Animation

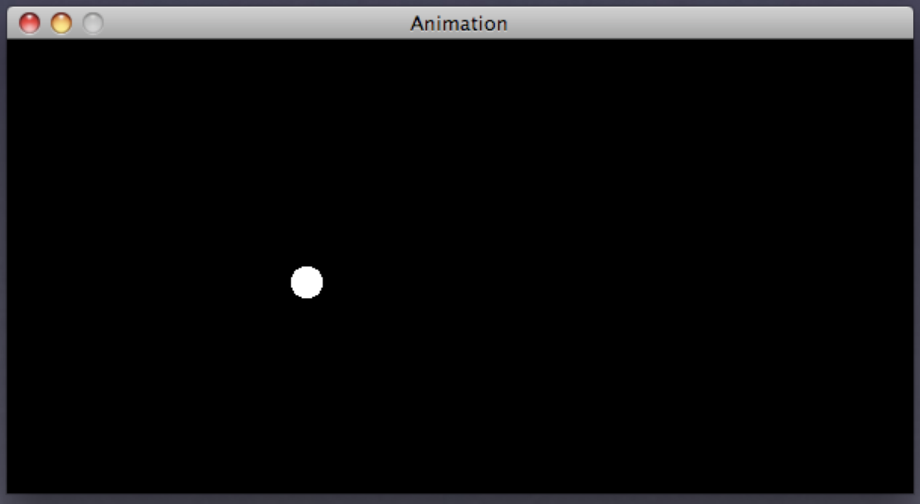
Animation

- *processing* makes animation very easy by using the function `draw()`
- If we define the `draw()` function in our application, it will be continuously executed in a constant loop (in a different thread)
- Initially, `draw()` repetition rate is fixed to 60 times per second This behaviour can be chnaged with `frameRate()` function
- `frameRate()` function establish an objective or target, but
- to achieve that depends on the possibilities of the machine in executing the `draw()` function at the specified framerate

Animation

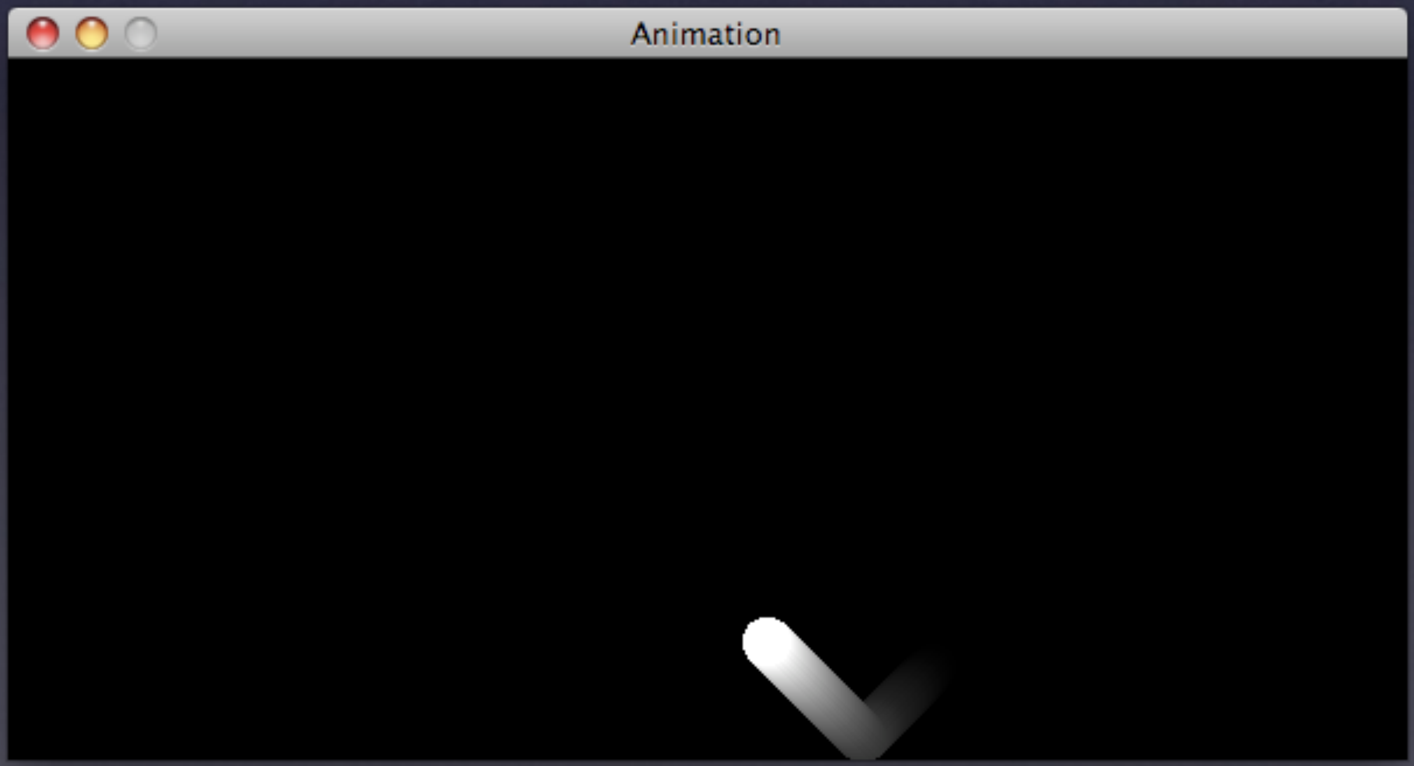
```
// A classic example:
// A bouncing ball
// If we want animation we have to
// define setup() and draw() functions

// Position
int px, py;

// Velocity
int vx, vy;

// Ball diametre
int diametre = 20;

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

  // Ball initial
  // position
  px = width/4;
  py = height/2;

  // Initial velocity
  vx = vy = 1;
}
```

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

  // Detecting collisions and
  // bouncing
  if (px + diametre/2 > width - 1 ||
      px - diametre/2 < 0)
    vx *= -1;
  if (py + diametre/2 > height - 1 ||
      py - diametre/2 < 0)
    vy *= -1;

  // Updating positions
  px += vx;
  py += vy;

  // Drawing the ball
  ellipse(px, py, diametre, diametre);
}
```



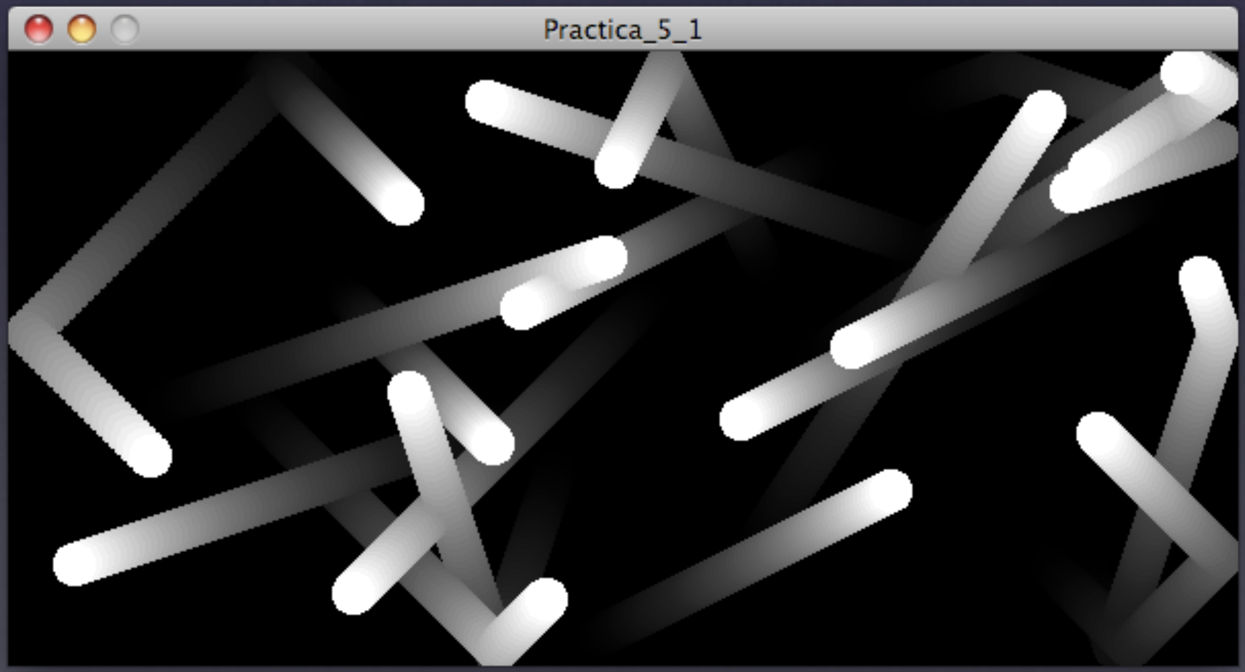
Animation

```
// Fading effect...
// By substituting background(0)
// with the following:

...
void draw()
{
      // We fill all the window
      // with the RGB color (0, 0, 0) and
      // transparency 20
      fill(0, 20);
      rect(0, 0, width, height);

      // Fill to 255 to draw
      // the ball
      fill(255);

      ...
```



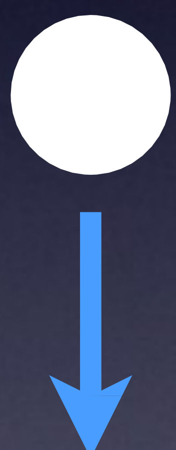
Practice 5-1

- Modify the previous application to visualise n boucing balls. Use arrays to manage the velocity and position values of each ball
- Initial positions should be random taking into account the diameter of the balls and the window size
- Velocities should also be random with values from -4 to 4 (zero not included)



Animation

- Considering gravity force:



$\boxed{F = m \cdot g}$

Gravity (g)

Euler integration:

$a = g$

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

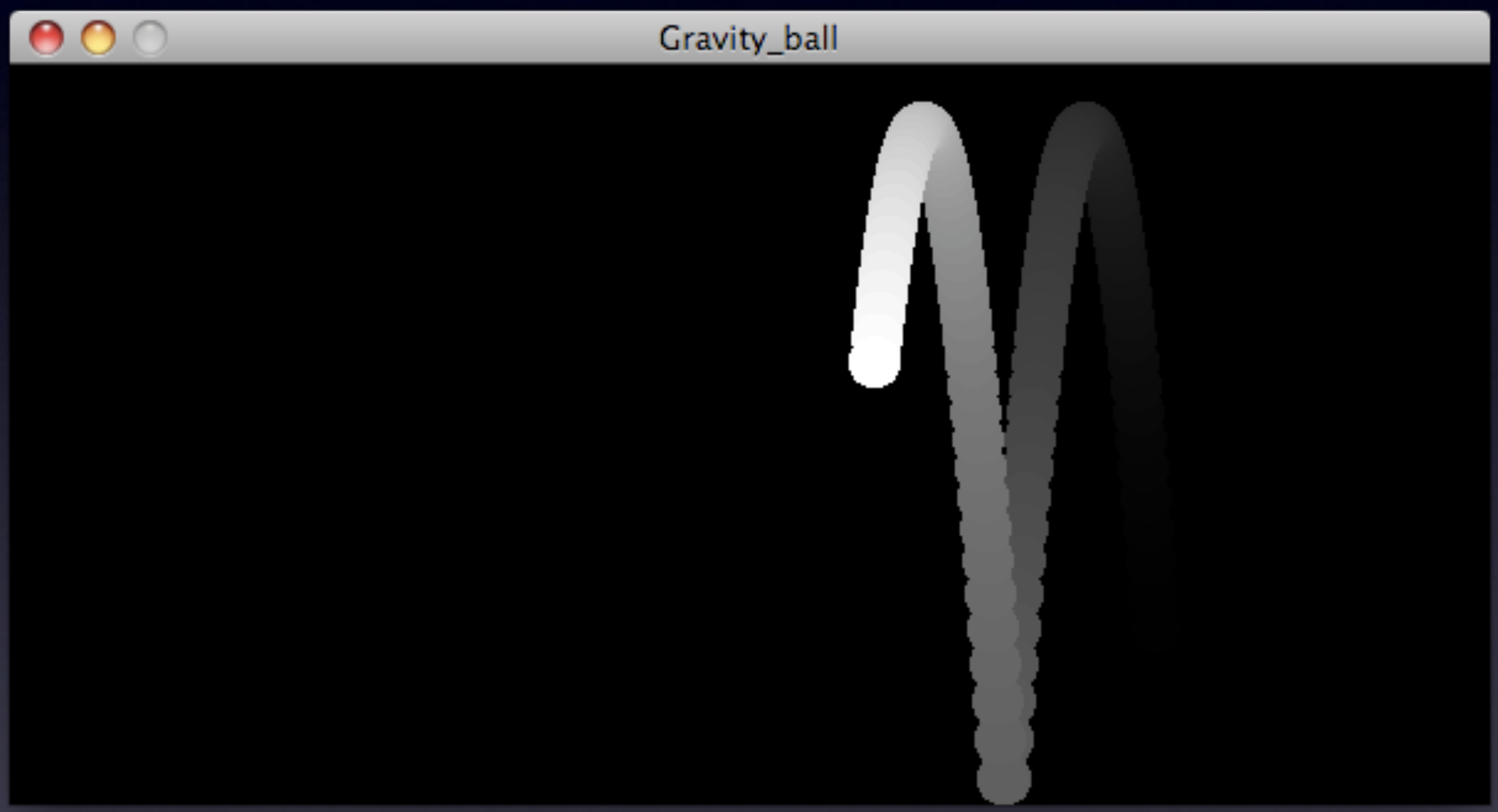
$+1) = e(t) + v(t+1)$

Animation

```
// We consider the
// effect.

// Position
float px, py;

// Velocity
float vx, vy;

// Ball diameter
int diametre = 20;

// gravity
float gravity = 0.5;

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

  // Ball initial position
  px = width/4;
  py = diametre/2;

  // Initial velocity
  vx = vy = 1.0;

  // Initial background
  background(0);
}
```

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

  // Detecting collisions and
  // bouncing
  if (px + diametre/2 > width - 1)
  {
    vx *= -1;
    px = width - 1 - diametre/2;
  }
  if (px - diametre/2 < 0)
  {
    vx *= -1;
    px = diametre/2;
  }
  if (py + diametre/2 > height - 1)
  {
    vy *= -1;
    py = height - 1 - diametre/2;
  }
  if (py - diametre/2 < 0)
  {
    vy *= -1;
    py = diametre/2;
  }

  // Updating positions
  vy += gravity;
  px += vx;
  py += vy;

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

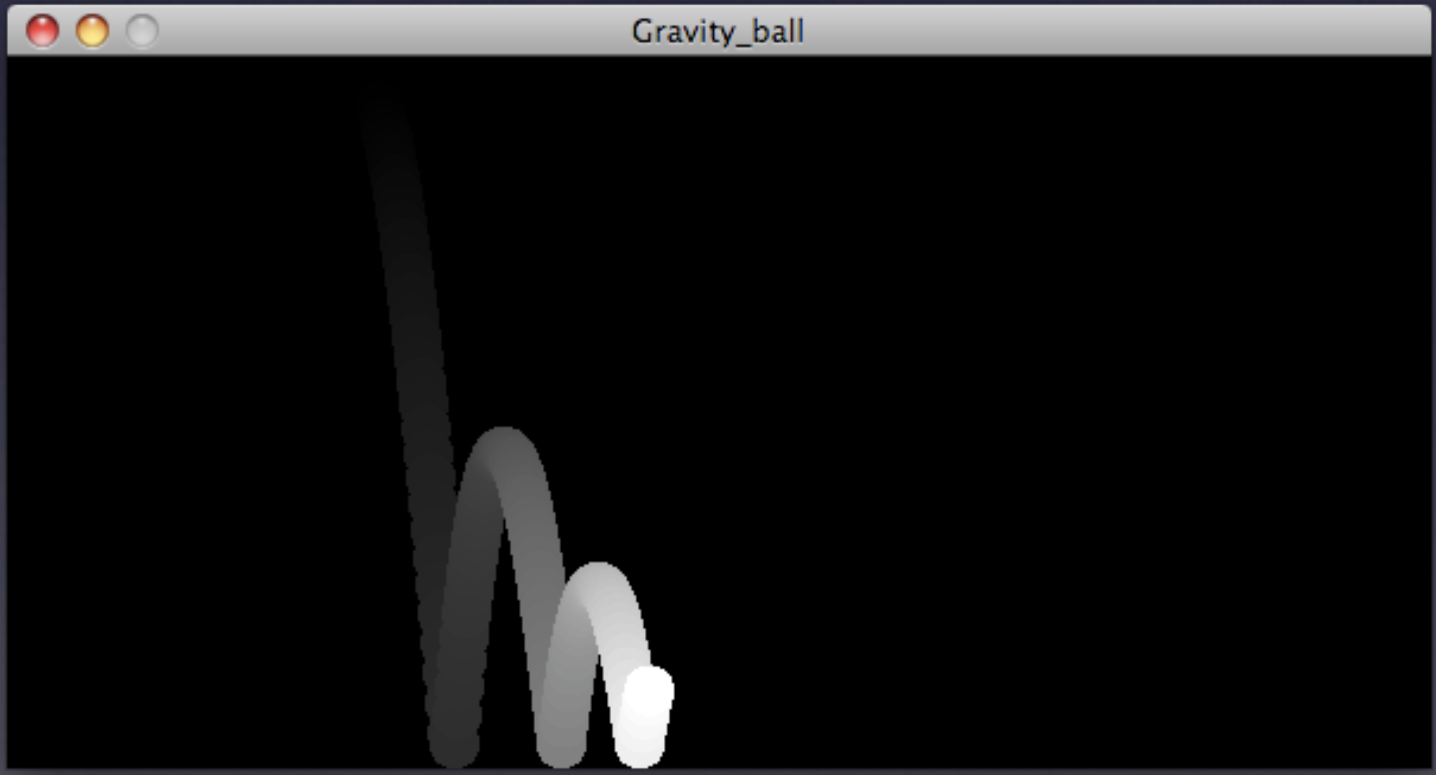
Animation



Animation

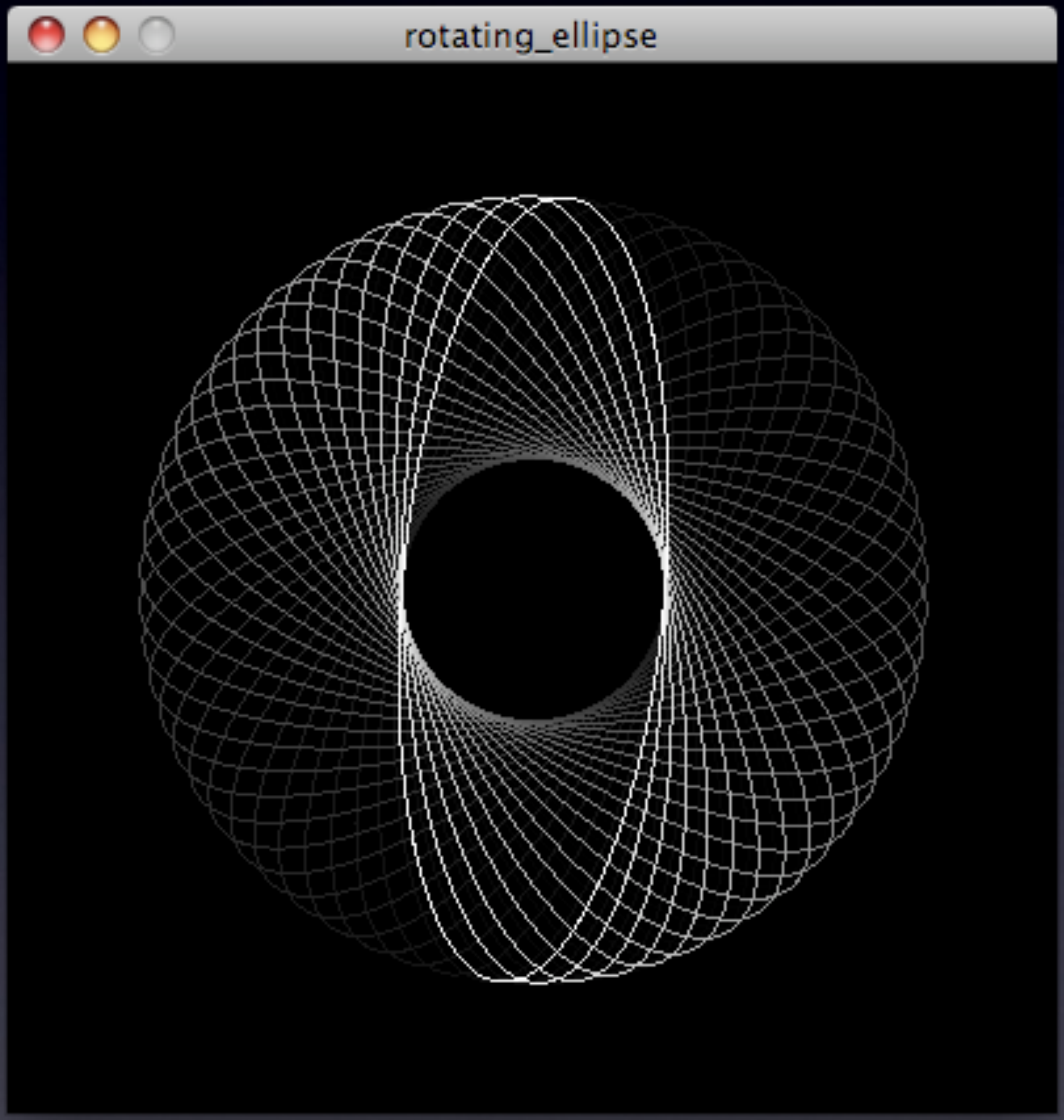
- To achieve a similar behaviour to the real one, the damping effect due to the pass of the ball through a fluid (atmosphere) has to be considered
- This effect is achieved just by damping the velocity down, as follows:

```
// We update positions
 vy += gravedad;
 vy *= 0.98; // This is new !
 px += vx;
 py += vy;
```



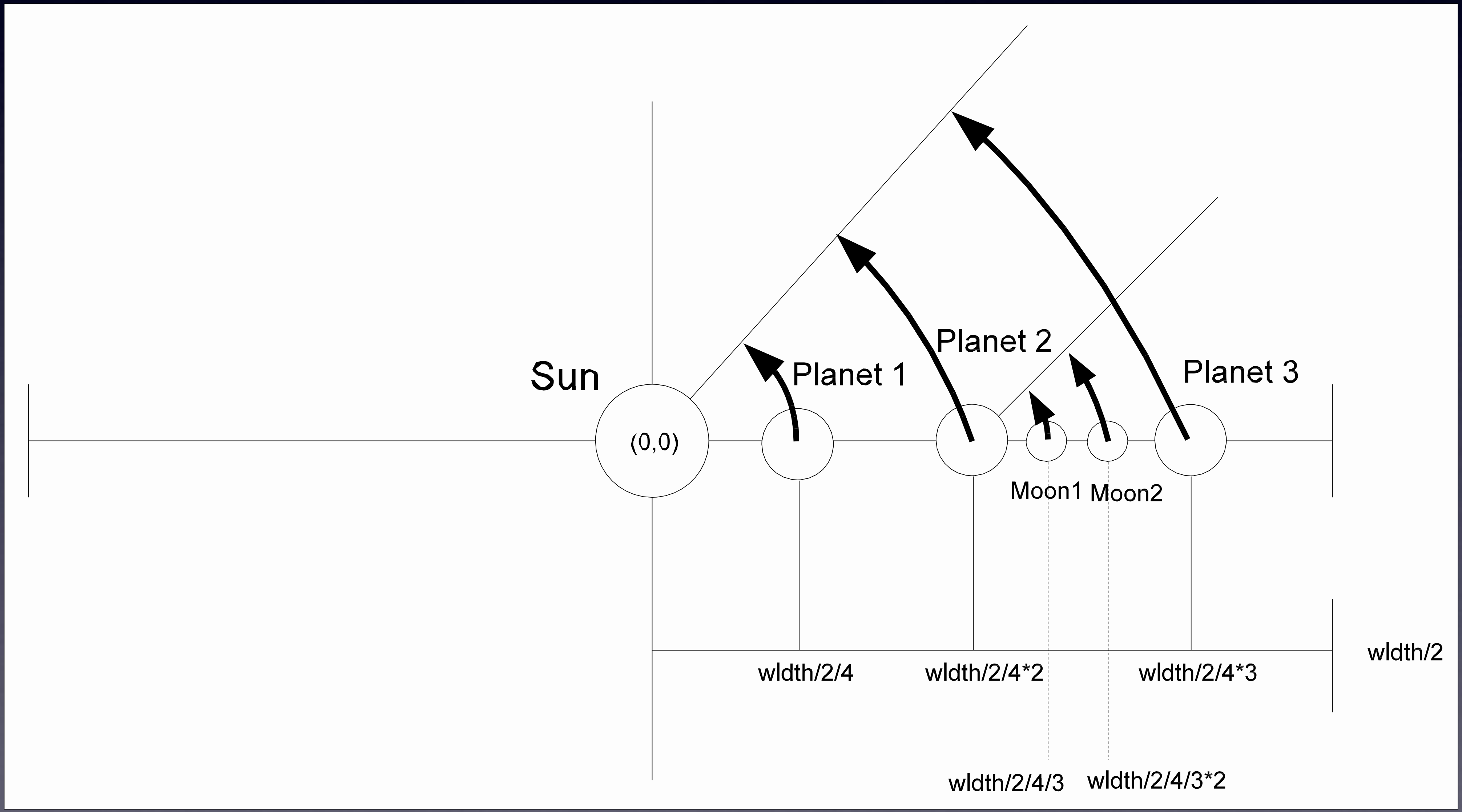
Animation

```
// An ellipse rotating continuously
float angle = 0.0;

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

// Look out !
// draw() initiates model/view
// matrix to the identity
void draw()
{
  fill(0, 20);
  noStroke();
  rect(0, 0, width, height);

  noFill();
  stroke(255);

  // We rotate around the middle
  // of the window, so we have to
  // translate to (0,0).
  // We must take into account
  // the order of the operations
  // (from bottom to top)
  translate(width/2, height/2);
  rotate(angle+=0.1);
  translate(-width/2, -height/2);

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



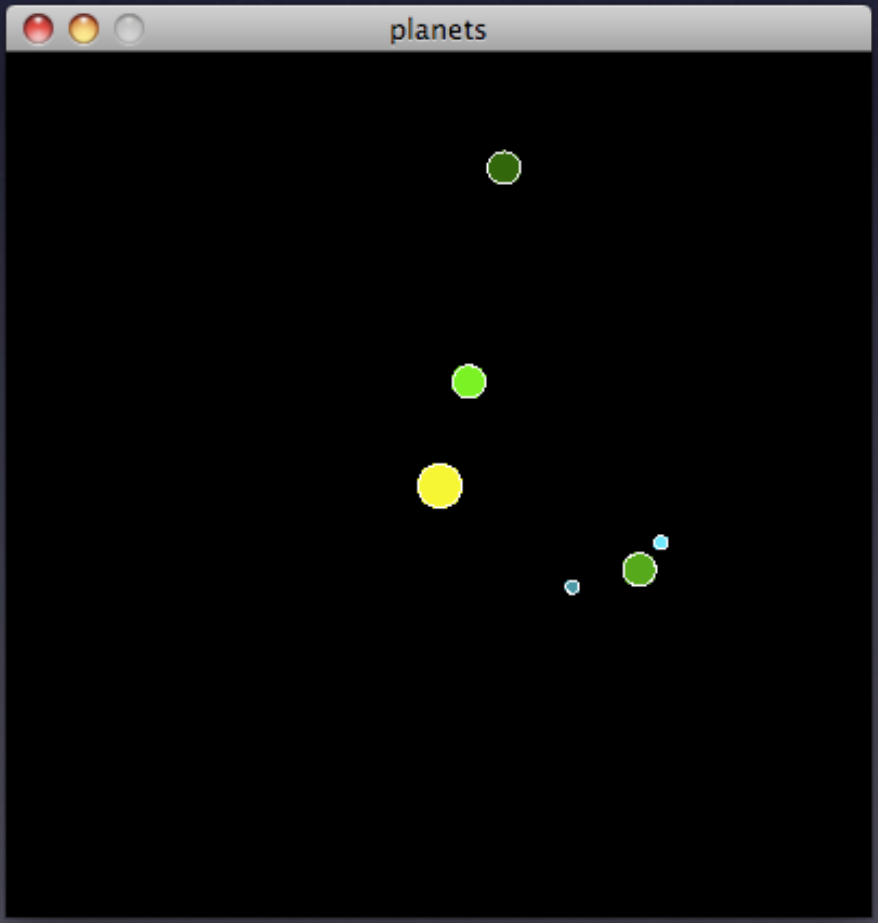
Animation

A solar system:



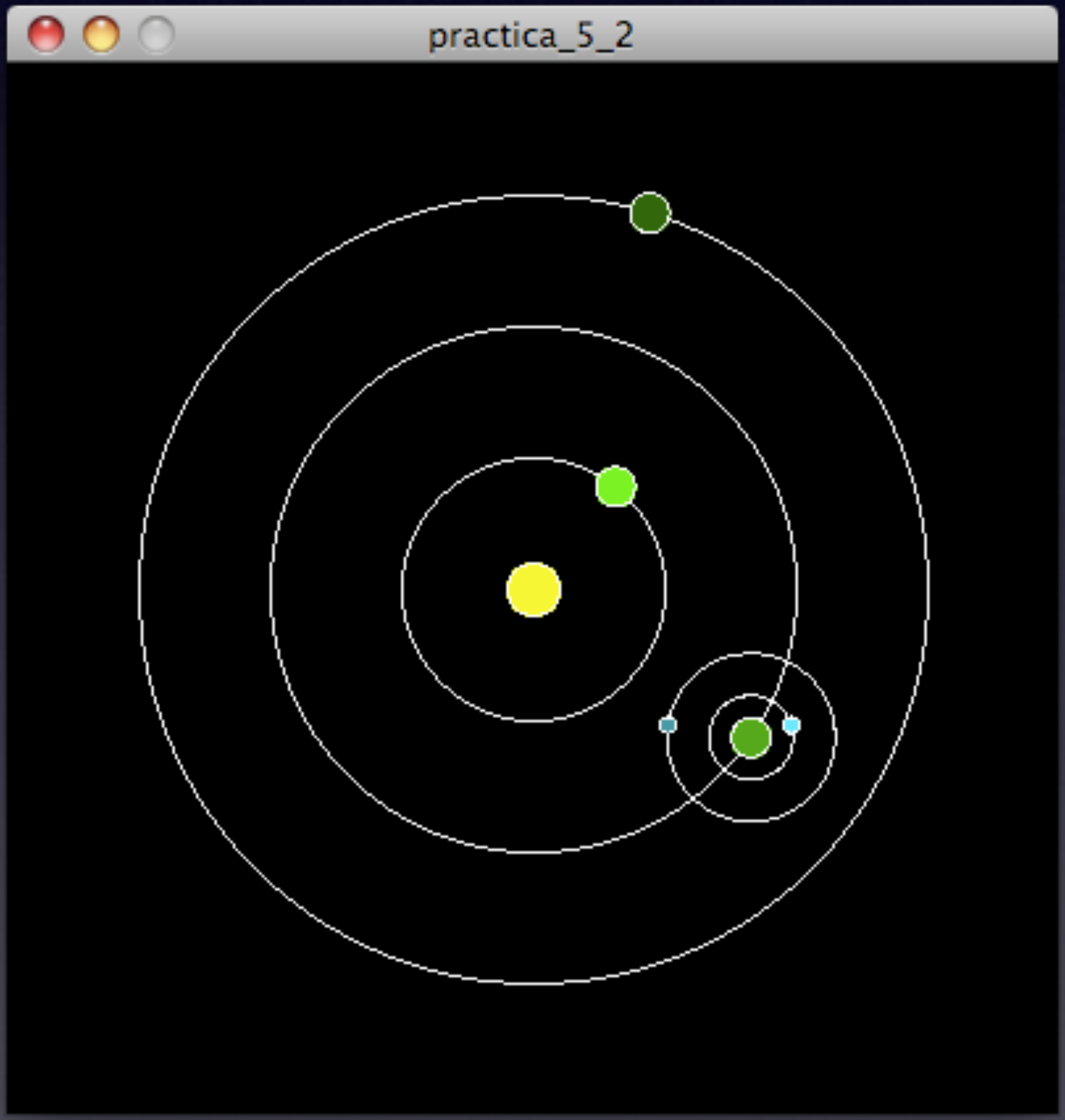
Animation

```
// A planetary system
// in 2D
// A sun, 3 planets and two moons
float angPlanet1 = 0.0,
      angPlanet2 = PI/3.0,
      angPlanet3 = 2.0*PI/3.0,
      angMoon1 = 0.0,
      angMoon2 = PI;

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

void draw()
{
  background(0);

  // We will draw everything centered
  // at (0,0) and will solve
  // their final positions through
  // 2D transformations

  // The sun in the center of
  // our universe
  translate(width/2, height/2);

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

  // Planet 1
  rotate(angPlanet1 += 0.1);
  translate(width/2/4, 0);
  fill(#05FA03);
  ellipse(0, 0, 15, 15);

  // Planet 2
  popMatrix();
  pushMatrix();
  rotate(angPlanet2 += 0.05);
  translate(width/2/4*2, 0);
  fill(#0BA00A);
  ellipse(0, 0, 15, 15);

  // Moon 1
  pushMatrix();
  rotate(angMoon1 += 0.1);
  translate(width/2/4/3, 0);
  fill(#08E4FF);
  ellipse(0, 0, 6, 6);

  // Moon 2
  popMatrix();
  rotate(angMoon2 += 0.05);
  translate(width/2/4/3*2, 0);
  fill(#118998);
  ellipse(0, 0, 6, 6);


  // Planet 3
  popMatrix();
  rotate(angPlanet3 += 0.025);
  translate(width/2/4*3, 0);
  fill(#075806);
  ellipse(0, 0, 15, 15);
}
```



Practice

- Modify the planetary system in order to draw the orbits of the planets and moons:



Practice

- Modify the previous application to allow the sun to orbit as well: