Práctica 6 y 7 - Procesamiento de imagen y video, y síntesis de sonido

David Alberto Medina Medina
Dr. Modesto Fernando Castrillon Santana

6 de abril 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 esta experiencia consite en desarrollar una aplicación de realidad aumentada desde la cual se puede interactuar con objetos virtuales en un espacio tridimensional y generar audio a partir de ellos. El modelo de una guitarra eléctrica es elegido en un intento de ilustrar esta funcionalidad.

En la escena se muestra la estructura de un esqueleto humano resultado de un algoritmo de seguimiento del cuerpo que procesa la *Kinect v1.8* obteniendo como resultado la información de los puntos de articulación del mismo (DART, s.f.).

La *Kinect* solo ofrece la posición de cada punto de articulación en el plano XY por lo que es necesario analizar la información que de los sensores IR si queremos conocer la profundidad de cada uno de los puntos de articulación que conforman el esqueleto.

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:

- Librería *Processing* (Processing, s.f.).
- Librería *Kinect4WinSDK* (Chung, s.f.).
- Librería *Sound* (Foundation, s.f.).
- Modelo *OBJ* de una guitarra eléctrica (stone777, s.f.).

2.2. Método

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

1. main
2. kinect
3. soundFX
4. object
5. algorithms

2.2.1. Paquete `main`

Clase `KinectMe`

Para poder utilizar las primitivas de *Processing* esta clase debe heredar de `PApplet`. Para iniciar una aplicación de *Processing*, el método estático `main()` debe llamar a la primitiva `PApplet.main()` para indicar el nombre de la clase principal desde la cual se llama y sobrescriben los métodos primitivos: `settings()`, `setup()` y `draw()`.

```
1            PApplet.main("main.KinectMe");
2        }
3    }
```

El método primitivo `settings()` establece el tamaño de la pantalla llamando al método `size()` y pasándole la marca del renderizador de gráficos 3D (`P3D`).

```
    @Override
    public void settings() {
        super.settings();
        size(640, 480, P3D);
    }
```

El método `setup()` inicializa un objeto de la clase `Kinect` que se utiliza como controlador de la misma. Además, se llama al método `spawnGuitar()` responsable del renderizado y puesta en escena de una guitarra que será el objeto tridimensional por medio del cual el usuario puede interactuar. El método `createFloor()` permite visualizar una rejilla que será utilizada como suelo de la escena.

```
    @Override
    public void setup() {
        super.setup();
        smooth();
        stroke(255);

        kinect = new Kinect(this, null, null, null);

        spawnGuitar();

        createFloor();
    }
```

El método privado `spawnGuitar()` carga el modelo 3D de la guitarra y le asigna una posición en la escena. El método `doDrawInteractionArea()` permite visualizar los volúmenes de interacción asociados a la guitarra si ajustamos el parámetro `DEBUG_AREAS` a `true`.

A continuación, se llama al método privado `addGuitarInteraction()` para generar los ortoedros de interacción a partir de los cuales el usuario puede interacturar con la guitarra. Se generan dos volúmenes de interacción: uno para el cuello y el otro para las cuerdas de la guitarra. Es posible visualizar los vértices de estos ortoedros si se establece el parámetro de configuración `DEBUG_VERTICES` a `true`.

```
    private void spawnGuitar() {
        PShape guitarModel = loadShape("../../data/models/guitar/
    guitar.obj");
        guitar = new Guitar(this, guitarModel, null, null);

        guitar.setPos(new PVector(width / 2.f, 4.f * height / 6.f,
    150));
        guitar.setRotation(new PVector(radians(0), radians(0),
    radians(-140)));
        guitar.scale(55.f);

        addGuitarInteraction("NECK", -25, -77, 4,
                radians(0), radians(0), radians(5),
                50, 4, 4);

        addGuitarInteraction("STRINGS", 25, -30, 4,
                radians(0), radians(0), radians(5),
                20, 4, 6);

```

```
17          guitar.doDrawInteractionArea(DEBUG_AREAS);
18      }
19
20      private void addGuitarInteraction(String id, float xOffset,
        float yOffset, float zOffset,
21                                        float xRotationOffset, float
        yRotationOffset, float zRotationOffset,
22                                        float width, float height,
        float depth) {
23          InteractiveVolume volume = new InteractiveVolume(this, id,
        width, height, depth);
24
25          volume.setTranslationOffset(xOffset, yOffset, zOffset);
26          volume.setRotationOffset(xRotationOffset, yRotationOffset,
        zRotationOffset);
27
28          volume.setPos(guitar.getPos().x,
29                  guitar.getPos().y,
30                  guitar.getPos().z);
31          volume.setRotation(guitar.getRotation().x,
32                  guitar.getRotation().y,
33                  guitar.getRotation().z);
34
35          guitar.getInteractions().add(volume);
36
37          volume.setVisualizeVertices(DEBUG_VERTICES);
38      }
```

El método primitivo `draw()` refresca los elementos de la escena y son mostrados por pantalla para su visualización.

```
1       @Override
2       public void draw() {
3           background(0);
4
5           setCamera();
6
7           kinect.doSkeleton(false);
8           kinect.refresh(KinectSelector.NONE, true);
9
10          kinect.doSkeleton(true);
11          kinect.refresh(KinectSelector.NONE, true);
12
13          lights();
14
15          guitarInteraction();
16
17          makeFloor();
18      }
```

En primer lugar, se establece al posición de la cámara en función de la posición del punto de articulación de la columna del esqueleto detectado por la *Kinect* al llamar al método `setCamera()`.

```
1       private void setCamera() {
2           PVector spine = kinect.getSkelPos(KinectAnathomy.SPINE);
3
4           if (spine != null) {
5               PVector camPos = new PVector(width/2.f,
6                       height/2.f,
7                       (height/2.f) / tan(PI * 30.f / 180.f));
8
```

```
            camera(camPos.x, camPos.y, camPos.z,
                    spine.x, spine.y, spine.z,
                    0,1, 0);
        }
    }
```

2.2.2. Paquete `object`

Clase `Object`

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*.

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

```

        this.pos = new PVector(0,0,0);
        this.rotation = new PVector(0,0,0);
```

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

```
        this.texture = texture;
        this.material = material;
        interactions = new ArrayList<>();
    }

    public PVector getPos() {
        return pos;
    }

    public void setPos(PVector pos) {
        this.pos = pos;
        for (InteractiveVolume iv :
                interactions) {
```

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

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()`.

```
        }
    }

    public PVector getRotation() {
        return rotation;
    }

    public void setRotation(PVector rotation) {
        this.rotation = rotation;
        for (InteractiveVolume iv :
                interactions) {
```

```
            iv.setRotation(rotation.x, rotation.y, rotation.z);
        }
    }

    public Material getMaterial() {
        return material;
    }

    public void setMaterial(Material material) {
        this.material = material;
    }

    public PShape getModel() {
        return model;
    }

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

    public List<InteractiveVolume> getInteractions() {
        return interactions;
    }

    public void setInteractions(List<InteractiveVolume>
    interactions) {
        this.interactions = interactions;
    }

    public void scale(float s) {
        model.scale(s);
    }

    public void refresh() {
        if (material != null) material.refresh();
        if (texture != null) model.setTexture(texture.getTexture())
    ;
        setTransformations();
        updateGuitarState();
    }

    private void setTransformations() {
        parent.pushMatrix();
        parent.translate(pos.x, pos.y, pos.z);
        parent.rotateX(rotation.x);
        parent.rotateY(rotation.y);
        parent.rotateZ(rotation.z);
        parent.shape(model, 0, 0);
        parent.popMatrix();
    }
```

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 abstract void touched(int id, PVector joint, int
    inRadius);

    public abstract void doDrawInteractionArea(boolean b);
}
```

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*).

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;
```

```
26      }
27
28      public void setSpecular(PVector specular) {
29          this.specular = specular;
30      }
31
32      public void setShininess(float shininess) {
33          this.shininess = shininess;
34      }
35
36      public PVector getAmbient() {
37          return ambient;
38      }
39
40      public PVector getEmissive() {
41          return emissive;
42      }
43
44      public PVector getSpecular() {
45          return specular;
46      }
47
48      public float getShininess() {
49          return shininess;
50      }
```

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

Chung, B. (s.f.). *Kinect For Processing Library.* http://www.magicandlove.com/blog/research/kinect-for-processing-library/. Magic & Love Interactive. (Accessed: 2019-04-06)

DART. (s.f.). *Kinect Controller.* http://dartecne.wikidot.com/kinect-controller. DART: Dialogs between Art and Technology. (Accessed: 2019-04-06)

Foundation, P. (s.f.). *Sound.* https://processing.org/reference/libraries/sound/index.html. Autor. (Accessed: 2019-04-06)

Processing. (s.f.). *Processing: Javadoc.* https://processing.github.io/processing-javadocs/core/index.html. Autor. (Accessed: 2019-02-19)

stone777. (s.f.). *White guitar 3D model.* https://www.turbosquid.com/FullPreview/Index.cfm/ID/1199062. TurboSquid. (Accessed: 2019-04-06)