Java Racer

Versión 2 – carretera con curvas

Curvas

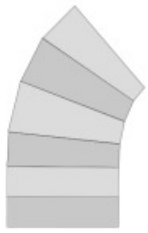
- En la versión 2 de Java Racer se le agregarán curvas a la carretera.

Curvas

- En la versión 1, los puntos solo utilizan la coordenada z.

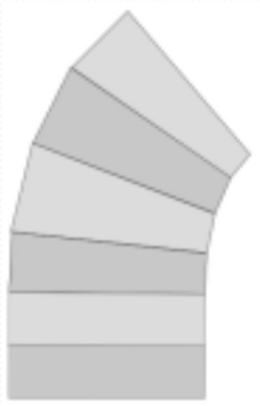
```
private void resetRoad()
    …
    PointProjection p1 = new PointProjection(new Point3D(0, 0, n * segmentLength), new Point3D(), new Point4I());
    PointProjection p2 = new PointProjection(new Point3D(0, 0, (n + 1) * segmentLength), new Point3D(), new Point4I());
    …
```

- Las coordenadas x y y valen 0 porque no se utilizan.



Curvas

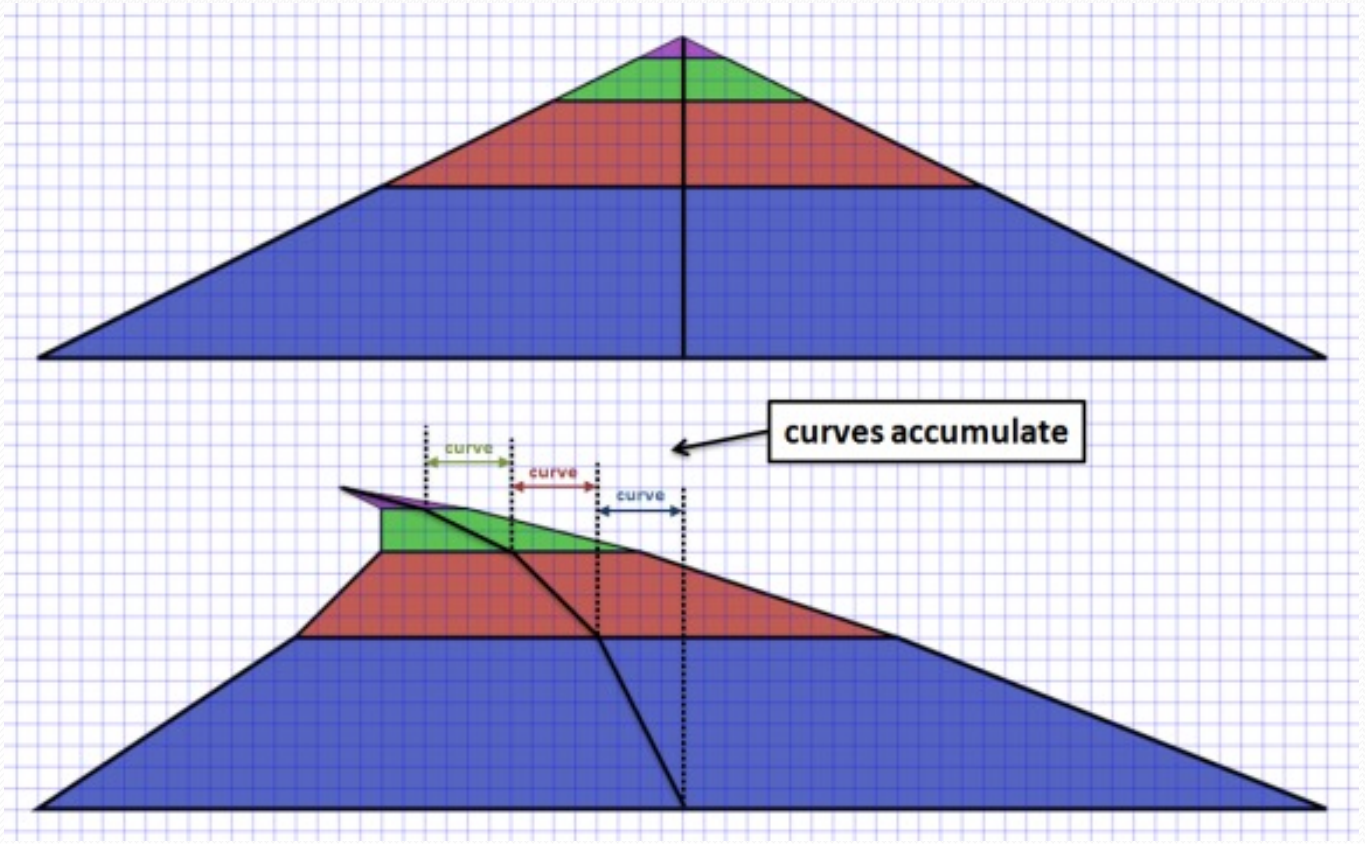
- En un programa 3D las curvas se pueden implementar calculando las coordenadas x y z para una serie de segmentos como esta:



Curvas

- La alternativa es usar *pseudo 3D* para simular curvas.
- La idea es que la imagen se *vea bien* aunque no sea físicamente realista.
- “Para curvar una carretera, solo se necesita cambiar la posición de la línea del centro en forma de curva. Comenzando en la parte baja de la pantalla, la cantidad que el centro de la carretera se recorre a la izquierda o derecha aumenta continuamente.”

Curvas



Implementación

- La estructura del código es similar a la versión 1.
- Clases que no cambian:
  - Paquete constant: Background, Colors, ColorTheme, Key, Sprites.
  - Paquete util: Point3D, Point4I, PointProjection, Util.
  - Paquete racer: Render.
- Clases que se actualizan: geometry.Segment, racer.Game.
- Clase nueva: geometry.Road.

Implementación

- En Java Racer, el centro es el valor cameraX que se le pasa a Util.project.
- Conforme se dibuja cada segmento de la carretera, las curvas se simulan aumentando cameraX.
- Se necesita agregar a la clase Segment un campo curve.
- Este valor representa que tanto se recorre el segmento del centro de la cámara.

Implementación

- Este valor debe ser:
- Negativo para curvas a la izquierda.
- Positivo para curvas a la derecha.
- Pequeño para curvas sencillas.
- Grande para curvas difíciles.
- Se agrega la clase Road con dos objetos enum.

Clase Road

```
public class Road {
    public enum LENGTH {
        NONE(0), SHORT(25), MEDIUM(50), LONG(100);  // Número de segmentos
        private final int length;
        LENGTH(int length) {
            this.length = length;
        }
        public int getValue() {
            return length;
        }
    }
```

Clase Road

```
    public enum CURVE {
        NONE(0), EASY(2), MEDIUM(4), HARD(6);   // incrementos
        private final int curve;
        CURVE(int curve) {
            this.curve = curve;
        }
        public int getValue() {
            return curve;
        }
    }
}
```

Interpolación

- La transición cuando una recta se convierte en curva se puede suavizar mediante interpolación.
- El valor curve se incrementa (o decrementa) lentamente hasta que toma el valor deseado.
- Se agregan las siguientes funciones de interpolación a la clase Util.

Clase Util

```
public static double easeIn(double a, double b, double percent) {
    return a + (b - a) * Math.pow(percent, 2);
}
public static double easeOut(double a, double b, double percent) {
    return a + (b - a) * (1 - Math.pow(1 - percent, 2));
}
public static double easeInOut(double a, double b, double percent) {
    return a + (b - a) * ((-Math.cos(percent * Math.PI) / 2.0) + 0.5);
}
```

Clase Game

- Se agregan métodos para agregar una recta, un segmento, curvas y para entrar, mantenerse y salir de una curva:

```
private void addStraight(int num)
private void addSegment(double curve)
private void addSCurves()
private void addCurve(int num, int curve)
private void addRoad(double enter, double hold, double leave, double curve)
```

Método addStraight

```
private void addStraight(int num)
{
    addRoad(num, num, num, 0);
}
```

Método addSegment

```
private void addSegment(double curve)
{
    int n = segments.size();
    segments.add(new Segment(
            n,
            new PointProjection(new Point3D(0, 0, n * segmentLength), new Point3D(),
new Point4I()),
            new PointProjection(new Point3D(0, 0, (n + 1) * segmentLength), new
Point3D(), new Point4I()),
            curve,
            (n / rumbleLength) % 2 == 1 ? Colors.DARK : Colors.LIGHT));
}
```

Método addRoad

```
private void addRoad(double enter, double hold, double leave, double curve) {
    for (int n = 0; n < enter; n++) {
        addSegment(Util.easeIn(0, curve, n / enter));
    }
    for (int n = 0; n < hold; n++) {
        addSegment(curve);
    }
    for (int n = 0; n < leave; n++) {
        addSegment(Util.easeInOut(curve, 0, n / leave));
    }
}
```

Método addCurve

```
private void addCurve(int num, int curve)
{
    addRoad(num, num, num, curve);
}
```

Método addSCurves

```
private void addSCurves() {
   addRoad(Road.LENGTH.MEDIUM.getValue(),  Road.LENGTH.MEDIUM.getValue(),
         Road.LENGTH.MEDIUM.getValue(), -Road.CURVE.EASY.getValue());
   addRoad(Road.LENGTH.MEDIUM.getValue(),  Road.LENGTH.MEDIUM.getValue(),
         Road.LENGTH.MEDIUM.getValue(),  Road.CURVE.MEDIUM.getValue());
   addRoad(Road.LENGTH.MEDIUM.getValue(),  Road.LENGTH.MEDIUM.getValue(),
         Road.LENGTH.MEDIUM.getValue(),  Road.CURVE.EASY.getValue());
   addRoad(Road.LENGTH.MEDIUM.getValue(),  Road.LENGTH.MEDIUM.getValue(),
         Road.LENGTH.MEDIUM.getValue(), -Road.CURVE.EASY.getValue());
   addRoad(Road.LENGTH.MEDIUM.getValue(),  Road.LENGTH.MEDIUM.getValue(),
         Road.LENGTH.MEDIUM.getValue(), -Road.CURVE.MEDIUM.getValue());
}
```

Método resetRoad

- El método resetRoad se cambia para que invoque a los métodos anteriores.

Método resetRoad

```
private void resetRoad()
{
    segments.clear();
    addStraight(Road.LENGTH.SHORT.getValue() / 4);
    addSCurves();
    addStraight(Road.LENGTH.LONG.getValue());
    addCurve(Road.LENGTH.MEDIUM.getValue(),
Road.CURVE.MEDIUM.getValue());
    addCurve(Road.LENGTH.LONG.getValue(),
Road.CURVE.MEDIUM.getValue());
    addStraight(Road.LENGTH.MEDIUM.getValue());
    addSCurves();
```

Método resetRoad

```
    // Agrega más segmentos (ver código fuente)
    …
    segments.get(findSegment(playerZ).index + 2).color = Colors.START;
    segments.get(findSegment(playerZ).index + 3).color = Colors.START;
    for (var n = 0; n < rumbleLength; n++) {
        segments.get(segments.size() - 1 - n).color = Colors.FINISH;
    }
    trackLength = segments.size() * segmentLength;
}
```

Cambios en update

- Se aplica una fuerza centrífuga al automóvil cuando va en una curva.

```
private final double centrifugal = 0.3;
```

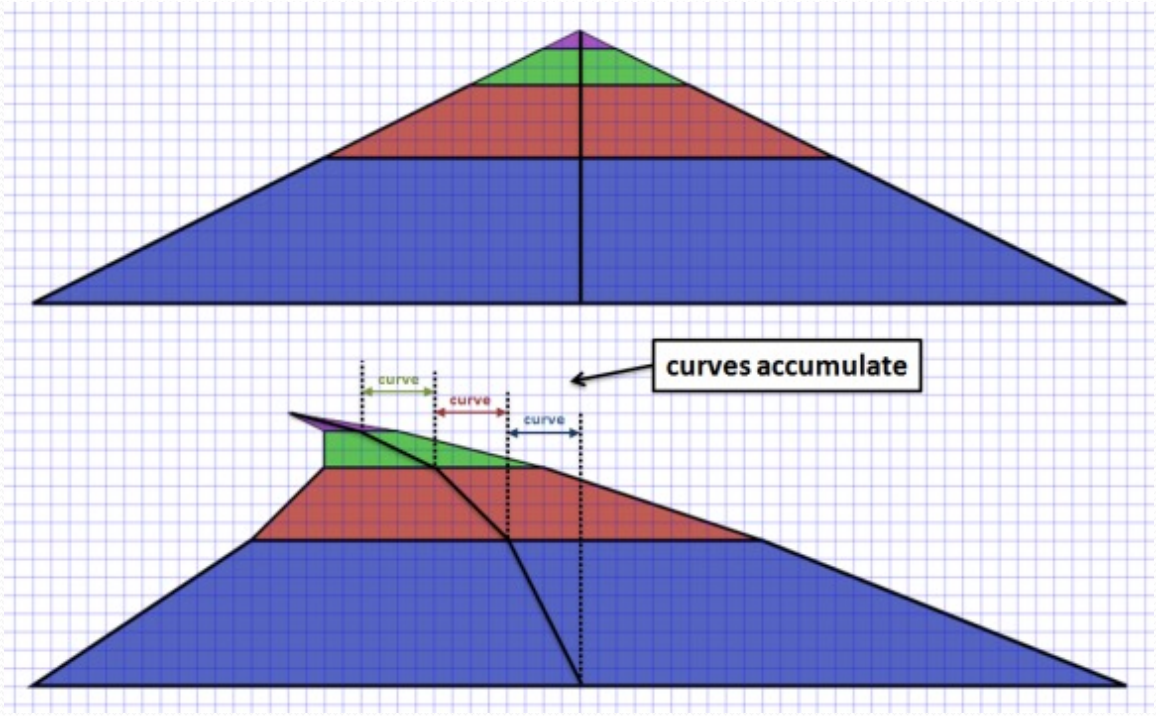
- Y se actualiza la posición playerX en base a la velocidad actual, la cantidad de curva y la fuerza centrífuga.

```
playerX = playerX - (dx * speedPercent * playerSegment.curve * centrifugal);
```

Dibujando curvas

- Las curvas se simulan cambiando el valor de cameraX que se usa en los cálculos de proyecciones al dibujar cada segmento de la carretera.

Dibujando curvas



Dibujando curvas

- Se necesita una variable acumulador dx que incremente la cantidad de curva (campo curve) de cada segmento.
- También se incrementa la variable x que se usa para recorrer cameraX.
- En resumen:
  - Recorrer la proyección de p1 de cada segmento por x.
  - Recorrer la proyección de p2 de cada segmento por x + dx.
  - Incrementar x en dx para el siguiente segmento.
- Finalmente, para suavizar la transición dx se inicializa con un valor interpolado de la curva del segmento actual.

Método render modificado

```
Segment baseSegment = findSegment(position);
double basePercent = Util.percentRemaining(position, segmentLength);
int maxY = height;
double x = 0;
double dx = -(baseSegment.curve * basePercent);
```

Método render modificado

```
for (int n = 0; n < drawDistance; n++) {
    …
    Util.project(segment.p1, (playerX * roadWidth) - x, cameraHeight,
      position - (segment.looped ? trackLength : 0), cameraDepth, width, height,
roadWidth);
    Util.project(segment.p2, (playerX * roadWidth) - x - dx, cameraHeight,
      position - (segment.looped ? trackLength : 0), cameraDepth, width, height,
roadWidth);
      x  = x + dx;
      dx = dx + segment.curve;
    …
}
```

Fondo con parallax scrolling

- Para lograr este efecto se mantiene un offset para cada capa.

```
private final double skySpeed = 0.001;
private final double hillSpeed = 0.002;
private final double treeSpeed = 0.003;
private double skyOffset = 0;
private double hillOffset = 0;
private double treeOffset = 0;
```

Fondo con parallax scrolling

- Los offsets se incrementan en update en base a la curva del segmento actual del jugador y a su velocidad.

```
skyOffset = Util.increase(skyOffset, skySpeed * playerSegment.curve * speedPercent, 1);
hillOffset = Util.increase(hillOffset, hillSpeed * playerSegment.curve * speedPercent, 1);
treeOffset = Util.increase(treeOffset, treeSpeed * playerSegment.curve * speedPercent, 1);
```

Fondo con parallax scrolling

- Los offsets se utilizan en el método render.

```
Render.background(g2, background, width, height, Background.SKY,
skyOffset);
Render.background(g2, background, width, height, Background.HILLS,
hillOffset);
Render.background(g2, background, width, height, Background.TREES,
treeOffset);
```

Resultado

- Lista la versión con curvas falsas con pseudo 3D.