Práctica 5 - Escena

David Alberto Medina Medina
Dr. Modesto Fernando Castrillon Santana

12 de marzo 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 una escena tridimensional.

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*.

Utilizaremos las primitivas `camera()` y `perspective()` que dispone *Processing* para manipular la vista y perspectiva del mundo que generaremos.

Además, aprenderemos a utilizar las primitivas que ofrece *Processing* para la gestión de la iluminación y materiales de los objetos tridimensionales.

El objetivo de esta experiencia consistirá en crear una escena de una habitación donde podamos utilizar las distintas herramientas de gestión de la iluminación y cámara que ofrece *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 la mesa y pared.
- Modelo *OBJ* de un candelabro nave espacial.
- Librerías propias de *Processing*.

2.2. Método

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

1. camera
2. light
3. object
4. player
5. scene
6. utils

2.2.1. Paquete `scene`

Clase `Scene`

En esta clase se define el método principal que heredará de la clase `PApplet` de *Processing* con el objeto de poder acceder a todas las primitivas de la librería. El método principal debe llamar al método `PApplet.main()` para poder empezar a utilizar *Processing*.

```
1    public static void main(String[] args) {
2        PApplet.main("scene.Scene");
3    }
```

Se definen el tamaño de la ventana y se selecciona el *rederer* `P3D`.

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

Se inicializan la cámara, la iluminación, el jugador y los objetos que forman parte de la escena.

```
    public void setup() {
        light = new Light(this);
        Camera cam = new Camera(this);

        PVector pos = new PVector(width / 2.f, height / 2.f - 500.f
    , 1050.f);
        PVector center = new PVector(width / 2.f, height / 2.f -
    400.f, -200.f);
        PVector rotation = new PVector(.0f, .0f, .0f);
        player = new Player(this, pos, center, cam, 5.5f,1.f);

        spawnObjects();
        placeMouseCenter();
        noCursor();
    }
```

El método `placeMouseCenter()` coloca el ratón en el centro de la ventana y se oculta llamando a la primitiva `noCursor()`.

```
    private void placeMouseCenter() {
        try {
            Robot robot = new Robot();
            robot.mouseMove(displayWidth / 2, displayHeight / 2 +
    59);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

El método `spawnObjects()` es el responsable de renderizar todos los objetos del mundo y cargar las texturas y materiales de cada uno de ellos según corresponda. Este método llama a su vez a: `renderFloor()`, `renderStar()`, `renderCandle()`, `renderTable()` y `renderWall()`.

```
    private void spawnObjects() {
        renderFloor();
        renderStar();
        renderCandle();
        renderTable();
        renderWall();
    }

    private void renderWall() {
        PShape wallShape = createShape(PConstants.BOX, 1200.f, 950.
    f, 75.f);
        PImage img = loadImage("res/models/wall/Textures/
    VC_Summer__2983.jpg");
        PVector ambient = new PVector(142.f, 43.f, 2.f);
        PVector emissive = new PVector(153.f, 87.f, 76.f);
        PVector specular = new PVector(235.f, 155.f, 130.f);
        float shininess = 50.f;
        Material mat = new Material(this, ambient, emissive,
    specular, shininess);
```

```
17          wall = new SceneObject(this, new PVector(width / 2.f,
    height / 4.f - 150.f, -640.f), wallShape, new Texture(img),
    null);
18      }
19
20      private void renderTable() {
21          PShape tableShape = loadShape("res/models/table/tbl022.obj"
    );
22          tableShape.rotateX(radians(90));
23          tableShape.scale(.45f);
24          table = new SceneObject(this, new PVector(width / 2.f -
    200.f, height / 4.f - 20.f, -50.f), tableShape, null, null);
25      }
26
27      private void renderCandle() {
28          PShape candleShape = loadShape("res/models/candle/
    candles_obj.obj");
29          candleShape.rotateX(radians(180));
30          candleShape.scale(.5f);
31          PVector ambient = new PVector(0, 0, 0);
32          PVector emissive = new PVector(10, 10, 10);
33          PVector specular = new PVector(75, 75, 75);
34          float shininess = 15.f;
35          Material mat = new Material(this, ambient, emissive,
    specular, shininess);
36          candle = new SceneObject(this, new PVector(width / 2.f -
    200.f, height / 4.f - 200.f, -50.f), candleShape, null, null);
37      }
38
39      private void renderStar() {
40          PShape starShape = createShape(PConstants.ELLIPSE, .0f, .0f
    , 500.f, 500.f);
41          PImage img = loadImage("res/textures/2k_sun.jpg");
42          PVector ambient = new PVector(246.f, 246.f, 70.f);
43          PVector emissive = new PVector(255.f, 255.f, 212.f);
44          PVector specular = new PVector(255.f, 255.f, 212.f);
45          float shininess = 255.f;
46          Material mat = new Material(this, ambient, emissive,
    specular, shininess);
47          star = new SceneObject(this, new PVector(3 * width / 4.f, -
    height / 6.f, -1000.f), starShape, new Texture(img), mat);
48      }
49
50      private void renderFloor() {
51          PShape floorShape = createShape(PConstants.BOX, 2400.f, 90,
     2400.f);
52
53          PImage img = loadImage("res/textures/floor.jpg");
54          PVector ambient = new PVector(67.f, 59.f, 59.f);
55          PVector emissive = new PVector(119.f, 112.f, 112.f);
56          PVector specular = new PVector(174.f, 166.f, 166.f);
57          float shininess = 128.f;
58          Material mat = new Material(this, ambient, emissive,
    specular, shininess);
59          floor = new SceneObject(this, new PVector(0, 0, 0),
    floorShape, new Texture(img), mat);
60      }
```

Una vez realizados los ajustes previos, se carga en la ventana todos los componentes de los objetos de la escena haciendo uso de las primitivas `pushMatrix()` y `popMatrix()`. En cada iteración del método `draw()` se refresca el estado del jugador y se ajustan los parámetros de iluminación.

```
    public void draw() {
        background(128);

        placeMouseCenter();

        player.refresh();

        lightSetting();

        pushMatrix();
        starMove = dir * 1;
        star.getPos().x = star.getPos().x + starMove;
        translate(star.getPos().x, star.getPos().y, star.getPos().z
    );
        if (star.getPos().x < -300) dir = -dir;
        else if (star.getPos().x > 1200) dir = -dir;
        star.refresh();
        popMatrix();

        pushMatrix();
        translate(candle.getPos().x, candle.getPos().y, candle.
    getPos().z);
        candle.refresh();
        popMatrix();

        pushMatrix();
        translate(table.getPos().x, table.getPos().y, table.getPos
    ().z);
        table.refresh();
        popMatrix();

        pushMatrix();
        translate(wall.getPos().x, wall.getPos().y, wall.getPos().z
    );
        wall.refresh();
        popMatrix();

        pushMatrix();
        translate(width / 2.f, height / 2.f, .0f);
        floor.refresh();
        popMatrix();
    }
```

Los parámetros de iluminación son gestionados en el método `lightSetting()` donde se podrá encender o apagar el iluminado haciendo clic derecho en el ratón. Cuando la iluminación genérica no está habilitada, se establecen los parámetros de iluminación básicos de la escena: ambiente, direccional, especular y punto-luz (*point-light*).

```
    private void lightSetting() {
        if (mousePressed) light.switchOn();
        else light.switchOff();
        if (!light.isOn()) background(70.f);

        PVector ambient = star.getMaterial().getAmbient();
        light.setAmbientLight(ambient.x, ambient.y, ambient.z, star
    .getPos().x, star.getPos().y, star.getPos().z);
        ambientLight(255,255,255, candle.getPos().x, candle.getPos
    ().y, candle.getPos().z);
        directionalLight(255,255,255, -star.getPos().x, star.getPos
    ().y, -star.getPos().z);
```

```
10          light.setLightSpecular(255,255,255);
11          pointLight(255,0,0,player.getPos().x, player.getPos().y,
    player.getPos().z);
12          light.refresh();
13      }
```

2.2.2. Paquete `light`

Clase `Light`

Esta clase utiliza tres vectores que serán los responsables de la gestión de los parámetros de iluminación. Estos vectores controlan la luz ambiental, direccional y especular. El constructor inicializa estos vectores a sus valores por defecto.

```
1       private PApplet parent;
2       private PVector[] ambientLight;
3       private PVector[] directionalLight;
4       private PVector lightSpecular;
5       private boolean isOn;
6
7       public Light(PApplet parent) {
8           this.parent = parent;
9
10          ambientLight = new PVector[]{
11                  new PVector(.0f, .0f, .0f),
12                  new PVector(.0f, .0f, .0f)
13          };
14          directionalLight = new PVector[]{
15                  new PVector(.0f, .0f, .0f),
16                  new PVector(.0f, .0f, 1.f)
17          };
18          lightSpecular = new PVector(255.f, 255.f, 255.f);
19      }
```

La luz ambiental, especular `setters`.

```
1       public void setAmbientLight(float r, float g, float b) {
2           ambientLight[0].x = r;
3           ambientLight[0].y = g;
4           ambientLight[0].z = b;
5       }
6
7       public void setAmbientLight(float r, float g, float b, float x,
         float y, float z) {
8           setAmbientLight(r, g, b);
9
10          ambientLight[1].x = x;
11          ambientLight[1].y = y;
12          ambientLight[1].z = z;
13      }
14
15      public void setDirectionalLight(float r, float g, float b,
    float x, float y, float z) {
16          directionalLight[0].x = r;
17          directionalLight[0].y = g;
18          directionalLight[0].z = b;
19
20          directionalLight[1].x = x;
21          directionalLight[1].y = y;
22          directionalLight[1].z = z;
23      }
```

```
24
25      public void setLightSpecular(float r, float g, float b) {
26          lightSpecular.x = r;
27          lightSpecular.y = g;
28          lightSpecular.z = b;
29      }
```

Es posible invocar a las primitivas `lights()` y `noLights()` llamando a los métodos `switchOn()` y `switchOff()`, respectivamente. Sendos métodos modificarán un testigo para indicar si se está trabajando con la iluminación por defecto o no. Se puede acceder a este testigo a través del método `isOn()`.

```
1       public boolean isOn() {
2           return isOn;
3       }
4
5       public void switchOn() {
6           parent.lights();
7           isOn = true;
8       }
9
10      public void switchOff() {
11          parent.noLights();
12          isOn = false;
13      }
```

Los parámetros de iluminación se llevan a cabo cada vez que se invoca al método `refresh()`.

```
1       public void refresh() {
2           parent.ambientLight(ambientLight[0].x, ambientLight[0].y,
        ambientLight[0].z,
3                   ambientLight[1].x, ambientLight[1].y, ambientLight
        [1].z);
4
5           parent.directionalLight(directionalLight[0].x,
        directionalLight[0].y, directionalLight[0].z,
6                   directionalLight[1].x, directionalLight[1].y,
        directionalLight[1].z);
7
8           parent.lightSpecular(lightSpecular.x, lightSpecular.y,
        lightSpecular.z);
9       }
```

2.2.3. Paquete `object`

Clase `SceneObject`

Esta clase tiene un constructor que toma como parámetros de entrada el modelo del objeto (`PShape`), las texturas (`Texture`) y el material (`Material`).

```
1       private PApplet parent;
2
3       private PVector pos;
4       private PShape model;
5       private Texture texture;
6       private Material material;
7
8       public SceneObject(PApplet parent, PVector pos, PShape model,
        Texture texture, Material material) {
```

```
        this.parent = parent;

        this.pos = pos;
        this.model = model;
        this.texture = texture;
        this.material = material;
    }

    public SceneObject(PApplet parent, PVector pos, PShape model,
    Material material) {
        this.parent = parent;
        this.pos = pos;
        this.model = model;
        this.material = material;
    }
```

Se puede acceder a estas propiedades a través de sus `getters`.

```
    public PVector getPos() {
        return pos;
    }

    public Material getMaterial() {
        return material;
    }

    public void setPos(PVector pos) {
        this.pos = pos;
    }

    public void setModel(PShape model) {
        this.model = model;
    }

    public PShape getModel() {
        return model;
    }
```

Se puede actualizar el estado de un objeto `SceneObject` llamando al método `refresh()`. El material, textura y posición de una instancia de esta clase son actualizados con cada iteración de `draw()`.

```
    private void refreshPosition() {
        parent.pushMatrix();
        parent.translate(pos.x, pos.y, pos.z);
        parent.shape(model);
        parent.popMatrix();
    }
```

Clase `Texture`

Esta clase es la responsable de almacenar las texturas de una instancia del objeto `SceneObject`.

```
    private PImage texture;

    public Texture(PImage texture) {
        this.texture = texture;
    }

    public void setTexture(PImage texture) {
```

```
        this.texture = texture;
    }

    public PImage getTexture() {
        return texture;
    }
```

Clase `Material`

Esta clase es la responsable de gestionar los parámetros de los materiales que son utilizados por los objetos de la escena. Estos parámetros controlan cómo se comporta cada objeto de la escena ante la luz. El método `refresh()` es el responsable de llamar a las primitivas en cada iteración.

```
        this.parent = parent;
        this.ambient = ambient;
        this.emissive = emissive;
        this.specular = specular;
        this.shininess = shininess;
    }

    public void refresh() {
        parent.pushStyle();

        parent.noStroke();
        parent.ambient(ambient.x, ambient.y, ambient.z);
        parent.emissive(emissive.x, emissive.y, emissive.z);
        parent.specular(specular.x, specular.y, specular.z);
        parent.shininess(shininess);

        parent.popStyle();
    }

    public void setAmbient(PVector ambient) {
        this.ambient = ambient;
    }

    public void setEmissive(PVector emissive) {
        this.emissive = emissive;
    }

    public void setSpecular(PVector specular) {
        this.specular = specular;
    }

    public void setShininess(float shininess) {
        this.shininess = shininess;
    }

    public PVector getAmbient() {
        return ambient;
    }

    public PVector getEmissive() {
        return emissive;
    }

    public PVector getSpecular() {
        return specular;
    }

```

```
48      public float getShininess() {
49          return shininess;
50      }
```

2.2.4. Paquete `utils`

CLase `PVector4D`

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.

Este clase también será utilizada por `Camara` para definir los atributos de la primitiva `perspective()` realizando una llamada al segundo constructor que se ha creado explícitamente para generar un vector 4D.

```
1       public float t;
2
3       public PVector4D(float x, float y, float z, float t) {
4           super(x, y, z);
5           this.t = t;
6       }
7   }
```

2.2.5. 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.

perspective Objeto de la clase `PVectorUV` que define los parámetros de la primitiva `perspective()` con el objeto de definir la perspectiva de la cámara.

```
1       public final float CAMERAZ;
2
3       private PApplet parent;
4       private PVector pos;
5       private PVector center;
6       private PVector rotation;
7       private PVector4D perspective;
8
9       public Camera(PApplet parent, PVector pos, PVector center,
        PVector rotation, PVector4D perspective) {
```

```
10          this.parent = parent;
11          this.pos = pos;
12          this.center = center;
13          this.rotation = rotation;
14          this.perspective = perspective;
15
16          CAMERAZ = (parent.height / 2.f) / PApplet.tan(PApplet.PI *
    60.f / 360.f);
17      }
18
19      public Camera(PApplet parent, PVector pos, PVector center,
    PVector4D perspective) {
20          this.parent = parent;
21          this.pos = pos;
22          this.center = center;
23          this.rotation = new PVector(0, 1, 0);
24          this.perspective = perspective;
25
26          CAMERAZ = (parent.height / 2.f) / PApplet.tan(PApplet.PI *
    60.f / 360.f);
27      }
28
29      public Camera(PApplet parent) {
30          this.parent = parent;
31          CAMERAZ = (parent.height / 2.f) / PApplet.tan(PApplet.PI *
    60.f / 360.f);
32          reset();
33      }
```

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.)).

```
1       public void reset() {
2           this.pos = new PVector(parent.width / 2.f,
3                   parent.height / 2.f,
4                   (parent.height / 2.f) / PApplet.tan(PApplet.PI *
    30.f / 180.f));
5
6           this.center = new PVector(parent.width / 2.f,
7                   parent.height / 2.f,
8                   .0f);
9
10          this.rotation = new PVector(.0f, 1.f, .0f);
11
12          this.perspective = new PVector4D(PApplet.PI / 3.f,
13                  (float) parent.width / parent.height,
14                  CAMERAZ / 10.f, CAMERAZ * 10.f);
15      }
```

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

```
1       public void setPosition(float x, float y, float z) {
2           pos = new PVector(x, y, z);
3       }
4
5       public void setCenter(float x, float y, float z) {
6           center = new PVector(x, y, z);
7       }
8
9       public void setRotation(float x, float y, float z) {
10          rotation = new PVector(x, y, z);
```

```
11      }
12
13      public void setPerspective(float fovy, int aspect, float zNear,
         float zFar) {
14          perspective = new PVector4D(fovy, aspect, zNear, zFar);
15      }
```

El método `refresh()` aplica los cambios en la cámara, llamando a la primitiva `camera()` y `perspective()` 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           parent.perspective(perspective.x, perspective.y,
        perspective.z, perspective.t);
4       }
```

2.2.6. Paquete `player`

Clase `Player`

El jugador de la escena se implementa con una cámara en primera persona por lo que la posición del jugador y de la cámara serán la misma. Los parámetros de entrada más destacados del constructor son: el vector de posición del jugador y el vector de foco o centro (este mismo parámetro será utilizado por `Camara`).

El método `setInitialCenter()` ajusta el vector de foco de la cámara utilizando el sistema de coordenadas polares.

```
1           this.STEP = step;
2           this.SPEED = speed;
3
4           this.parent = parent;
5           this.pos = pos;
6           this.center = center;
7           this.cam = cam;
8
9           setInitialCenter();
10      }
11
12      private void setInitialCenter() {
13          radius = PApplet.sqrt(center.x * center.x + center.y *
        center.y + center.z * center.z);
14          zenith = PApplet.acos(center.z / radius);
15          azimuth = PApplet.atan(center.y / center.x);
16      }
```

El movimiento puede es gestionado por la clase `moveTo()` permitiéndo mover el jugador en la escena al pulsar las teclas `W, A, S` y `D`. Dependiendo de qué tecla se pulse se llamará al método responsable de mover el jugador en la dirección deseada.

```
1       public void setModel(PShape model) {
2           this.model = model;
3       }
4
5       public PShape getModel() {
6           return model;
7       }
8
```

```
9       public void refresh() {
10          if (material != null) material.refresh();
11          if (texture != null) model.setTexture(texture.getTexture())
        ;
12          refreshPosition();
13      }
14
15      private void refreshPosition() {
16          parent.pushMatrix();
17          parent.translate(pos.x, pos.y, pos.z);
18          parent.shape(model);
19          parent.popMatrix();
20      }
21  }
```

El método `refresh()` actualiza la posición y foco del jugador en el mundo.

Conclusiones

En esta experiencia hemos aprendido a manipular la vista del mundo con las primitivas `camara()` y `perspective()` de *Processing*. Hemos aprendido a manipular los parámetros de iluminación y materiales con el objeto de obtener escenas con un iluminado personalizado.

Se ha intentado ver cómo afecta a la iluminación de los objetos la luz ambiental. Este parámetro cambia su posición a lo largo del tiempo, siguiendo la posición de la pequeña esfera del escenario. Se puede observar este cambio influye en la iluminación de los objetos de la escena.

Referencias

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

Wikipedia. (s.f.). *Wikipedia: Tridimensional.* `https://es.wikipedia.org/wiki/Tridimensional`. Autor. (Accessed: 2019-02-26)