Práctica 3 - Sistema Solar

David Alberto Medina Medina
Dr. Modesto Fernando Castrillon Santana

19 de febrero de 2019

Índice general

Introducción

Processing es un es un IDE *opensource* que utiliza *Java* como lenguaje de programación. Este proyecto está desarrollado y mantenido por la *Processing Foundation* que sirve como soporte de aprendizaje para instruir a estudiantes de todo el mundo en el mundo de la codificación dentro del contexto de las artes visuales.

El objetivo de este proyecto de laboratorio consiste en utilizar las herramientas de dibujo que nos proporciona el entorno de programación de *Processing* para contruir un sistema solar.

Durante el desarrollo de esta experiencia, aprenderemos a utilizar las diferentes herramientas gráficas que dispone *Processing* para la representación 3D de objetos gracias a las librerías *OpenGL* que este integra en su framework. Se hace uso de las primitivas que facilitan las tareas de traslación y rotación del sistema de referencia que ofrece la librería de *Processing*.

Método y materiales

2.1. Materiales

El desarrollo de este proyecto se ha llevado a cabo utilizando el IDE de desarrollo de aplicaciones *Java* de *JetBrains*, *IntelliJ*, y las siguientes herramientas:

- Texturas de los planetas que conforman el sistema solar (Scope, s.f.)
- Fichero de sonido utilizado para la música de fondo (*DL-Sounds*, s.f.)
- Modelo *OBJ*, fichero *MTL* y materiales de la nave espacial (Nieves, s.f.)
- Librerías propias de *Processing*: Sound, Core, ...

2.2. Método

Las siguientes clases que se definenen en este documente se organizan en los siguientes paquetes:

1. camera
2. vehicle
3. galaxy.

2.2.1. Paquete `galaxy`

Clase `Galaxy`

En primera instancia se declaran las propiedades de clase responsables de la renderización del Sol y sus planetas, la nave espacial y el Universo que contiene a los dos anteriores.

```
1     private PShape universe;
2     private Planet sun;
3     private Spaceship spaceship;
```

El método `main()` llama al método estático homónimo de la clase `PApplet` importado de la librería de *Processing*.

```
1     public static void main(String... args) {
2         PApplet.main("galaxy.Galaxy");
3     }
```

El método, `settings()`, es un método propio de *Processing* necesario para realizar las configuraciones iniciales de nuestro programa. En concreto, aquí inicializamos el tamaño de la ventana a 1200×1200 píxeles de resolución y seleccionamos el parámetro de renderización `P3D`.

```
1    public void settings() {
2        size(1200, 1200, P3D);
3    }
```

La función principal de `setup()` consiste en realizar las configuraciones e instanciaciones iniciales de los planetas –y, cuando proceda, sus anillos–, la nave espacial y, el Universo. En este método también se instancian las texturas del Universo, los planetas y anillos.

Como puede observarse, el Sistema Solar está conformado por el Sol y una agrupación de satélites que en conjunto conforman los planetas del Sistema –desde Mercurio hasta Neptuno, incluyendo la Luna.

Además, también se inicializa y reproduce el sonido de fondo de la aplicación instanciando al objeto `SoundFile`.

```
1    public void setup() {
2        background(200);
3
4        noStroke();
5
6        SoundFile soundtrack = new SoundFile(this, "res/sounds/
     Orbit Beat 130.wav");
7
8        universe = createShape(SPHERE, 1000);
9        PImage universe_mat = loadImage("res/planets-texture/2
     k_stars_milky_way.jpg");
10       universe.setTexture(universe_mat);
11
12       int[] labelRGB = new int[]{150, 150, 150};
13
14       PShape spaceship_model = loadShape("res/Falcon t45 Rescue
     ship/Falcon t45 Rescue ship flying.obj");
15       spaceship = new Spaceship(this, spaceship_model, new
     PVector(width/2.f,height/2.f,600.f),
16                1.5f, .2f, labelRGB);
17
18       PImage sun_mat = loadImage("res/planets-texture/download/2
     k_sun.jpg");
19       sun = new Planet(this, "Sun", sun_mat, new PVector(0,0,0),
     110f, .0f,
20                .0f, .0f, .0f, 2.5f, .0f, .0f, labelRGB);
21
22       PImage mercury_mat = loadImage("res/planets-texture/2
     k_mercury.jpg");
23       Planet mercury = new Planet(this, "Mercury", mercury_mat,
     new PVector(-140, 0, 0),
24                9.351f, 140.f, .0f, random(360), .25f,
25                .25f, .01f, random(-45, 45), labelRGB);
26
27       PImage venus_mat = loadImage("res/planets-texture/2
     k_venus_atmosphere.jpg");
28       Planet venus = new Planet(this, "Venus", venus_mat, new
     PVector(-200, 0, 0),
29                15.87f, 200.f, .0f, random(360), .25f,
30                .25f, 177.3f, random(-45, 45), labelRGB);
31
```

```
32          PImage earth_mat = loadImage("res/planets-texture/2
    k_earth_daymap.jpg");
33          Planet earth = new Planet(this, "Earth", earth_mat, new
    PVector(-260, 0, 0),
34                  15.9f, 260.f, .0f, random(360), .25f,
35                  .25f, 23.26f, random(-45, 45), labelRGB);
36
37          PImage moon_mat = loadImage("res/planets-texture/2k_moon.
    jpg");
38          Planet moon = new Planet(this, "Moon", moon_mat, new
    PVector(-40, 0, 0),
39                  6.25f, 40.f, .0f, random(360), .25f,
40                  .25f, 1.5f, random(-45, 45), labelRGB);
41
42          PImage mars_mat = loadImage("res/planets-texture/2k_mars.
    jpg");
43          Planet mars = new Planet(this, "Mars", mars_mat, new
    PVector(-320, 0, 0),
44                  15.487f, 320.f, .0f, random(360), .25f,
45                  .25f, 25.19f, random(-45, 45), labelRGB);
46
47          PImage jupiter_mat = loadImage("res/planets-texture/2
    k_jupiter.jpg");
48          Planet jupiter = new Planet(this, "Jupiter", jupiter_mat,
    new PVector(-380, 0, 0),
49                  30.1f, 380.f, .0f, random(360), .25f,
50                  .25f, 3.13f, random(-45, 45), labelRGB);
51
52          PImage saturn_mat = loadImage("res/planets-texture/2
    k_saturn.jpg");
53          Planet saturn = new Planet(this, "Saturn", saturn_mat, new
    PVector(-440, 0, 0),
54                  25.07f, 440.f, .0f, random(360), .25f,
55                  .25f, 26.73f, random(-45, 45), labelRGB);
56
57          PImage saturn_ring_mat = loadImage("res/planets-texture/2
    k_saturn_ring_alpha.png");
58          Ring saturn_ring = new Ring(this, new PVector(0, 0, 0),
    saturn.getRadius() + saturn.getRadius() * 1.2f,
59                  saturn.getRadius() + saturn.getRadius() * .025f,
    saturn_ring_mat, 200);
60
61          PImage uranus_mat = loadImage("res/planets-texture/2
    k_uranus.jpg");
62          Planet uranus = new Planet(this, "Uranus", uranus_mat, new
    PVector(-500, 0, 0),
63                  10.92f, 500.f, .0f, random(360), .25f,
64                  .25f, 97.77f, random(-45, 45), labelRGB);
65
66          PImage neptune_mat = loadImage("res/planets-texture/2
    k_neptune.jpg");
67          Planet neptune = new Planet(this, "Neptune", neptune_mat,
    new PVector(-560, 0, 0),
68                  10.36f, 560.f, .0f, random(360), .25f,
69                  .25f, 28.32f, random(-45, 45), labelRGB);
70
71          earth.addSatellite(moon);
72
73          saturn.setRing(saturn_ring);
74
75          sun.addSatellite(mercury);
76          sun.addSatellite(venus);
```

```
77          sun.addSatellite(earth);
78          sun.addSatellite(mars);
79          sun.addSatellite(jupiter);
80          sun.addSatellite(saturn);
81          sun.addSatellite(uranus);
82          sun.addSatellite(neptune);
83
84          soundtrack.play();
85      }
```

El método `draw()` es el bucle principal del programa en donde se refresca el estado de la nave espacial, el Universo y los planetas. Como puede observarse, el sistema de referencia se translada al centro de la pantalla.

```
1       public void draw() {
2           spaceship.refresh();
3
4           translate(width/2.f, height/2.f, 0);
5
6           shape(universe);
7
8           sun.refresh();
9       }
```

El usuario puede mover la nave espacial que se genera en el mundo pulsando los botones `A`, `W`, `S` y `D`, y controlar la velocidad de la nave al presionar `+` y , para acelarar o desacelerar la nave –o incluso ir marcha atrás.

```
1       public void keyTyped() {
2           if (key == '+') {
3               spaceship.updateSpeed(spaceship.POWER_UP);
4           } else if (key == '-') {
5               spaceship.updateSpeed(spaceship.POWER_DOWN);
6           } else if (key == 's') {
7               spaceship.updatePitch(spaceship.UP);
8           } else if (key == 'w') {
9               spaceship.updatePitch(spaceship.DOWN);
10          } else if (key == 'a') {
11              spaceship.updateRoll(spaceship.LEFT);
12          } else if (key == 'd') {
13              spaceship.updateRoll(spaceship.RIGHT);
14          }
15      }
```

Clase `Planet`

La renderización de los planetas y satélites del Sistema Solar son gestionados en esta clase. Es por este motivo que son necesarios los siguientes parámetros para instanciar un objeto de la clase `Planet`:

parent Objeto de la clase `PApplet` que nos permitirá llamar a las primitivas de dibujo de *Processing*

name String que identifica al planeta en cuestión. Esta variable será utilizada para el etiquetado del planeta.

material Objeto de la clase `PImage` que contiene las texturas del planeta.

pos Objeto de la clase `PVector` que define la posición inicial del planeta.

radius Número en coma flotante que define el radio del planeta.

orbitRadius Número en coma flotante que define el radio de la órbita de traslación del planeta.

rotationDegree Número en coma flotante que define el ángulo de rotación del planeta.

translationDegree Número en coma flotante que define el ángulo de traslación del planeta en su órbita.

translationSpeed Número en coma flotante que define la velocidad de traslación.

rotationSpeed Número en coma flotante que define la velocidad de rotación del planeta.

axialTilt Número en coma flotante que define el ángulo del eje de rotación del planeta. Este parámetro define los ángulos reales de los ejes de traslación de cada planeta del Sistem Solar (Astronoo, s.f.).

orbitTilt Número en coma flotante que define el ángulo que describe la órbita del planeta con la horizontal.

labelRGB Array de enteros que define los valores RGB del color de la etiqueta del planeta.

```
1     private final float TRANSLATION_SPEED;
2     private final float ROTATION_SPEED;
3     private final float AXIAL_TILT;
4     private final float ORBIT_TILT;
5     private final PImage MATERIAL;
6
7     private final String PLANET_LABEL;
8
9     private PApplet parent;
10    private PShape planet;
11    private Ring ring;
12    private List<Planet> moons;
13    private int[] labelRGB;
14
15    private PVector pos;
16    private float radius;
17    private float orbitRadius;
18    private float rotationDegree;
19    private float translationDegree;
20
21    public Planet(PApplet parent, String name, PImage material,
      PVector pos, float radius, float orbitRadius, float
      rotationDegree,
22                  float translationDegree, float translationSpeed,
      float rotationSpeed, float axialTilt, float orbitTilt, int[]
      labelRGB) {
23        TRANSLATION_SPEED = PApplet.radians(translationSpeed);
24        ROTATION_SPEED = PApplet.radians(rotationSpeed);
25        AXIAL_TILT = PApplet.radians(axialTilt);
26        ORBIT_TILT = PApplet.radians(orbitTilt);
27
28        PLANET_LABEL = name;
```

```
29
30          this.pos = pos;
31          this.radius = radius;
32          this.orbitRadius = orbitRadius;
33          this.MATERIAL = material;
34          this.rotationDegree = PApplet.radians(rotationDegree);
35          this.translationDegree = PApplet.radians(translationDegree)
    ;
36          this.parent = parent;
37
38          moons = new ArrayList<>();
39          planet = parent.createShape(PConstants.SPHERE, radius);
40          planet.setTexture(material);
41          this.labelRGB = labelRGB;
42      }
```

Es posible añadir satélites al planeta –método `addSatellite()`–. Además, un planeta puede devolver la lista de satélites que ya tiene almacenado –método getSatellites()– y el valor de su radio –método getRadius().

```
1       public void addSatellite(Planet planet) {
2           moons.add(planet);
3       }
4
5       public List<Planet> getSatellites() {
6           return moons;
7       }
8
9       public float getRadius() {
10          return radius;
11      }
```

El planeta Saturno se caracteriza por tener un anillo de asteroides que lo rodea. Es por ello, que la clase `Planet` dispone de un método para generar dicho anillo, el método `setRing()`. También se puede comprobar si el planeta anillo –hasRing().

```
1       public void setRing(Ring ring) {
2           this.ring = ring;
3       }
4
5       private boolean hasRing() {
6           if(ring != null)
7               return true;
8           else
9               return false;
```

Un planeta dispone de una etiqueta que lo identifica con su nombre. Con el objetivo de que las etiquetas estén bien alineadas se deshacen las rotaciones de la órbita y radio del planeta que hayan podido tener lugar cuando se llama al método `refresh()` –ver más adelante. A continuación, se instancia un objeto de la clase `BodyLabel` que es responsable del etiquetado de los planetas.

```
1       private void label() {
2           parent.rotateY(-translationDegree);
3           parent.rotateZ(-ORBIT_TILT);
4
5           BodyLabel bl = new BodyLabel(parent, PLANET_LABEL, radius +
     100, 1, labelRGB);
6           bl.refresh();
7       }
```

Una vez todos que los parámtros del planeta han sido correctamente configurados se procede con el proceso de visualización del mismo, el cual tiene lugar dentro del método `refresh()` siguiendo una serie de pasos:

En primer lugar, se guarda el contexto de la matriz de transformación del sistema de coordenadas actual llamando a la primitiva `popMatrix()`. A continuación, se realiza una rotación del sistema de referencia sobre el eje Z e Y para ajustar los ejes de rotación y traslación del planeta.

Una vez concluído este paso, se actualiza el valor del ángulo de traslación –restringiéndose a un máximo de 2π radianes– y se procede a trasladar el eje de coordenadas tantas unidades como indica el parámetro del radio de órbita.

En segundo lugar, se guarda el contexto de la matriz de transformación anterior para poder ajustar el ángulo de rotación y el ángulo del eje de rotación del planeta. Se comprueba si el planeta tiene anillos –si es así se procede con su rederizado–, se procede con el proceso de visualización del planeta y se sale del contexto creado llamando a la primitiva `popMatrix()`.

En tercer lugar, se comprueba si el planeta tiene algún satélite y si es así se abre un contexto nuevo para el mismo llamando a las primitivas `pushMatrix()` y `popMatrix()` y se deshace, ordenadamente, la transformación de rotación del sistema de referencia que tuvo lugar en el primer paso. A continuación, se llama al método `refresh()` del satélite en cuestión.

Por último, se llama al método `label()` para generar la visualización de las etiquetas de los planetas y se sale del contexto –`popMatrix()`– generado al principio del método.

```
1     public void refresh() {
2         parent.pushMatrix();
3
4         parent.rotateZ(ORBIT_TILT);
5         parent.rotateY(translationDegree);
6
7         translationDegree = (translationDegree + TRANSLATION_SPEED)
       % PApplet.radians(360);
8
9
10        parent.translate(orbitRadius, 0, 0);
11
12
13        parent.pushMatrix();
14        parent.rotateY(rotationDegree);
15        rotationDegree = (rotationDegree + ROTATION_SPEED) %
      PApplet.radians(360);
16
17        parent.rotateZ(AXIAL_TILT);
18
19        if(hasRing()) ring.refresh();
20
21        parent.shape(planet);
22
23        if (hasRing()) ring.refresh();
24
25        parent.popMatrix();
26
27
28        if (getSatellites() != null && getSatellites().size() > 0)
      {
29            for (Planet satellite : getSatellites()) {
30                parent.pushMatrix();
```

```
                parent.rotateY(-translationDegree);
                parent.rotateZ(-ORBIT_TILT);
                satellite.refresh();
                parent.popMatrix();
            }
        }
        label();
        parent.popMatrix();
    }
```

Clase `Ring`

Esta clase es la responsable de la visualización de los anillos del planeta Saturno. Como parámetros de inicialización tenemos:

parent Objeto de la clase `PApplet` que nos permitirá llamar a las primitivas de dibujo de *Processing*.

pos Objeto de la clase `PVector` que define la posición inicial del anillo.

outerRadio Número en coma flotante que define el radio externo del anillo.

innerRing Número en coma flotante que define el radio interno del anillo.

material Objeto de la clase `PImage` que contiene las texturas del anillo.

numParts Número entero que define el número de partes de las que está hecha el anillo.

```
1   private final int NUM_PARTS;
2   private final PImage MATERIAL;
3
4   private PShape ring;
5
6   private PApplet parent;
7   private List<PVectorUV> outerRing;
8   private List<PVectorUV> innerRing;
9
10  private PVector pos;
11  private float outerRadius;
12  private float innerRadius;
13
14  public Ring(PApplet parent, PVector pos, float outerRadius,
    float innerRadius, PImage material, int numParts) {
15      this.NUM_PARTS = numParts;
16
17      this.parent = parent;
18      this.pos = pos;
19      this.outerRadius = outerRadius;
20      this.innerRadius = innerRadius;
21      this.MATERIAL = material;
22
23      this.outerRing = new ArrayList<>();
24      this.innerRing = new ArrayList<>();
25
26      renderRing();
27  }
```

El constructor de la clase `Ring` llama al método privado `renderRing()` que es el responsable de generar el anillo.

```
    private void renderRing() {
        float angleStep = PApplet.radians(360.f/NUM_PARTS);

        float angle = .0f;

        for (int i = 0; i <= NUM_PARTS; i++) {
            angle = (i * angleStep) % PApplet.radians(360);
            if ((i % 2) == 0) {
                outerRing.add(buildRingPerimeter(angle, outerRadius
    , true, false));
                innerRing.add(buildRingPerimeter(angle, innerRadius
    , false, false));
            } else {
                outerRing.add(buildRingPerimeter(angle, outerRadius
    , true, true));
                innerRing.add(buildRingPerimeter(angle, innerRadius
    , false, true));
            }
        }
    }
```

Este método llama al método privado `buildRingPerimeter()` que crea los vértices que conforman el contorno interno y externo del anillo. Es en este método, donde se definen además los componentes de los vectores u y v, de modo que se pueda mapear correctamente la imagen con la texturas con una celda que conformará parte del anillo del planeta.

```
    private PVectorUV buildRingPerimeter(float angle, float radius,
     boolean isOuter, boolean isNextPair) {
        float x = radius * PApplet.cos(angle) - radius * PApplet.
    sin(angle);
        float y = 0;
        float z = radius * PApplet.sin(angle) + radius * PApplet.
    cos(angle);
        if(!isNextPair)
            if(isOuter)
                return new PVectorUV(x, y, z, 0, MATERIAL.height);
            else
                return new PVectorUV(x, y, z, MATERIAL.width,
    MATERIAL.height);
        else
            if(isOuter)
                return new PVectorUV(x, y, z, 0, 0);
            else
                return new PVectorUV(x, y, z, MATERIAL.width, 0);
    }
```

Finalmente, la visualización del anillo tiene lugar en el método `refresh()` donde se creará el anillo y se asignará su textura.

```
    public void refresh() {
        ring = parent.createShape();

        ring.beginShape(PConstants.QUAD_STRIP);
        ring.noStroke();
        ring.texture(MATERIAL);

        setUpVertices();

        ring.endShape(PConstants.CLOSE);
        parent.shape(ring);
```

```
12      }
```

Para que la asignación de textura tenga efecto, se selecciona la marca `QUAD_STIP` para que, una vez definidos los vértices de manera sistemática, se generen rectángulos contiguos sobre los cuales se aplicará de manera ordenada las texturas.

```
1       private void setUpVertices() {
2           for (int i = 0; i < NUM_PARTS; i++) {
3               PVectorUV outer1 = outerRing.get(i %NUM_PARTS);
4               PVectorUV inner1 = innerRing.get(i %NUM_PARTS);
5               assignVerticesSet(outer1, inner1);
6
7               PVectorUV outer2 = outerRing.get((i+1) %NUM_PARTS);
8               PVectorUV inner2 = innerRing.get((i+1) %NUM_PARTS);
9               assignVerticesSet(outer2, inner2);
10          }
11      }
12
13      private void assignVerticesSet(PVectorUV outer, PVectorUV inner
        ) {
14          ring.vertex(outer.x, outer.y, outer.z, outer.u, outer.v);
15          ring.vertex(inner.x, inner.y, inner.z, inner.u, inner.v);
16      }
```

2.2.2. Paquete `utils`

Clase `BodyLabel`

Esta clase forma parte del paquete `galaxy.utils` y permite la visualización de las etiquetas de los planetas. Para instaciar un objeto de esta clase necesitaremos pasarle los siguientes parámetros:

parent Objeto de la clase `PApplet` que nos permitirá llamar a las primitivas de dibujo de *Processing*.

label Variable de tipo `String` que contiene el nombre del planeta a etiquetar.

lineLength Número en coma flotante que define la longitud de la línea que apunta hacia el planeta a etiquetar.

lineWeight Numero en coma flotante que define el grosor de la línea que apunta al planeta a etiquetar.

rgb Array de tipo entero que contiene los valores RGB del color de la etiqueta.

```
1       private final String LABEL_TXT;
2
3       private final PApplet parent;
4
5       private PMatrix pos;
6       private float lineLenght;
7       private int lineWeight;
8       private int[] rgb;
9
10      public BodyLabel(PApplet parent, String label, float lineLength
        , int lineWeight, int[] rgb) {
11          this.LABEL_TXT = label;
```

```
12
13          this.parent = parent;
14          this.pos = pos;
15          this.lineLenght = lineLength;
16          this.lineWeight = lineWeight;
17          this.rgb = rgb;
18      }
```

El método `refresh()` muestra por pantalla la etiqueta con el nombre del planeta y la flecha que apunta hacia el mismo.

```
1       public void refresh() {
2           parent.pushMatrix();
3
4           parent.setMatrix(pos);
5           parent.stroke(rgb[0], rgb[1], rgb[2]);
6           parent.strokeWeight(lineWeight);
7           parent.fill(rgb[0], rgb[1], rgb[2]);
8           parent.line(0,0,0,0,-lineLenght,0);
9           parent.text(LABEL_TXT,5,-lineLenght+10,0);
10
11          parent.popMatrix();
12      }
```

CLase `PVectorUV`

Se trata de una clase que extiende de la clase `PVector` que se utiliza para almacenar de manera ordenada las coordenadas de un vértice y las coordenadas UV de su vector de mapeo correspondiente. Esta clase es usada por la clase definida en la sección 2.2.1.

```
1   public class BodyLabel extends PApplet {
2       private final String LABEL_TXT;
3
4       private final PApplet parent;
5
6       private PMatrix pos;
7       private float lineLenght;
8       private int lineWeight;
9       private int[] rgb;
```

2.2.3. Paquete `camera`

Clase `Camera`

Esta clase pertenece al paquete `camera` y gestiona todas las operaciones que se realicen sobre la cámara. Un objeto de la clase `Camera` toma como parámetros de incialización:

parent Objeto de la clase `PApplet` que nos permitirá llamar a las primitivas de dibujo de *Processing*.

pos Objeto de la clase `PVector` que define la posición inicial de la cámara.

center Objeto de la clase `PVector` que define la posición inicial del centro o foco de la cámara.

rotation Objeto de la clase `PVector` que define la rotación inicial de la cámara.

```
    private PApplet parent;
    private PVector pos;
    private PVector center;
    private PVector rotation;

    public Camera(PApplet parent, PVector pos, PVector center,
    PVector rotation) {
        this.parent = parent;
        this.pos = pos;
        this.center = center;
        this.rotation = rotation;
    }

    public Camera(PApplet parent, PVector pos, PVector center) {
        this.parent = parent;
        this.pos = pos;
        this.center = center;
        this.rotation = new PVector(0,1,0);
    }

    public Camera(PApplet parent) {
        this.parent = parent;
        reset();
    }
```

La inicialización o *reset* de los parámetros de la cámara a sus valores por defecto se establecen llamando al método `reset()` (ver (Processing, s.f.)).

```
    public void reset() {
        this.pos = new PVector(parent.width/2.f, parent.height/2.f,
     (parent.height/2.f) / PApplet.tan(PApplet.PI*30.f / 180.f));
        this.center = new PVector(parent.width/2.f, parent.height
    /2.f, .0f);
        this.rotation = new PVector(.0f,1.f,.0f);
    }
```

La posición, centro y rotación de la cámara pueden ser fácilmente modificados llamando a sus correspondientes *setters*.

```
    public void setPosition(float x, float y, float z) {
        pos.x = x;
        pos.y = y;
        pos.z = z;
    }

    public void setCenter(float x, float y, float z) {
        center.x = x;
        center.y = y;
        center.z = z;
    }

    public void setRotation(float x, float y, float z) {
        rotation.x = x;
        rotation.y = y;
        rotation.z = z;
    }
```

El método `refresh()` aplica los cambios en la cámara, llamando a la primitiva `camera()` de *Processing*.

```
1     public void refresh() {
2         parent.camera(pos.x, pos.y, pos.z, center.x, center.y,
      center.z, rotation.x, rotation.y, rotation.z);
3     }
```

2.2.4. Paquete `spaceship`

Clase `Spaceship`

Esta clase es la responsable de la configuración y visualización de la nave espacial. Sus parámetros de inicialización son:

parent Objeto de la clase `PApplet` que nos permitirá llamar a las primitivas de dibujo de *Processing*.

spaceship Objeto de la clase `PShape` que contiene el modelo en 3D –fichero `OBJ--` de la nave espacial.

pos Objeto de la clase `PVector` que define la posición inicial de la nave espacial.

speed Número en coma flotante que define la velocidad con la que se puede mover la nave.

scale Número en com flotante que define el escalado de la nave espacial.

labelRGB Array de tipo entero que contiene los valores RGB del color del texto de inicio de juego.

Otras variables importantes son:

headingX Número en coma flotante que define el ángulo de la nave con respecto al eje X.

headingY Número en coma flotante que define el ángulo de la nave con respecto al eje Y.

headingZ Número en coma flotante que define el ángulo de la nave con respecto al eje Z.

pitch Número en coma flotante que define el ángulo de *cabeceo* acumulado.

roll Número en coma flotante que define el ángulo de *alabeo* acumulado.

```
1     private final static float POS_X_SPAN = 5.f;
2     private final static float POS_Y_SPAN = -50.f;
3     private final static float POS_Z_SPAN = 200.f;
4     private final static float CENTER_DISTACE = -200.f;
5     private final static float MAX_SPEED = 10.f;
6     private final static float SPEED_STEP = .25f;
7     private final static float PITCH_STEP = PApplet.radians(.5f);
8     private final static float ROLL_STEP = PApplet.radians(1.f);
9     private final static float ROTATION_ANIMATION = PApplet.radians
      (200.f);
10
11    public final static float UP = -1.f;
12    public final static float DOWN = 1.f;
13    public final static float LEFT = -1.f;
```

```
14      public final static float RIGHT = 1.f;
15      public final static float POWER_DOWN = -1.f;
16      public final static float POWER_UP = 1.f;
17
18
19      private PApplet parent;
20      private PShape spaceship;
21      private PVector pos;
22      private float speed;
23      private Camera cam;
24
25      private float headingX;
26      private float headingY;
27      private float headingZ;
28
29      private float pitch;
30      private float roll;
31
32      private int intro;
33      private int[] labelRGB;
34
35
36      public Spaceship(PApplet parent, PShape spaceship, PVector pos,
         float speed, float scale, int[] labelRGB) {
37          this.parent = parent;
38          this.spaceship = spaceship;
39          this.pos = pos;
40          this.speed = speed;
41          this.cam = new Camera(parent);
42
43          headingX = PApplet.cos(-90);
44          headingY = PApplet.cos(90);
45          headingZ = PApplet.cos(180);
46
47          pitch = .0f;
48          roll = .0f;
49
50          intro = 0;
51          this.labelRGB = labelRGB;
52
53          this.spaceship.scale(scale);
54          this.spaceship.rotateX(PApplet.radians(180));
55          this.spaceship.rotateY(PApplet.radians(3));
56          setupCameraView();
57      }
```

Al inicializar un objeto de la clase `Spaceship`, se ajusta la posición de la cámara.

```
1       }
2
3       public void refresh() {
4           moveForward();
5
6           cam.setPosition(pos.x + POS_X_SPAN, pos.y + POS_Y_SPAN, pos
    .z + POS_Z_SPAN);
7           cam.setCenter(pos.x, pos.y ,pos.z + CENTER_DISTACE);
8           cam.refresh();
9
10          parent.pushMatrix();
11          parent.translate(pos.x, pos.y, pos.z);
12
13          intro();
```

```
14
15          parent.rotateX(pitch * ROTATION_ANIMATION);
16          parent.rotateZ(roll * ROTATION_ANIMATION);
17          parent.shape(spaceship);
18          parent.popMatrix();
19      }
20
21      private void setupCameraView() {
22          cam.setPosition(pos.x + POS_X_SPAN, pos.y + POS_Y_SPAN, pos
    .z + POS_Z_SPAN);
23          cam.setCenter(pos.x, pos.y ,pos.z + CENTER_DISTACE);
24          cam.setRotation(.0f, 1.f, 0.f);
25          cam.refresh();
26      }
```

La velocidad y, los ángulos de cabeceo y/o alabeo son actualizados cada vez que el usuario interactúa con la aplicación a través del teclado, llamándos a los métodos `updateSpeed()`, `updatePitch()` y `updateRoll()`, respectivamente.

Cada vez que se actualiza el ángulo de cabeceo, se suma una cantidad discreta de ángulo definida en una constante de clase –`PITCH_STEP` y `ROLL_STEP`– en las variables `pitch` y `roll`. Este ángulo de cabeceo y alabeo serán utilizados para definir la nueva ruta de movimiento de la nave espacial con respecto al eje Y –*pitch* o ángulo cabeceo– y eje X –`roll` o ángulo de alabeo.

```
1       public void updateSpeed(float speedDir) {
2           this.speed += speedDir * SPEED_STEP %MAX_SPEED;
3       }
4
5       public void updatePitch(float pitchDir) {
6           pitch += pitchDir * PITCH_STEP %PApplet.radians(360);
7           headingY += pitchDir * PApplet.cos(pitch);
8       }
9
10      public void updateRoll(float rollDir) {
11          roll += rollDir * ROLL_STEP %PApplet.radians(360);
12          headingX += rollDir * PApplet.cos(roll);
13      }
```

El movimiento de la nave está definido por sus coordenadas polares, tal que

$$\left.\begin{aligned} x &= speed \times \cos_x \alpha \\ y &= speed \times \cos_y \beta \\ z &= speed \times \cos_z \gamma \end{aligned}\right\}$$

donde $\cos_x \alpha, \cos_y \beta$ y $\cos_z \gamma$ corresponde al *heading* con respecto al eje X, Y y Z, respectivamente.

```
1       private void moveForward() {
2           pos.x += speed * headingX;
3           pos.y += speed * headingY;
4           pos.z += speed * headingZ;
5       }
```

Existe también una función para mostrar un pequeño tutorial que indica al usuario los controles necesarios para controlar la nave.

```
1       private void intro() {
2           parent.stroke(labelRGB[0], labelRGB[1], labelRGB[2]);
3           if(intro < 40) parent.text("PULSE A, W, S o D para moverse"
    , -150,-150,-200);
```

```
4         else if (intro > 40 && intro < 80) parent.text("PULSE + o -
      para acelerar y desacelerar",-150,-150,-200);
5
6         if (intro < 80) intro++;
7     }
```

La cámara se actualiza junto con la posición de la nave espacial a lo largo de su viaje por el espacio. Los movimientos de cabeceo y alabeo generan una rotación de la nave que puede ser fácilmente apreciable por pantalla.

```
1     public void refresh() {
2         moveForward();
3
4         cam.setPosition(pos.x + POS_X_SPAN, pos.y + POS_Y_SPAN, pos
      .z + POS_Z_SPAN);
5         cam.setCenter(pos.x, pos.y ,pos.z + CENTER_DISTACE);
6         cam.refresh();
7
8         parent.pushMatrix();
9         parent.translate(pos.x, pos.y, pos.z);
10
11        intro();
12
13        parent.rotateX(pitch * ROTATION_ANIMATION);
14        parent.rotateZ(roll * ROTATION_ANIMATION);
15        parent.shape(spaceship);
16        parent.popMatrix();
17    }
```

Resultados y discusión

Los materiales software y métodos aquí descritos han sido favorables para la realización y culminación del proyecto. Se ha conseguido diseñar y visualizar un sistema planetario ((Medina, s.f.)). Las traslaciones y rotaciones de los planetas se han implementado con relativa sencillez gracias a las primitivas que dispone *Processing* para manipular el sistema de coordenadas.

Se ha logrado trasladar y establecer un método de seguimiento de la nave espacial a lo largo de su trayectoria por el espacio. Ha sido posible migrar del IDE oficial de *Processing* a *IntelliJ*.

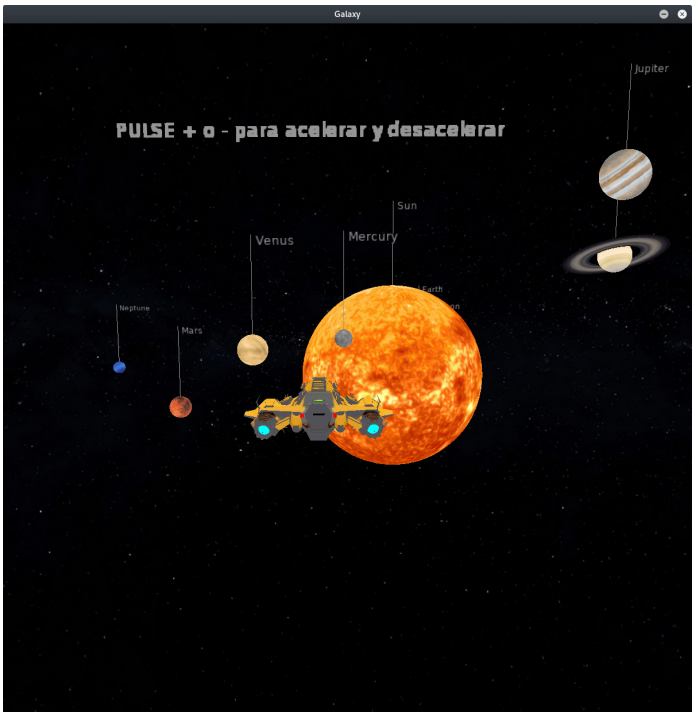

Figura 3.1: Nave espacial viajando por el Sistema Solar

Conclusiones

El conocimiento de las coordenadas polares ((?, ?) ha sido de real importancia a la hora de definir un método para establecer el movimiento de la nave.

Las primitivas de *Processing* para la manipulación del sistema de referencia han facilitado en gran medida el proceso de generación y movimiento de los planetas y satélites del Sistema Solar. Los métodos de *Processing* de gestión de la pila de las matrices de transdormación –`pushMatrix()` y `popMatrix()`– han sido de gran importancia para establecer contextos sobre los cuales se definen de manera aislada transformaciones lineales sobre el sistema de referencia.

Referencias

Astronoo. (s.f.). *Inclination of the Planets.* http://www.astronoo.com/en/articles/axial-tilt-planets.html. Autor. (Accessed: 2019-02-19)

DL-Sounds. (s.f.). https://www.dl-sounds.com/royalty-free/orbit-beat-130/. (Accessed: 2019-02-19)

Medina, D. A. M. (s.f.). *Github: Solar System.* https://github.com/david00medina/solar-system. Github. (Accessed: 2019-02-19)

Nieves, H. (s.f.). *Falcon t45 Rescue ship.* https://www.turbosquid.com. Turbo Squid. (Accessed: 2019-02-19)

Processing. (s.f.). *Processing: Reference.* https://processing.org/reference/. Autor. (Accessed: 2019-02-19)

Scope, S. S. (s.f.). *Solar Textures.* https://www.solarsystemscope.com/textures/. Autor. (Accessed: 2019-02-19)