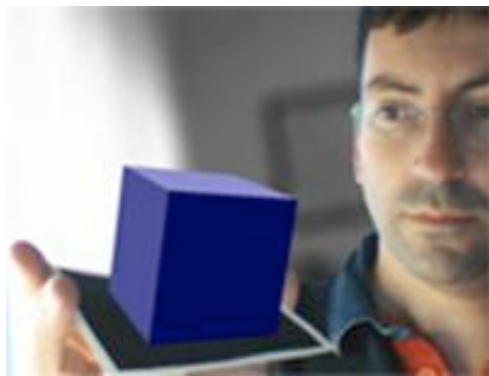


Gráficos por Computador

Animación



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>

Animación

- *processing* facilita enormemente la realización de animaciones mediante la función `draw()`
- Si en una aplicación definimos la función `draw()`, ésta se ejecutará continuamente en un constante bucle (en una hebra, thread, diferente)
- Inicialmente, la repetición de la función `draw()` está fijada en 60 veces por segundo
- Éste comportamiento puede cambiarse con la función `frameRate()`
- La función `frameRate()` marca un objetivo, pero su consecución depende de las posibilidades de la máquina en ejecutar el código de la función `draw()` en el framerate especificado

Animación

```
// Ejemplo clásico:
// Una bola rebotando por los cuatro
// lados de la ventana
// Si queremos animación tenemos que
// definir las funciones setup() y draw()

// Posición
int px, py;

// Velocidad
int vx, vy;

// Diámetro de la bola
int diametre = 20;

void setup()
{
  size(600, 300);
  fill(255);
  noStroke();

  // La bola en la mitad
  // de la ventana
  px = width/4;
  py = height/2;

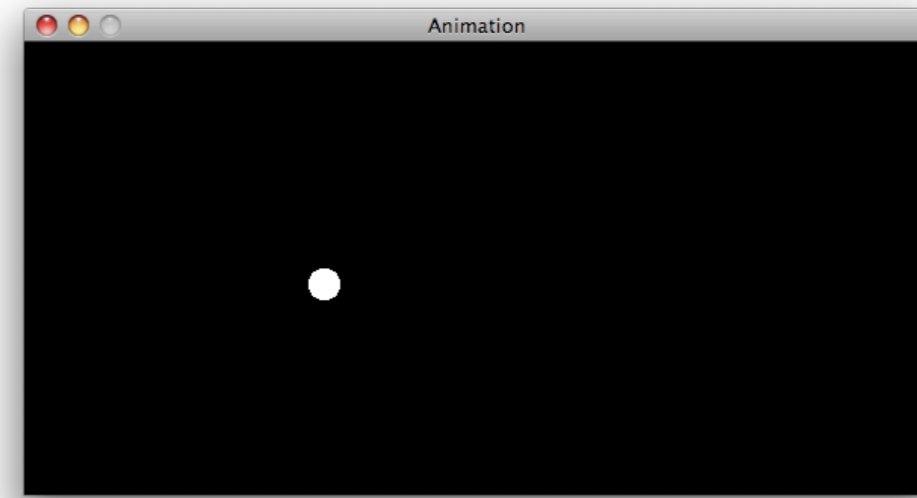
  // Velocidad inicial
  vx = vy = 1;
}
```

```
// Función draw()
// Se ejecutará en un continuo
// bucle sin fin
void draw()
{
  background(0);

  // Detectamos colisiones
  // y rebotamos
  if (px + diametre/2 > width - 1 ||
      px - diametre/2 < 0)
    vx *= -1;
  if (py + diametre/2 > height - 1 ||
      py - diametre/2 < 0)
    vy *= -1;

  // Actualizamos posiciones
  px += vx;
  py += vy;

  // Dibujamos
  ellipse(px, py, diametre, diametre);
}
```



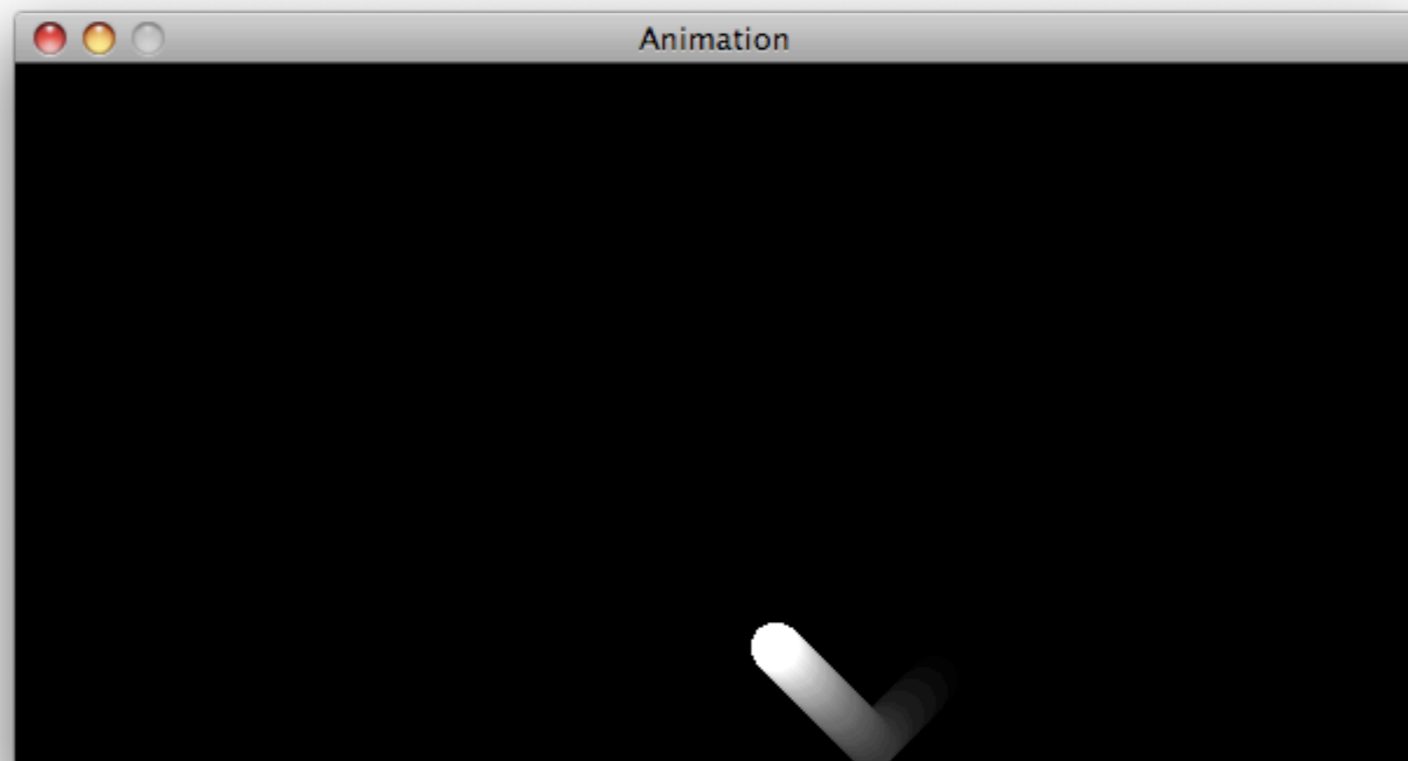
Animación

```
// Efecto rastro ...
// Substituyendo el background(0)
// por lo siguiente:

...
void draw()
{
    // Rellenamos toda la ventana
    // de color RGB (0, 0, 0) y
    // transparencia 20
    fill(0, 20);
    rect(0, 0, width, height);

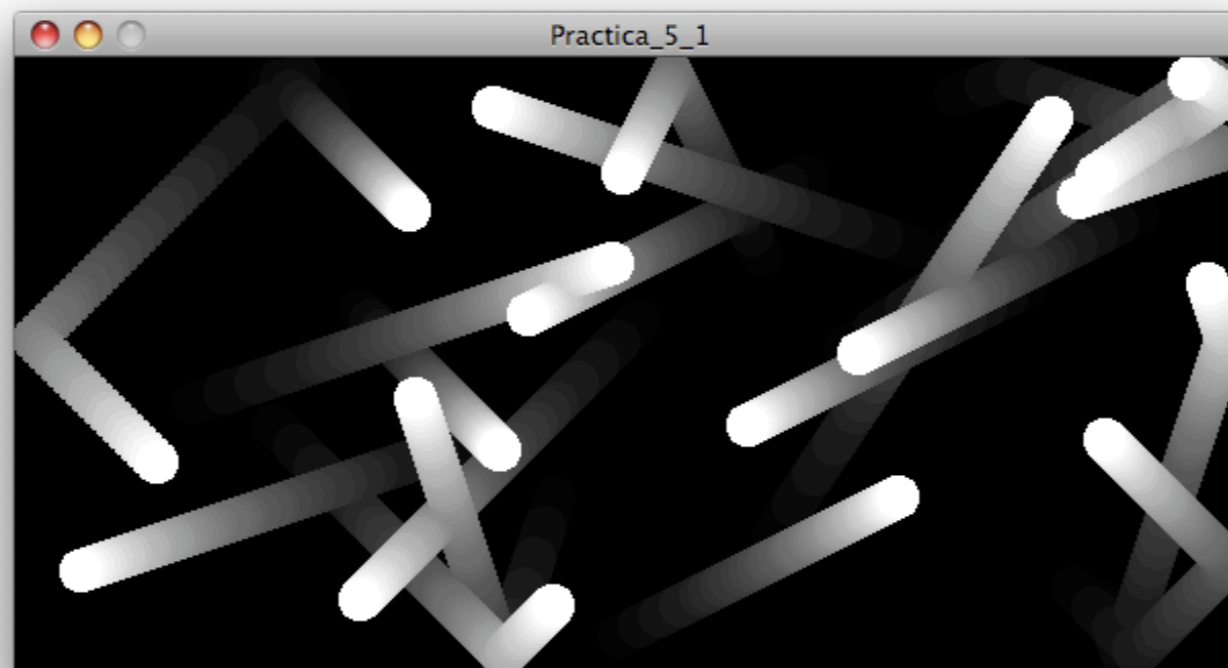
    // Dejamos fill a blanco para
    // dibujar la bola
    fill(255);

    ...
}
```



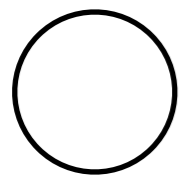
Práctica 5-1

- Modificar la aplicación anterior para visualizar n bolas rebotando por la ventana. Utilizad arrays para almacenar los valores de posición y velocidad de cada una de las bolas
- Las posiciones iniciales deben ser aleatorias teniendo en cuenta el diámetro de las bolas y las dimensiones de la ventana
- Las velocidades deben ser también iniciadas aleatoriamente con valores de entre -4 a $+4$ (cero excluido)



Animación

- Considerando la fuerza de la gravedad:



$$F = m \cdot g$$



Gravedad (g)

Integración por Euler:

$$a = g$$

$$v(t+1) = v(t) + a$$

$$e(t+1) = e(t) + v(t+1)$$



Animación

```
// Consideramos el efecto
// de la gravedad

// Posición
float px, py;

// Velocidad
float vx, vy;

// Diámetro de la bola
int diámetro = 20;

// Gravedad
float gravedad = 0.5;

void setup()
{
    size(600, 300);
    fill(255);
    noStroke();

    // La bola en la mitad
    // de la ventana
    px = width/4;
    py = diámetro/2;

    // Velocidad inicial
    vx = vy = 1.0;

    // Relleno inicial
    background(0);
}
```

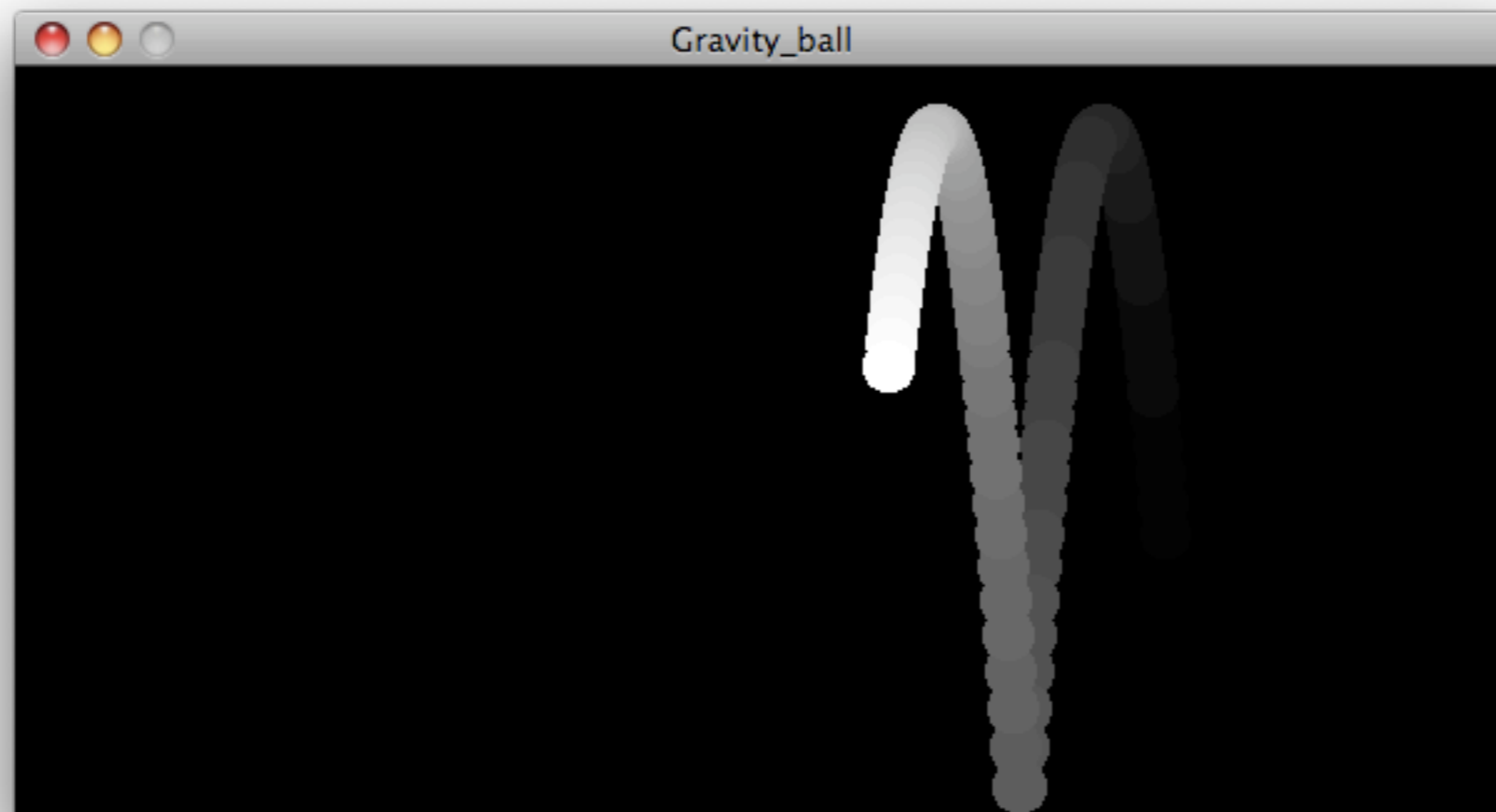
```
void draw()
{
    fill(0, 20);
    rect(0, 0, width, height);
    fill(255);

    // Detectamos colisiones
    // y rebotamos
    if (px + diámetro/2 > width - 1)
    {
        vx *= -1;
        px = width - 1 - diámetro/2;
    }
    if (px - diámetro/2 < 0)
    {
        vx *= -1;
        px = diámetro/2;
    }
    if (py + diámetro/2 > height - 1)
    {
        vy *= -1;
        py = height - 1 - diámetro/2;
    }
    if (py - diámetro/2 < 0)
    {
        vy *= -1;
        py = diámetro/2;
    }

    // Actualizamos posiciones
    vy += gravedad;
    px += vx;
    py += vy;

    // Dibujamos
    ellipse(px, py, diámetro, diámetro);
}
```

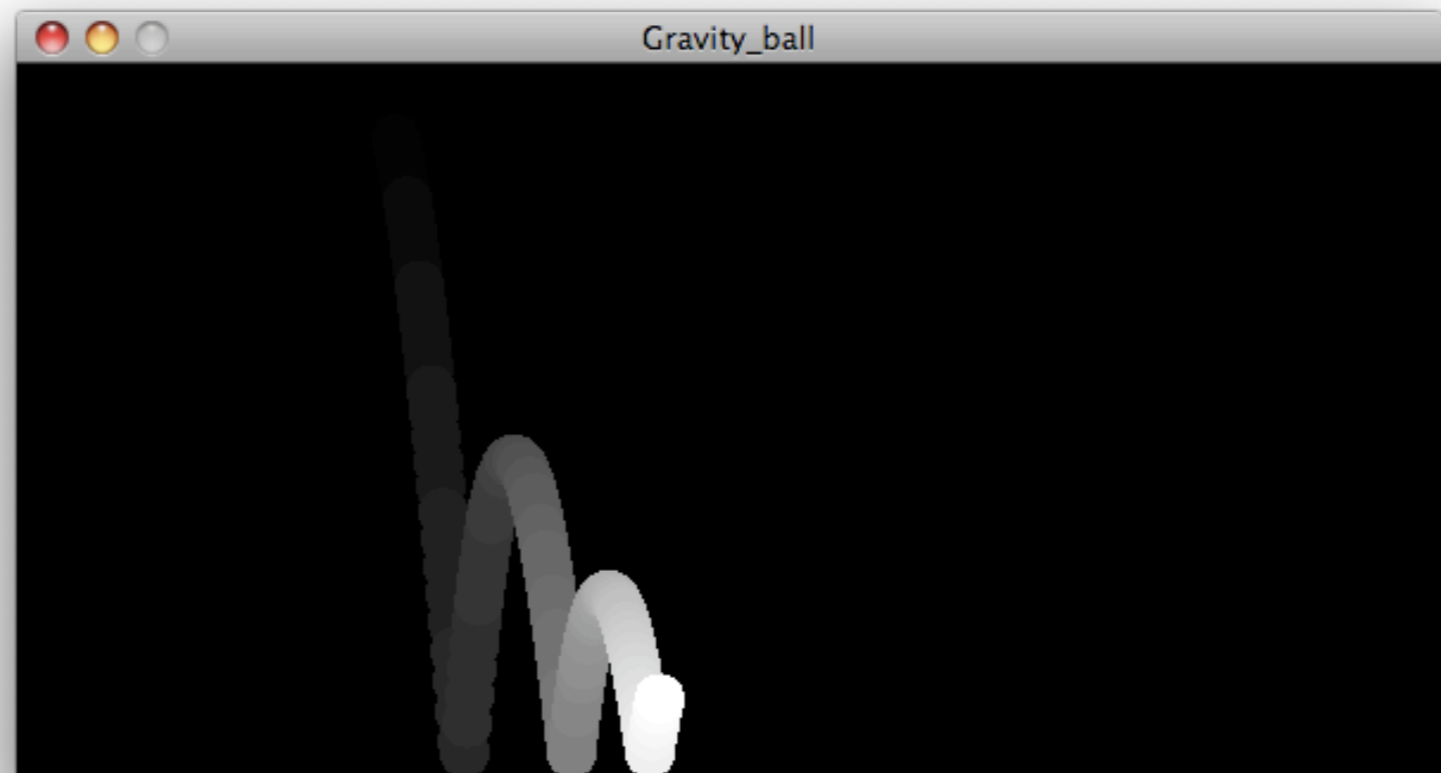
Animación



Animación

- Para que el comportamiento de la bola se asemeje a la realidad, debe considerarse la amortiguación producida por su paso por un fluido (la atmósfera)
- Este efecto se consigue simplemente amortiguando la velocidad, de la siguiente forma:

```
// Actualizamos posiciones  
vy += gravedad;  
vy *= 0.98; // Esto es nuevo !  
px += vx;  
py += vy;
```



Animación

```
// Elipse continuamente rotando
float angulo = 0.0;

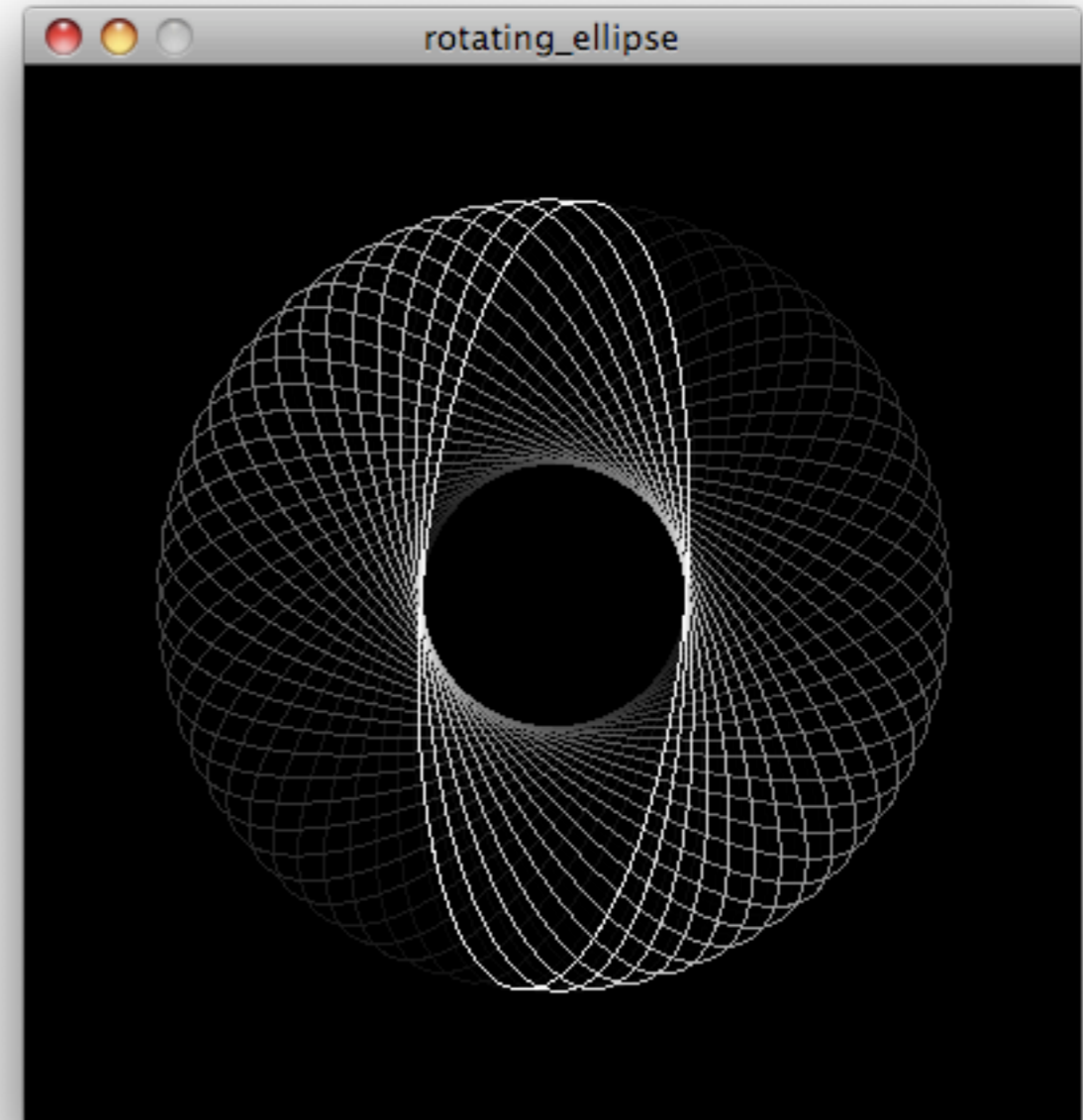
void setup()
{
  size(400, 400);
}

// Atención !
// draw() inicia la matriz
// modelo/vista a la identidad
void draw()
{
  fill(0, 20);
  noStroke();
  rect(0, 0, width, height);

  noFill();
  stroke(255);

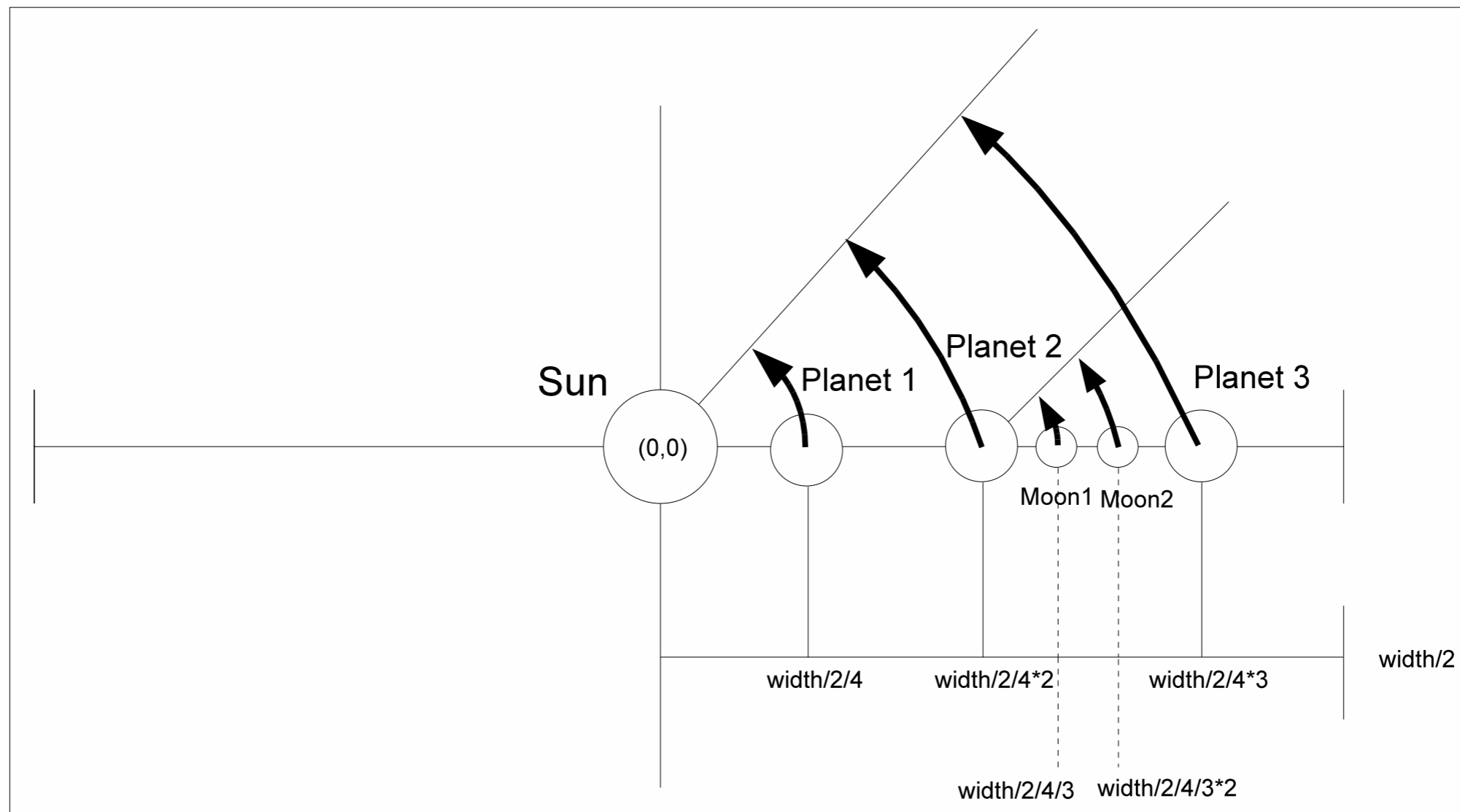
  // Rotamos respecto al centro de la
  // ventana. Para ello, debemos
  // trasladar este centro al (0,0)
  // antes de llevar a cabo la
  // rotación. OJO! al orden de las
  // operaciones (de abajo a arriba)
  translate(width/2, height/2);
  rotate(angulo+=0.1);
  translate(-width/2, -height/2);

  ellipse(width/2, height/2, 100, 300);
}
```



Animación

Un sistema solar:



Animación

```
// Un sistema planetario
// en 2D
// Un sol, 3 planetas y dos lunas
float angPlaneta1 = 0.0,
      angPlaneta2 = PI/3.0,
      angPlaneta3 = 2.0*PI/3.0,
      angLuna1 = 0.0,
      angLuna2 = PI;

void setup()
{
  size(400, 400);
  stroke(255);
  frameRate(30);
}

void draw()
{
  background(0);

  // Dibujaremos todo centrado
  // en el (0,0) y resolveremos
  // sus posiciones finales con
  // transformaciones 2D

  // El sol en el centro de
  // nuestro universo
  translate(width/2, height/2);

  // Sol
  fill(#F1FA03); // Hex. mediante color selector
  ellipse(0, 0, 20, 20);
  pushMatrix();

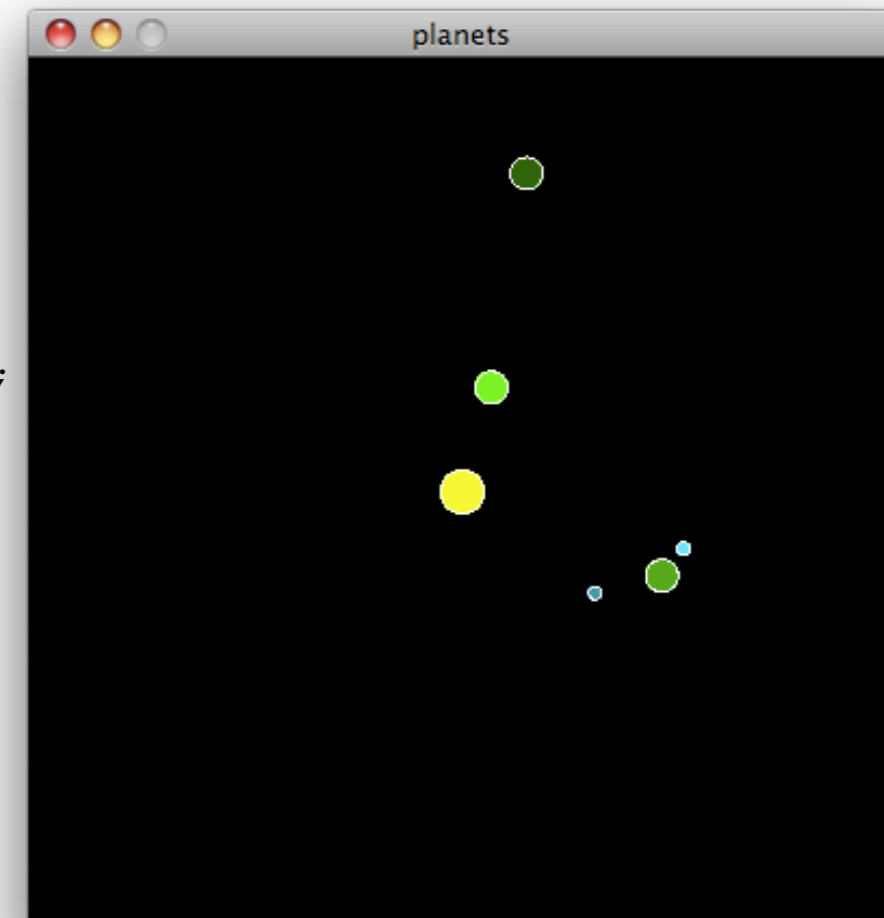
  // Planeta 1
  rotate(angPlaneta1 += 0.1);
  translate(width/2/4, 0);
  fill(#05FA03);
  ellipse(0, 0, 15, 15);

  // Planeta 2
  popMatrix();
  pushMatrix();
  rotate(angPlaneta2 += 0.05);
  translate(width/2/4*2, 0);
  fill(#0BA00A);
  ellipse(0, 0, 15, 15);

  // Luna 1
  pushMatrix();
  rotate(angLuna1 += 0.1);
  translate(width/2/4/3, 0);
  fill(#08E4FF);
  ellipse(0, 0, 6, 6);

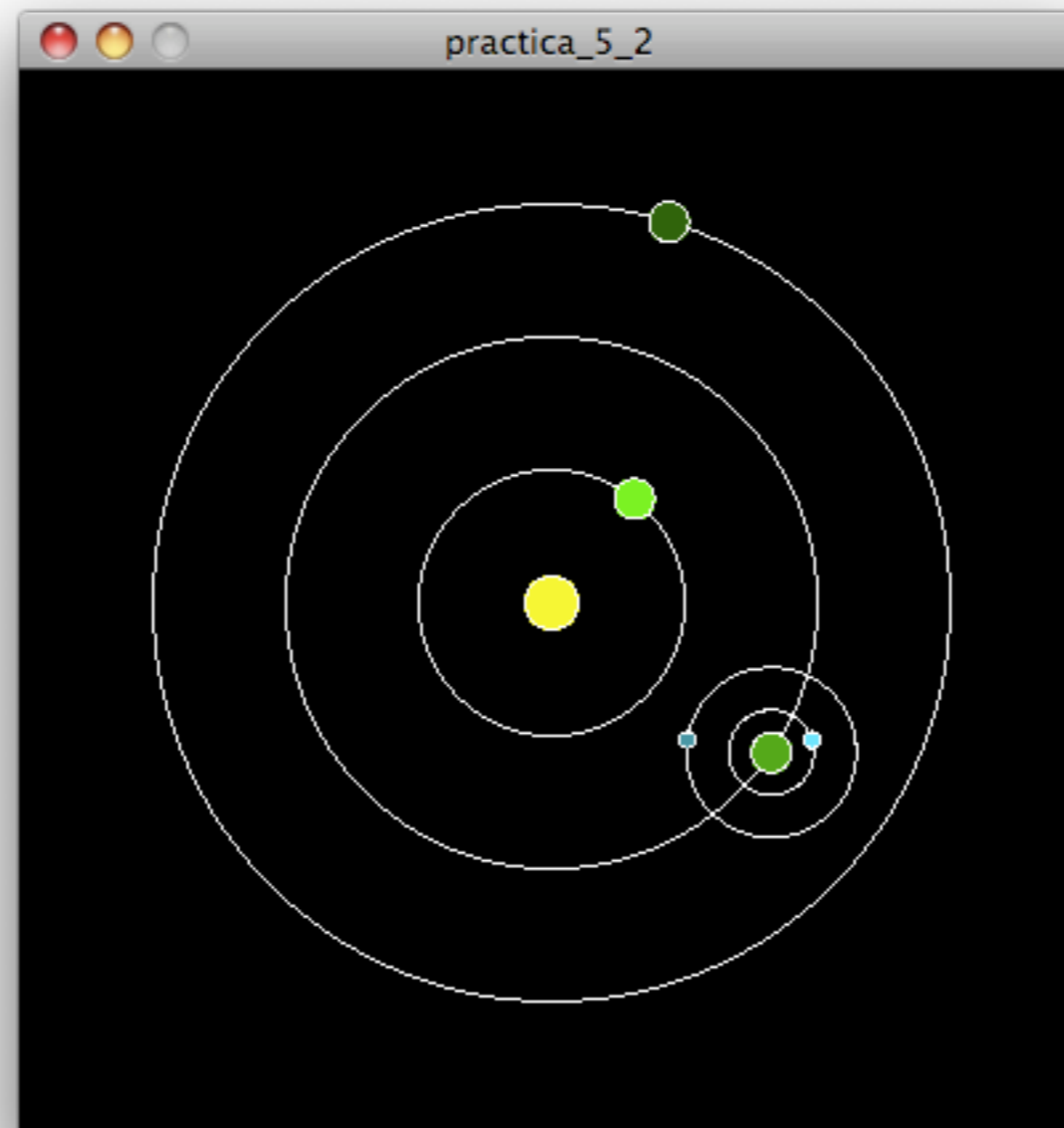
  // Luna 2
  popMatrix();
  rotate(angLuna2 += 0.05);
  translate(width/2/4/3*2, 0);
  fill(#118998);
  ellipse(0, 0, 6, 6);

  // Planeta 3
  popMatrix();
  rotate(angPlaneta3 += 0.025);
  translate(width/2/4*3, 0);
  fill(#075806);
  ellipse(0, 0, 15, 15);
}
```



Práctica 5-2

- Modificar el sistema planetario para que se dibuje las órbitas de los planetas y lunas:



Práctica 5-3

- Conseguir que el sol también orbite respecto al centro de la ventana:

