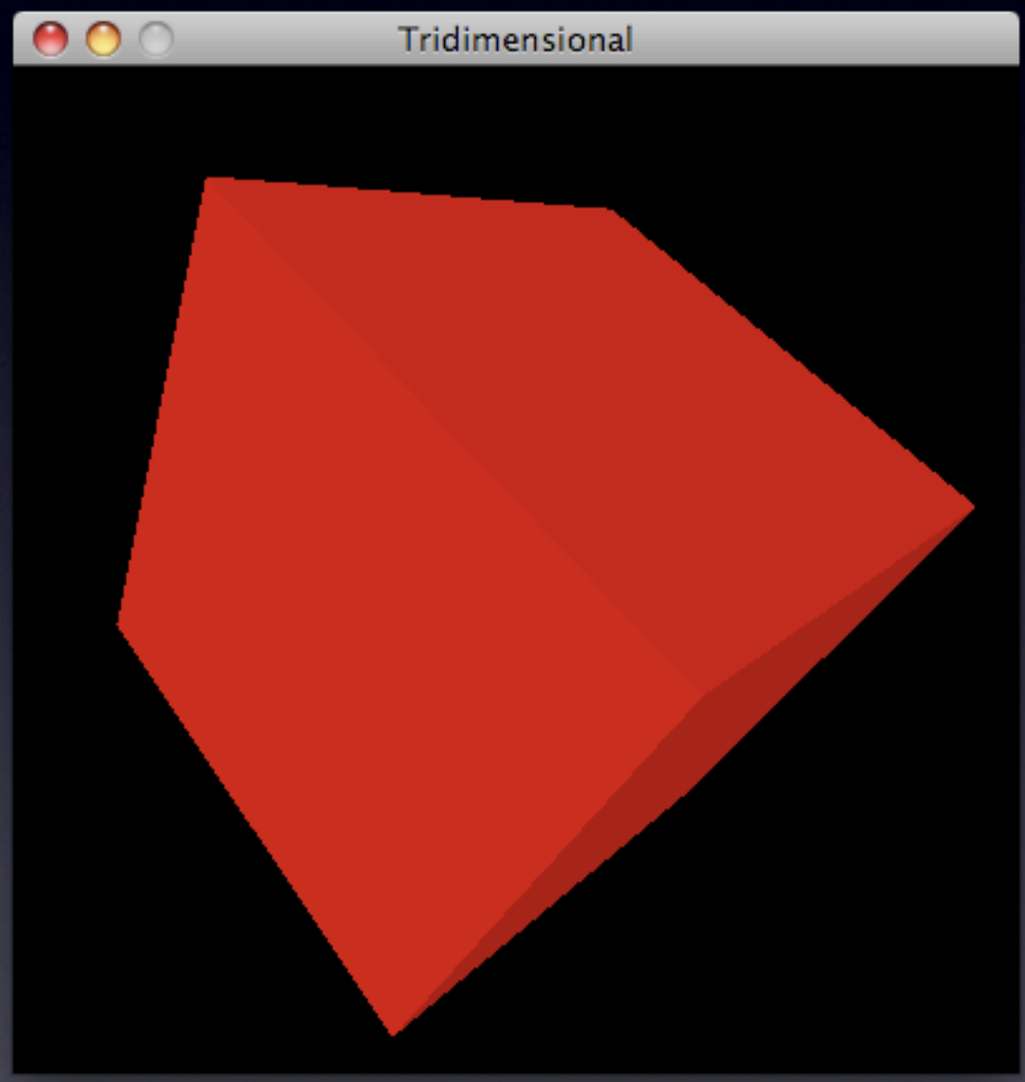
3D with processing

```
// A cube rotating
// around the three axes

// Solid version

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

void draw()
{
   background(0);

  // Drawing centered
  // in (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 with processing

```
// A cube rotating
// around the three axes

// Basic lighting version

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

void draw()
{
   background(0);

  // Iluminación básica
  lights();

  // Drawing centered
  // in (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 with processing

```
// Interactive cube
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;
}
```

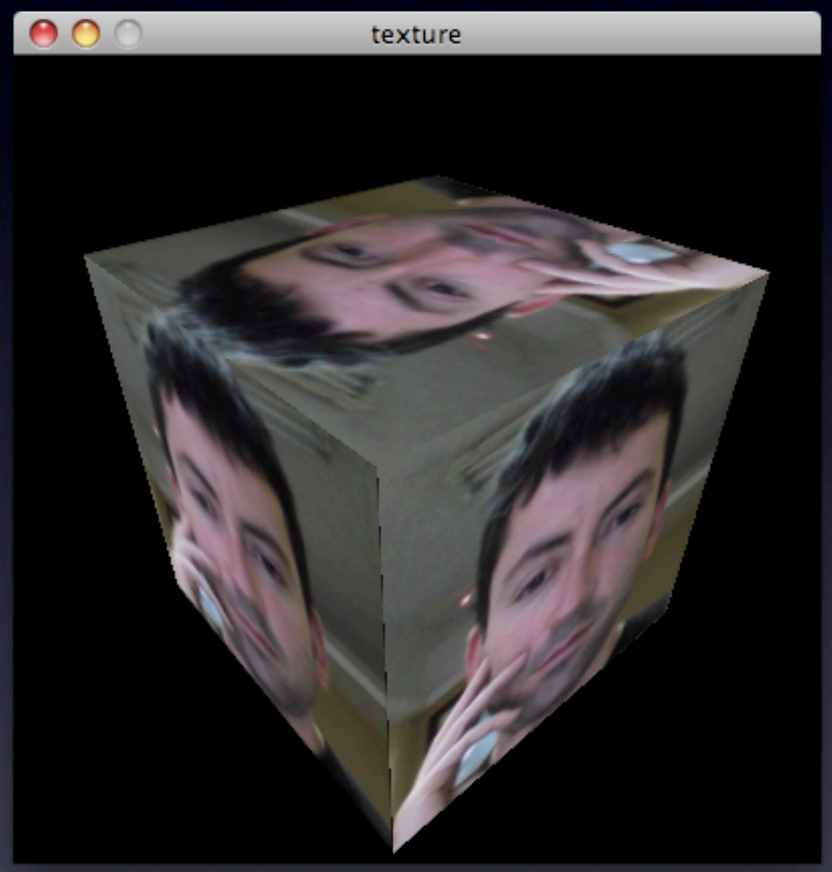
Practice 7-1

- Modify the previous program to allow zoom, by moving the cube through the z axis
- For that it will be needed to add a z component to the translation
- Initially this displacement will be 0 and will vary from 0 to -500 with steps of 10
- Use the `UP` and `DOWN` keys catching the `keyPressed()` event to change this zoom

3D with processing

```
// Let's do it with OpenGL
// The import is compulsory
import processing.opengl.*;

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

// Texture
PImage foto;

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

  // We want to work with texture
  // coordinates from (0,0) to (1,1)
  textureMode(NORMALIZED);
}

void draw(){
  background(0);

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

  // We want the cube 200 x 200 x 200
  // We draw it from -1 to 1
  scale(100, 100, 100);

  beginShape(QUADS);
  texture(foto);

// We provide the vertices of
// each face of the cube.
// The last two values
// are the texture coordinates
// that correspond to the
// vertex

// +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 with processing



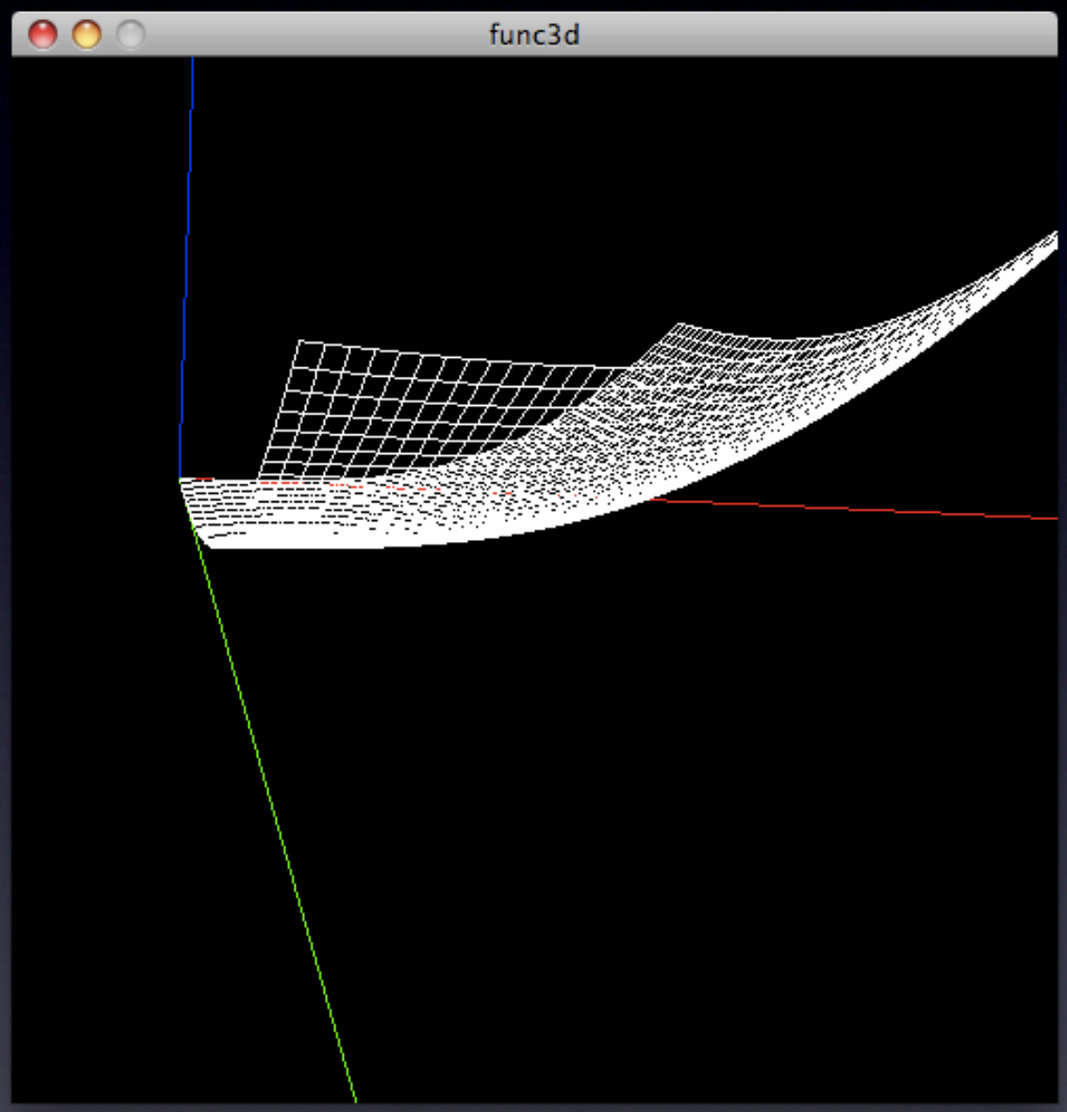
3D with processing

```
import processing.opengl.*;

// Drawing a 3D function
float rotX = 0.0, rotY = 0.0;
int lastX, lastY;
float distX = 0.0, distY = 0.0;

// Function steps
int steps = 50;

// z scale
float scaleZ = 200.0;

// z zoom
float zoomZ = -300.0;

// Graphic size
float gX = 500.0, gY = 500.0;

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

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

```
void draw()
{
  background(0);

  // We center the results on window
  translate(gX/2, gY/2, zoomZ);

  // Rotation
  rotateY(rotY + distX);
  rotateX(rotX + distY);

  // Centering around (0, 0);
  translate(-gX/2, -gY/2);

  // Function covers
  // 400 x 400 x scaleZ
  scale(gX, gY, scaleZ);

  // Drawing the function
  stroke(255);
  drawFunction();

  // Drawing axes
  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);
}
```

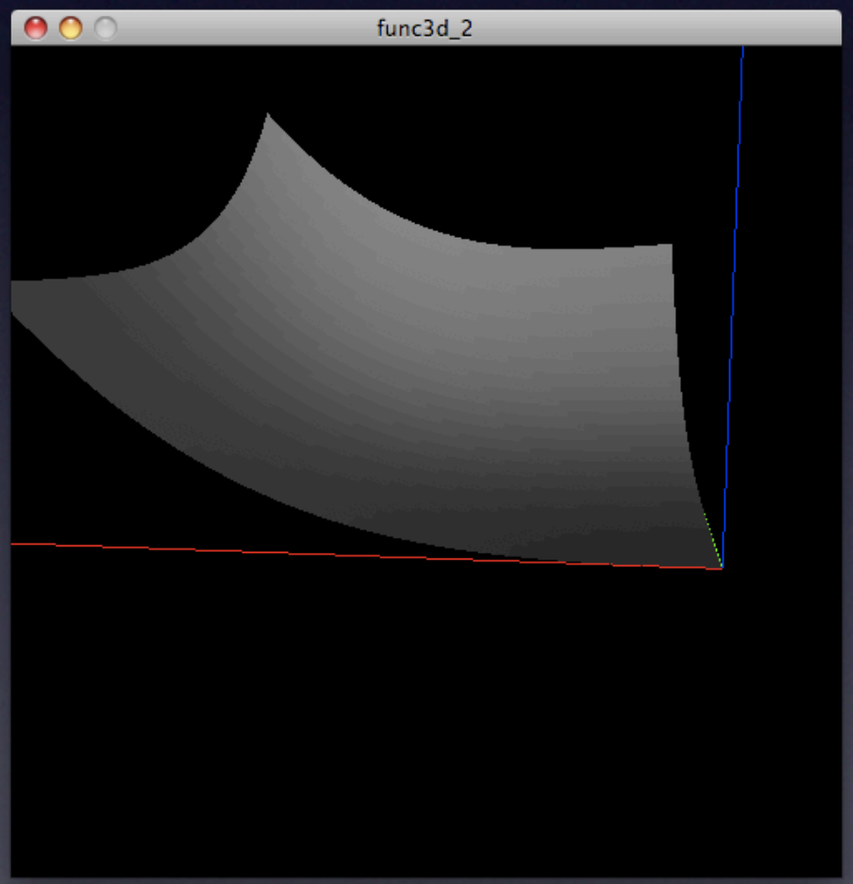
```
void drawFunction()
{
  float x, y, z;
  int i = 0, j = 0;
  float in_steps = 1.0 / steps;

  float[][] matrix = 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++)
      matrix[i][j] = function(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, matrix[i][j]);
        vertex(x, y + in_steps, matrix[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 with processing



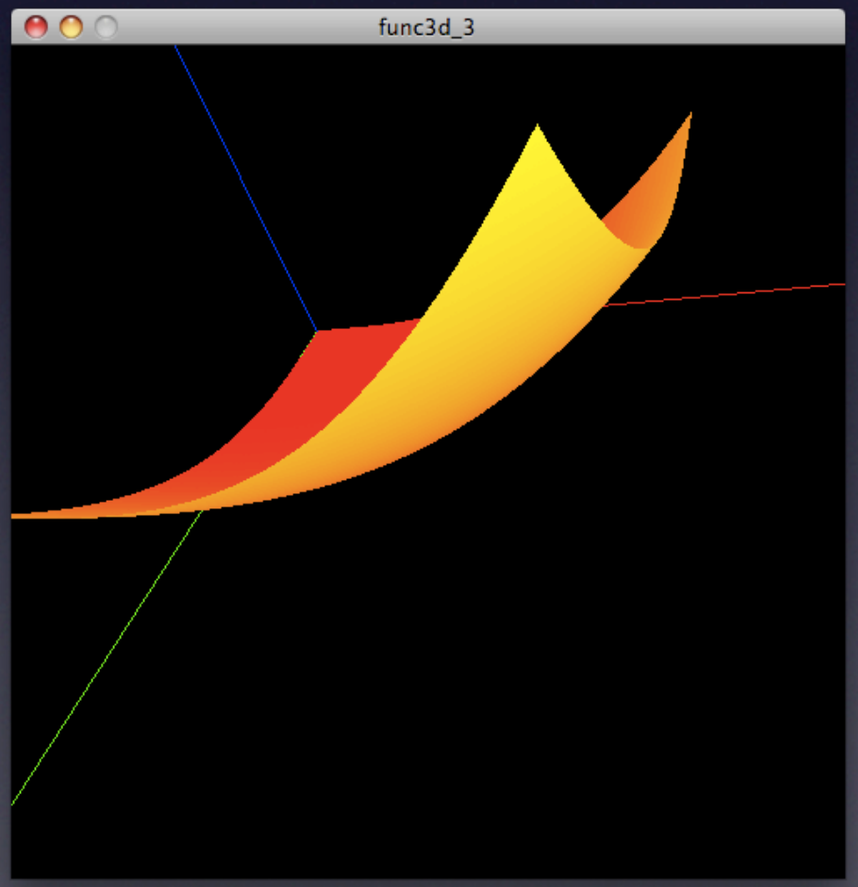
Practice 7-2

- Modify the previous program in order to draw the surface in solid mode using the basic lighting (`lights()` and a unique `fill()`)



Practice 7-3

- Modify the previous program to draw the surface with a two-color gradient (red for low values of z and yellow for high values, for instance)
- For that, use a `fill()` call before each `vertex()`



Practice 7-4

- Modify again the previous program to, instead of drawing a function, loads an image and, after grayscaling it, uses these values for the z values of the quadrilateral strips