Práctica 1 - Pong

David Alberto Medina Medina
Dr. Modesto Fernando Castrillon Santana

11 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 superficies de revolución.

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.

Método y materiales

2.1. Materiales

El desarrollo de este proyecto se ha llevado a cabo utilizando el IDE de *Processing* el cual incorpora las librerías las librerías de desarrollo gráfico *OpenGL*.

2.2. Método

2.2.1. Clase `RotatingShape`

La clase `RotatingShape` inicializará los parámetros que serán utilizados para la construcción de superficies de revolución. Se declaran un objeto de la clase `Profile3D`, el cual define el perfil de revolución, y un objeto de la clase `LinearRotation3D`, que define la superficie de revolución resultado de revolucionar el perfil de rotación sobre sí mismo.

El patrón del perfil de rotación es definido por el usuario al hacer clic en la pantalla pantalla. Este perfil se rota alrededor del eje vertical –de color blanco– que divide la pantalla en dos mitades. La variable `angleSteps` define el número de perfiles de rotación (o longitudes) que se representan por pantalla a lo largo del giro de 360º del perfil patrón sobre este eje.

```
1  Profile3D p;
2  LinearRotation3D lr;
3
4  boolean drawingMode;
5
6  int[] backgroundColor = {10, 20, 50};
7  int[] axisColor = {255, 255, 255};
8  int axisWeight = 3;
9
10 int[] figureColor = {0, 200, 0};
11 int[] figureFill = {220, 220, 220};
12 int figureWeight = 3;
13
14 int longitude = 20;
15 float angle = 360/longitude;
```

El método `setup()` inicializa la pantalla a un tamaño de 800x800 píxeles y habilita el modo de renderización `P3D` que permite pintar objetos tridimensionales en la pantalla. Además, la variable `drawingmode` se inicializa a `TRUE`, habilitándose el modo de dibujo, durante el cual el usuario es capaz de diseñar el patrón del perfil de rotación del objeto que desee representar por pantalla. Se inicializan los siguientes objetos:

1. Un objeto de la clase `Profile3D`, asignándole el color y grosor de la línea del perfil.

2. Un objeto de la clase `LinearRotation3D`, en donde se define el color, grosor y número de cortes longitudinales que se realizan sobre el cuerpo tridimensional, los cuales son necesarios para la construcción de un volumen de revolución.

```
void setup(){
  size(800,800, P3D);

  drawingMode = true;

  p = new Profile3D(figureWeight, figureColor);
  lr = new LinearRotation3D(longitude, figureWeight, figureColor,
    figureFill);
}
```

El método `draw()` ajusta a negro el fondo de pantalla –realizando la llamada a la primitiva `background()`– y el dibuja el eje vertical de rotación en el centro de la pantalla –primitivas `stroke()` y `line()`. Este será el eje sobre el cual se genera la superficie de rotación para el perfil que defina el usuario.

Si el modo *dibujo* está habilitado, se mostrará por pantalla el patrón del perfil de rotación que diseñe el usuario. Sino se muestra el objeto de revolución generado a partir de su perfil.

```
void draw() {
  background(backgroundColor[0], backgroundColor[1],
    backgroundColor[2]);

  strokeWeight(axisWeight);
  stroke(axisColor[0], axisColor[1], axisColor[2]);
  line(width/2, 0, width/2, height);

  if(drawingMode) p.refresh();
  else {
    lr.refresh();
  }
}
```

Cada vez que el usuario haga clic en la pantalla se añade un nuevo punto que define el patron de rotación si y sólo si el modo de dibujo está habilitado. Esta rutina será gestionada por el evento `mouseClicked()`.

```
void mouseClicked() {
  if(drawingMode) {
    p.addVertex(mouseX, mouseY, 0);
  }
}
```

Si el modo *dibujo* está habilitado y se pulsa la tecla `ESPACIO`, se renderizará el objeto de revolución asociado al patrón definido por el usuario y se deshabilita, como resultado, el modo *dibujo*.

Si el modo *dibujo* está deshabilitado, se destruye el objeto de revolución generado y su patrón asociado además de definir el número de lineas de latitud que conforman el cuerpo a generar –el número de líneas de latitud es igual al número de puntos que el usuario haya utilizado para definir el perfil de revolución. Finalmente, se vuelve a habilitar el modo *dibujo*.

```
void keyPressed() {
  if(key == ' ' && drawingMode) {
    lr.setLatitude(p.getVertices().size());
    lr.createSurface(p);
    drawingMode = false;
  } else if(key == ' ' && !drawingMode) {
    p.destroy();
    lr.destroy();
    drawingMode = true;
  }
}
```

2.2.2. Clase `Profile3D`

El perfil de la superficie de revolución es definido en esta clase como una lista de puntos del tipo `PVector` que definen el patrón que el usuario diseñe. El color y grosor la línea del perfil están definidos en las propiedades `colorProfile` y `weight`, respectivamente. Una vez la estructura ha sido definida por el usuario y este pulse la tecla `ESPACIO`, la clase guardará la configuración del perfil en la variable `profile` de tipo `PShape`.

El constructor toma como parámetros de entrada el grosor y un vector de enteros que representa el color del perfil.

```
  private List<PVector> vertices;
  private PShape profile;
  private int weight;
  private int[] colorProfile;

  public Profile3D(int weight, int[] colorProfile) {
    vertices = new ArrayList<PVector>();
    this.weight = weight;
    this.colorProfile = colorProfile;
  }
```

Cada punto que el usuario defina en el modo *dibujo* es añadido en la lista `vertices`, el cual es almacenado en un objeto `PVector` que contiene las coordenadas XYZ de dicho punto. Esta lista de puntos puede, además, ser recuperada llamando a su *getter*.

```
  public void addVertex(float x, float y, float z) {
    vertices.add(new PVector(x, y, z));
  }

  public List<PVector> getVertices() {
    return vertices;
  }
```

El método `refresh()` es el responsable de representar por pantalla el polígono que diseñe el usuario para construir la superficie de revolución llamando al método interno `buildProfile()` y, mostrandolo por pantalla realizando una llamada a la rutina `shape()`. Además, ub objeto de tipo `Profile3D` puede ser destruído al llamar al método `destroy()`.

```
  public void refresh() {
    buildProfile();
    shape(profile);
  }
```

```
5
6   public void destroy() {
7     vertices.clear();
8     profile = null;
9   }
```

El método privado `buildProfile()` instancia un objeto de la clase `PShape` sobre el cual se asignan los puntos de la lista de vértices que definen la figura geométrica plana que el usuario ha diseñado. La definición de los atributos de este objeto `PShape` se configuran después de llamar a la primitiva `beginShape()`. Una vez finalizada la configuración del objeto de tipo `PShape`–color y ancho de línea–, se llama a la primitiva `endShape()`, finalizando, de este modo, la edición y creación de la figura.

```
1   private void buildProfile() {
2     profile = createShape();
3
4     if(vertices.size() < 2) profile.beginShape(POINTS);
5     else if(vertices.size() < 3) profile.beginShape(LINES);
6     else profile.beginShape();
7
8     strokeWeight(8);
9     stroke(255, 0, 0);
10
11    profile.noFill();
12    profile.stroke(colorProfile[0], colorProfile[1], colorProfile
      [2]);
13    profile.strokeWeight(weight);
14
15    for(PVector v: vertices) {
16      profile.vertex(v.x, v.y, v.z);
17      point(v.x, v.y, v.z);
18    }
19
20    if(vertices.size() > 0) {
21      PVector v = vertices.get(0);
22      profile.vertex(v.x, v.y, v.z);
23    }
24
25    profile.endShape();
26  }
```

Finalmente, se dispone de una serie de métodos *getter* para obtener la configuración de color y grosor de línea, así como el objeto de tipo `PShape` que define el patrón de la superficie de revolución.

```
1   public PShape getShape() {
2     buildProfile();
3     return profile;
4   }
5
6   public int[] getColor() {
7     return colorProfile;
8   }
9
10  public int getWeight() {
11    return weight;
12  }
```

2.2.3. Clase `LinearRotation3D`

Esta clase crea una superficie de revolución tridemencional a partir de una matriz de dimensión *longitud* × *latitud* que contiene la nube puntos que definen el contorno de la figura geométrica, en donde

- Cada fila define una línea de puntos de longitud.
- Cada columna define una línea de puntos de latitud.

Además, el constructor configura el color y grosor de la línea, y color de relleno que definen la superficie a generar. El núero de puntos .

```
1  int longitude;
2  int latitude;
3  PVector[][] pointCloud;
4  List<PShape> stripes;
5  int weight;
6  int[] color3DModel;
7  int[] figureFill;
8
9  public LinearRotation3D(int longitude, int weight, int[]
     color3DModel, int[] figureFill) {
10   this.longitude = longitude;
11   latitude = -1;
12   this.weight = weight;
13   this.color3DModel= color3DModel;
14   this.figureFill = figureFill;
15   stripes = new ArrayList<PShape>();
16 }
```

El método público `create3DRotation()` toma como parámetro de entrada un perfil de revolución que define la forma del primer corte meridional de la cuerpo tridimensional que se desea renderizar. Este método llama a su vez a tres métodos privados:

1. El método `createPointsCloud()`, cuyo parámetro de entrada es perfil de revolución, genera la nube de puntos que define la superficie y los ordena en la matriz `pointsCloud`.
2. El método `build3DMode` construye un objeto de tipo `PShape` que se origina como resultado de triangular todos y cada uno de los puntos obtenidos en el método anterior.

```
1  public void createSurface(Profile3D p) {
2    createPointsCloud(p);
3    build3DModel();
4  }
```

En concreto el método `createPointsCloud()` inicializa una lista de objetos `Profile3D` cuyos vértices resultan de una transformación lineal de rotación definida por la siguiente expresión (Academy, s.f.),

$$(x_2, y_2, z_2) = (x_1 - \gamma, y_1, z_1) \begin{pmatrix} \cos\theta & 0 & \sin\theta + \gamma \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

por tanto,

$$(x_2, y_2, z_2) = \begin{cases} x_2 = (x_1 - \gamma)\cos\theta - z_1 \sin\theta + \gamma \\ y_2 = y_1 \\ z_2 = (x_1 - \gamma)\sin\theta + z_1 \cos\theta \end{cases}$$

tal que, para todo punto del perfil de revolución de coordenadas (x_1, y_1, z_1), se obtiene un nuevo punto (x_2, y_2, z_2) que resulta de la rotación sobre un eje vertical a una distancia $\gamma = w/2$ del origen de coordenadas, siendo, w, el ancho de la ventana que contiene el cuerpo a renderizar. El ángulo θ define cada uno de los cortes meridionales que se sobre el volumen a renderizar a lo largo de sus 350º.

Finalmente, los puntos (x_2, y_2, z_2), resultado de la transformación lineal de rotación, son almacenados y ordenados según la latitud y longitud en que se encuentren en el array de dos dimensiones, `pointsCloud`, como una instancia de objetos de tipo `PVector`.

```
1  private void createPointsCloud(Profile3D p) {
2    PShape shape = p.getShape();
3
4    pointCloud = new PVector[longitude][latitude];
5
6    for(int i = 0; i < longitude; i++) {
7      float theta = i*2*PI/longitude;
8      for(int j = 0; j < latitude; j++) {
9        PVector vertex = shape.getVertex(j);
10       float gamma = width/2;
11       float x = (vertex.x - gamma)*cos(theta) - vertex.z*sin(
   theta) + gamma;
12       float y = vertex.y;
13       float z = (vertex.x - gamma)*sin(theta) + vertex.z*cos(
   theta);
14       pointCloud[i][j] = new PVector(x, y, z);
15     }
16   }
17 }
```

Siguiendo el mismo procedimiento descrito en la sección 2.2.2, el método `build3DModel()` construye un modelo tridimensional con todos y cada uno de los puntos definidos en la matriz `pointsCloud`. La renderización y triangulación se lleva a cabo al definir los puntos que se encuentra en la superficie al recorrer horizontalmente dos líneas de latitud contigüas y, utilizando el parámetro `TRIANGLE_STIP` para llevar a cabo el proceso de triangulado. Este proceso se repite tantas veces como sean necesarias para cubrir toda la superficie del cuerpo tridimensional a generar.

```
1  private void build3DModel() {
2    for(int i = 0; i <= longitude; i++) {
3
4      PShape model3D = createShape();
5      model3D.beginShape(TRIANGLE_STRIP);
6
7      model3D.stroke(color3DModel[0], color3DModel[1], color3DModel
   [2]);
8      model3D.strokeWeight(weight);
9      model3D.fill(figureFill[0], figureFill[1], figureFill[2]);
10
11     for(int j = 0; j <= latitude; j++) {
```

```
        PVector v1 = pointCloud[i%longitude][j%latitude];
        model3D.vertex(v1.x, v1.y, v1.z);
        PVector v2 = pointCloud[(i+1)%longitude][j%latitude];
        model3D.vertex(v2.x, v2.y, v2.z);
      }

      model3D.endShape();
      stripes.add(model3D);
    }
  }
```

El método `refresh` contiene una pequeña rutina que nos permitirá actualizar la posición del modelo en la pantalla en la posición sobre la que se encuentre el ratón. Para ello es necesario llamar al método privado `getCentroid` para hallar el centroide de la superficie generada, de modo que nos permita mover el centro del cuerpo justo en donde se encuentra el ratón. La primitiva es `translate()` es la responsable de que este proceso se lleve a cabo.

Finalmente, se llama a la rutina `point` y `shape()` para mostrar por pantalla el modelo 3D generado por el usuario, así como los puntos que la definen.

El método `destroy()` destruirá el modelo generado y vaciará las listas que contienen los meridianos y paralelas del mismo, de modo que deja el objeto preparado para generar nuevas superficies de revolución.

```
  public void refresh() {
    PVector centroid = getCentroid();
    translate(mouseX-centroid.x, mouseY-centroid.y, 0);
    strokeWeight(10);
    for(int i = 0; i < longitude; i++) {
      for(int j = 0; j < latitude; j++) {
        PVector v = pointCloud[i][j];
        point(v.x, v.y, v.z);
      }
    }
    for(PShape stripe: stripes) {
      shape(stripe);
    }
  }

  public void destroy() {
    pointCloud = null;
    stripes.clear();
  }
```

Se dispone de un *setter* asignar el valor de la latitud con el número de puntos que se han utilizado para diseñar el perfil del cuerpo de revolución.

```
  public void setLatitude(int latitude) {
    this.latitude = latitude;
  }
```

La rutina *getCentroid()* calcula el centroide de la figura geométrica por medio de la media aritmética por componentes de todos y cada uno de los puntos que definen la superficie (*Wikipedia: Centroids*, s.f.).

```
  public PVector getCentroid() {
    int xSum = 0, ySum = 0, zSum = 0;
    int numPoints = latitude*longitude;
    for(int i = 0; i < longitude; i++) {
      for(int j = 0; j < latitude; j++) {
```

```
6           PVector v = pointCloud[i][j];
7           xSum += v.x;
8           ySum += v.y;
9           zSum += v.z;
10        }
11      }
12      return new PVector(xSum/numPoints, ySum/numPoints, zSum/
        numPoints);
13    }
```

Resultados y discusión

Las técnicas aquí descritas son útiles para diseñar cuerpos tridimensionales simétricos a partir de una transformación lineal de rotación de los puntos que definen el perfil de rotación de cualquier figura geométrica plana que el usuario defina como patrón (ver figura 3).

Cuando se aplica la transformación lineal sobre este patrón, se observa que el contorno de la figura plana define un volumen que mediante un proceso de renderizado y triangulado definen la superficie de un cuerpo de revolución simétrico con respecto al eje de rotación –línea de color blanco–, obteniéndose como resultado un cuerpo tridimensional (ver figura 3)

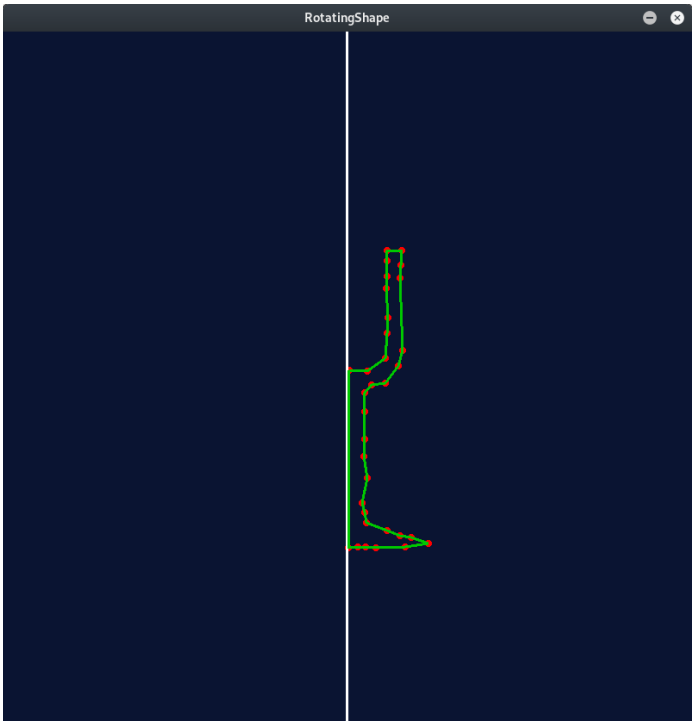

Figura 3.1: Figura geométrica plana que define el patrón que genera el cuerpo de revolución

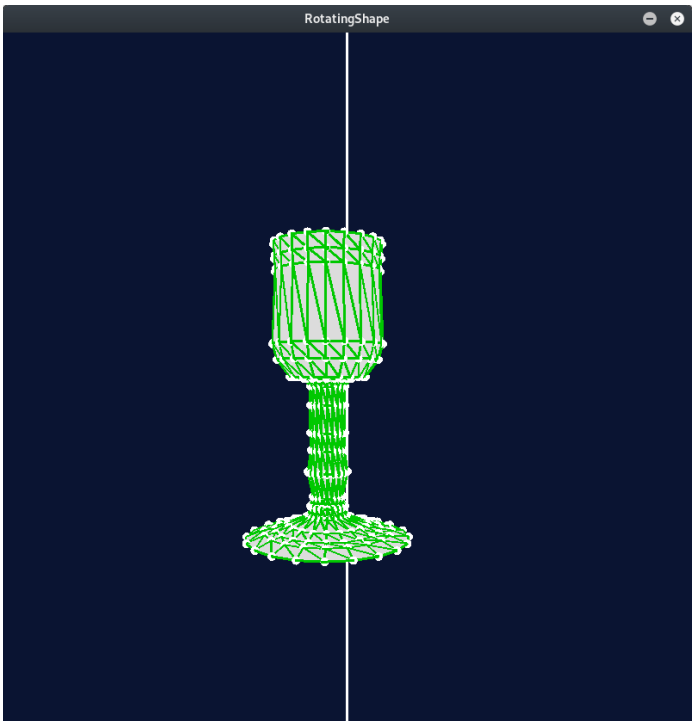

Figura 3.2: Volumen de revolución que se genera al hacer girar el patrón generatriz sobre el eje de rotación –linea de color blanco.

Conclusiones

El álgebra lineal y, en concreto, las transformaciones lineales son los conceptos más destacados y explotados durante todo el desarrollo de esta experiencia. Por medio de la definición de un contorno plano, es posible generar cuerpos simétricos al aplicar rotaciones. Estas sencillas técnicas definen los fundamentos del renderizado tridimensional.

Por medio de la técnica de triangulación generar planos bien definidos que serán utilizados por técnicas oclusivas como la técnica del *Z-buffer*, ofreciéndo un efecto de profundidad y tridimensionalidad al cuerpo generado.

Referencias

Academy, K. (s.f.). *Rotar Figuras 3D.* https://es.khanacademy.org/computing/computer-programming/programming-games-visualizations/programming-3d-shapes/a/rotating-3d-shapes. Autor. (Accessed: 2019-02-11)

Wikipedia: Centroids. (s.f.). https://en.wikipedia.org/wiki/Centroid. (Accessed: 2019-02-11)