Java Racer

Versión 1 – carretera recta

Out Run

- Out Run (también escrito como OutRun) es un videojuego arcade de conducción lanzado por Sega en septiembre de 1986.
- El objetivo es evitar el tráfico y llegar a uno de los cinco destinos.

Out Run

- Es un juego en seudo-3D (también conocido como 2.5D).
- Se usan proyecciones para simular objetos en 3D, cuando en realidad son objetos en 2D.

JavaScript Racer

- https://codeincomplete.com/articles/javascript-racer/
- 4 versiones:

1. Versión 1. Carretera recta.
2. Versión 2. Curvas.
3. Versión 3. Colinas.
4. Versión 4. Versión final.

- La versión final se puede jugar en https://codeincomplete.com/games/racer/

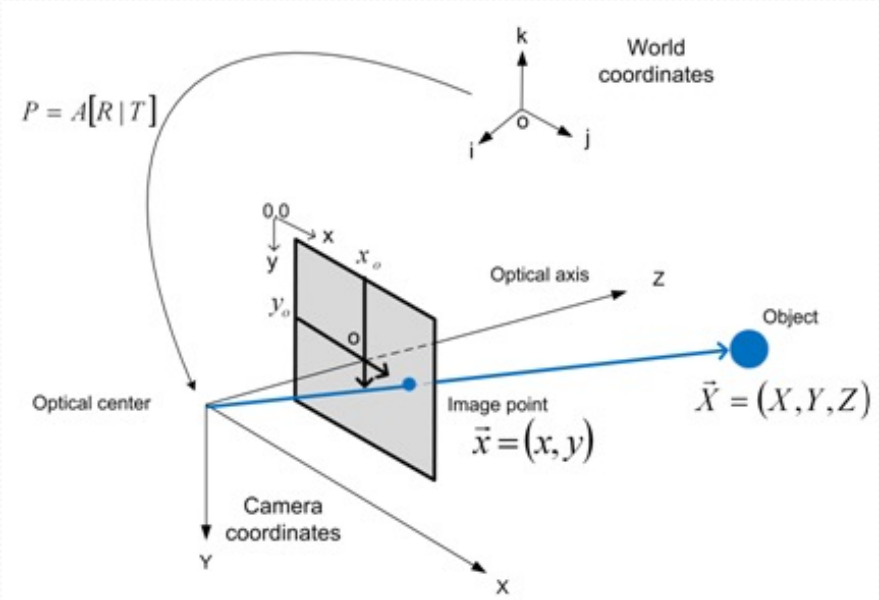
Java Racer – versión 1

- Temas:
- Proyección 3D a 2D.
- Estructura del código.
- Ciclo del juego.
- Imágenes y sprites.
- Variables del juego.
- Manejando un Ferrari.
- Geometría de la carretera.
- Dibujando el fondo.
- Dibujando el automóvil.

Proyección 3D a 2D

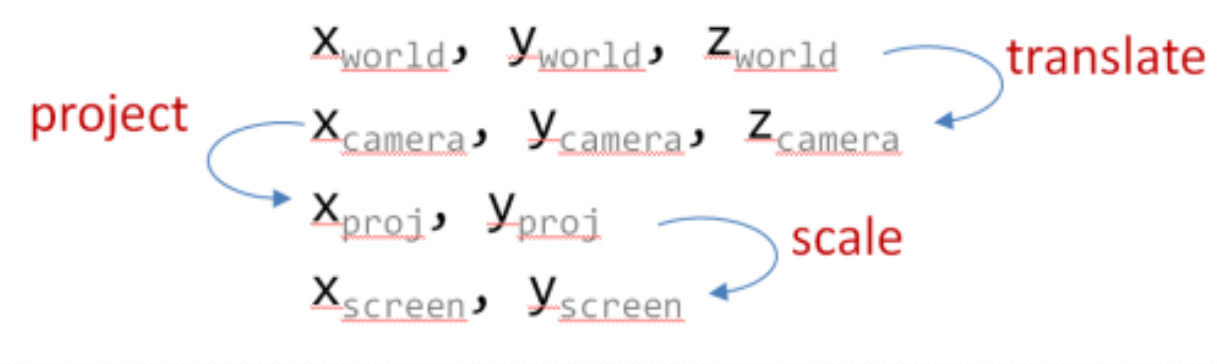
- Se necesita proyectar un punto de un mundo 3D a una pantalla 2D.
- Se coloca una cámara virtual en el mundo 3D.
- Frente a la cámara virtual se coloca un plano, llamado el plano de proyección.
- El objetivo es proyectar los objetos del mundo 3D en dicho plano.
- Luego el plano de proyección se proyecta a la pantalla (en un JFrame o JPlane).

Proyección 3D a 2D



http://archimede.bibl.ulaval.ca/archimede/fichiers/25229/25229_35.png

Camino de un punto



Trasladar

- Cambia las coordenadas de un punto de coordenadas del mundo a coordenadas de la cámara.
- $x_{\text{camera}} = x_{\text{world}} - \text{cameraX}$
- $y_{\text{camera}} = y_{\text{world}} - \text{cameraY}$
- $z_{\text{camera}} = z_{\text{world}} - \text{cameraZ}$
- NOTA: en un verdadero sistema 3D también habría un paso de rotación, pero como las curvas serán falsas, no hay que preocuparse por la rotación.

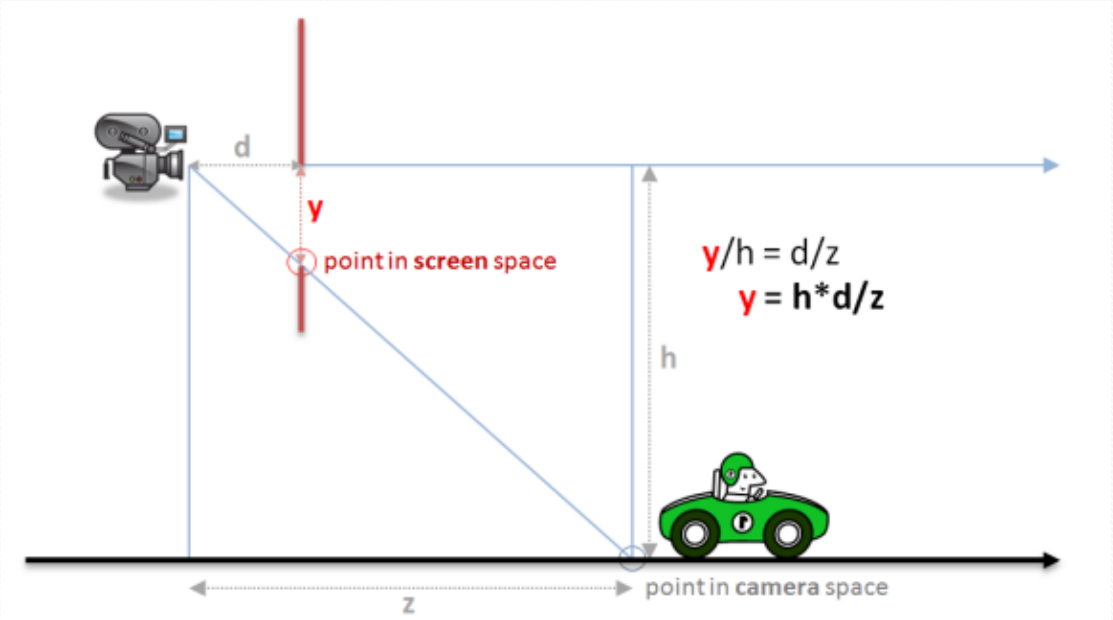
Proyectar

- Cambia las coordenadas de un punto de coordenadas de la cámara a coordenadas del plano de proyección.
- La proyección es de 3D a 2D y utiliza la ley de los triángulos semejantes:
- Sean:
- h la altura de la cámara.
- d la distancia de la cámara al plano de proyección.
- z la distancia de la cámara al objeto.
- y la coordenada y en el plano de proyección.

Proyectar

- El objetivo es calcular y a partir de h, d y z.
- $y = h \times d / z$

Proyectar



Proyectar

- Se puede dibujar un diagrama similar desde una vista de arriba hacia abajo y obtener una ecuación para calcular la coordenada x del plano de proyección:
- $x = w \times d/z$
- Donde w es la mitad del ancho de la carretera (desde la cámara hasta el borde de la carretera)
- Se puede ver que tanto para x como para y, lo que se hace es escalar por un factor de d/z.

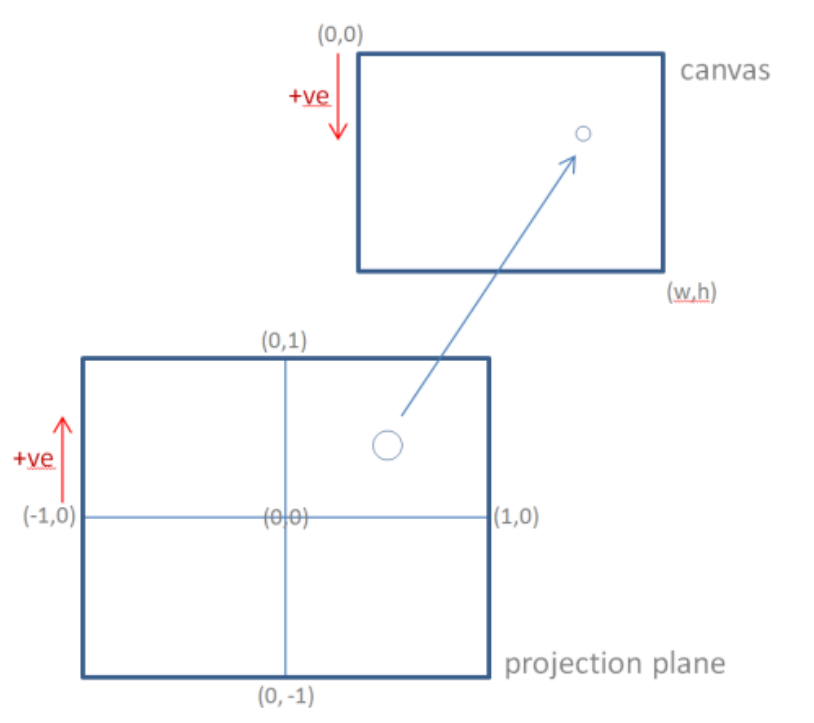
Proyectar

- En resumen:
- $x_{\text{proj}} = x_{\text{camera}} \times d / z_{\text{camera}}$
- $y_{\text{proj}} = y_{\text{camera}} \times d / z_{\text{camera}}$

Escalar

- Cambia las coordenadas de un punto de coordenadas del plano de proyección a coordenadas de la pantalla.
- Hay que tomar en cuenta que:
- En el plano de proyección, el origen está en el centro y el eje y va hacia arriba.
- En la pantalla, el origen está en la esquina superior izquierda y el eje y va hacia abajo.

Escalar



Escalar

- $x_{\text{screen}} = \left(\frac{w}{2}\right) + \left(\frac{w}{2}\right) \times x_{\text{proj}}$
- $y_{\text{screen}} = \left(\frac{h}{2}\right) + \left(\frac{h}{2}\right) \times y_{\text{proj}}$
- Donde w y h son el ancho y el alto de la pantalla en pixeles, respectivamente.

Ecuaciones

- En resumen:

translate

$$x_{camera} = x_{world} - cameraX$$
$$y_{camera} = y_{world} - cameraY$$
$$z_{camera} = z_{world} - cameraZ$$

project

$$x_{proj} = x_{camera} * d/z_{camera}$$
$$y_{proj} = y_{camera} * d/z_{camera}$$

scale

$$x_{screen} = (w/2) + (w/2)*x_{proj}$$
$$y_{screen} = (h/2) - (h/2)*y_{proj}$$

*... where (**w**,**h**) = canvas **width** and **height***

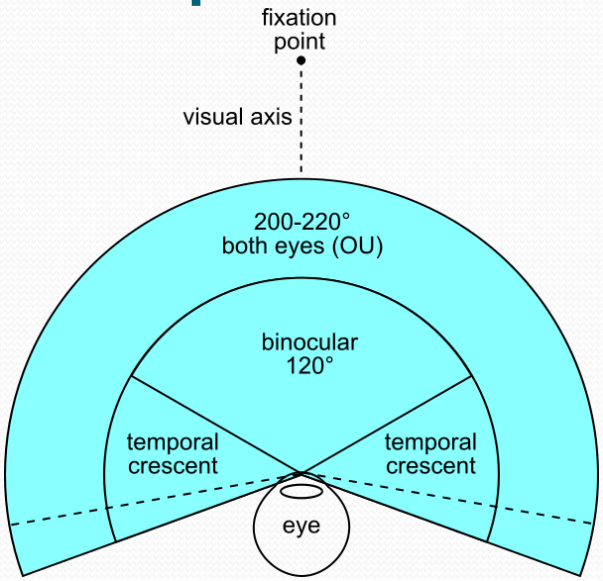
Distancia al plano de proyección

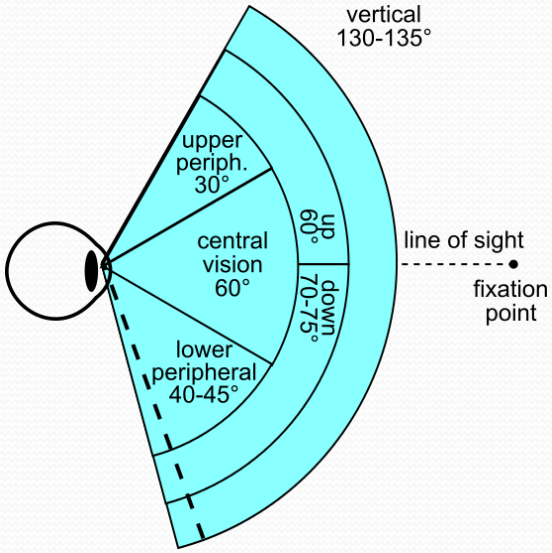
- Hay dos opciones:

1. Dejar la distancia fija.
2. Calcular la distancia a partir del campo de visión (field of view o fov).

- La segunda opción es más útil porque facilita hacer un zoom si se desea.

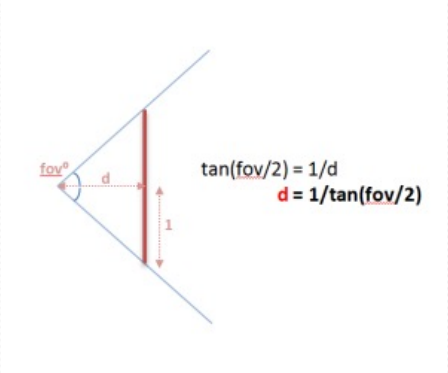
Campo de visión





- Fuente: https://en.wikipedia.org/wiki/Field_of_view

Distancia al plano de proyección



- Suponiendo un plano de proyección normalizado con coordenadas entre -1 y 1:
- $d = 1/\tan(\text{fov}/2)$

Estructura del código

- Clases principales:
- Main. Extiende a JFrame. Define el panel de control. Contiene el método main.
- Game. Extiende a JPanel. Contiene toda la lógica del juego.
- Render. Métodos de ayuda para dibujar el juego.
- Util. Métodos matemáticos de ayuda y otras utilerías.

Estructura del código

- Otras clases:
- Background. Sprites del fondo.
- Colors. Colores de la carretera.
- ColorTheme. Colores para la carretera, pasto, franja de ruido y separador de carriles.
- Key. Códigos de las teclas para controlar el automóvil.
- Sprites. Sprites del juego (automóvil, carteles, árboles).
- Segment. Define un segmento de la carretera.

Estructura del código

- Point3D. Define un punto en coordenadas del mundo (coordenadas x, y, z).
- Point4I. Define un punto en coordenadas de la pantalla (coordenadas x, y, w, scale).
- PointProjection. Define un punto en la carretera con coordenadas del mundo, de la cámara y de la pantalla.

Clase Game

- La clase Game extiende a JPanel y realiza lo siguiente:
- Su constructor carga las imágenes y construye la carretera en base a segmentos (clase Segment).
- Crea un objeto ScheduledExecutorService para invocar al método update cierto número de veces por segundo.
- Contiene el método paintComponent para dibujar el juego.
- paintComponent invoca al método render que es el que hace todo el trabajo.

Ciclo de juego

- El método update actualiza el juego.
- Se invoca de manera repetida mediante un objeto ScheduledExecutorService.

```
Runnable r = this::update;
ScheduledExecutorService executor =
Executors.newSingleThreadScheduledExecutor();
executor.scheduleAtFixedRate(r, 10, 1000 / FPS,
TimeUnit.MILLISECONDS);
```

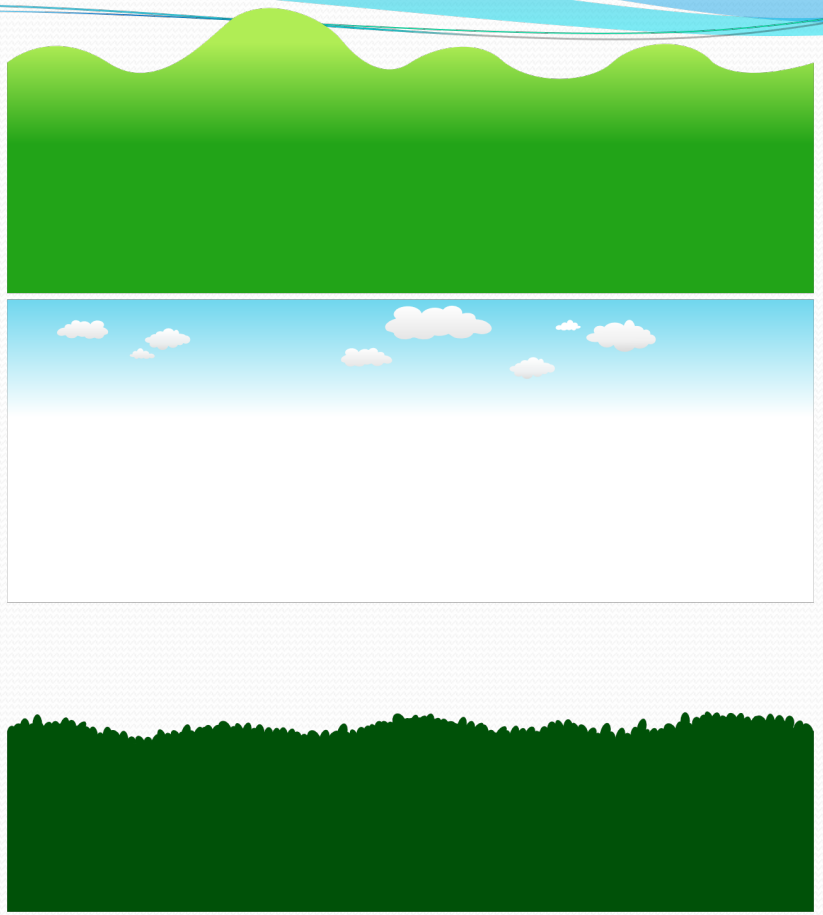
- Donde FPS es el número de frames por segundo (p.e. 60).

Imágenes y sprites

- Al comenzar el juego se cargan dos hojas de sprites:

1. background.png. Tres capas para el cielo, colinas y árboles.
2. sprites.png. Sprites del automóvil (además de árboles y carteles para la versión final).

Fondo



Sprites


SNAKES
DIVING SCHOOL
Captains
SEGA
BIRIN
LiquidPlanner
CityofRedmond
DANKE!
Code inComplete

Imágenes y sprites

- La clase Sprite almacena las coordenadas x, y, w, h de los sprites.

```
public class Sprites {
    public static final Rectangle
        PALM_TREE = new Rectangle(5, 5, 215, 540),
        BILLBOARD08 = new Rectangle(230, 5, 385, 265),
        TREE1 = new Rectangle(625, 5, 360, 360),
        DEAD_TREE1 = new Rectangle(5, 555, 135, 332),
        BILLBOARD09 = new Rectangle(150, 555, 328, 282),
            …
```

Variables del juego

- Además de los sprites se necesitan variables para almacenar el estado del juego.
- Las variables están declaradas en la clase Game.
- Algunas variables:

Variables del juego

```
    private static final int FPS = 60;      // how many 'update' frames per second
    private static final double STEP = 1.0 / FPS;     // how long is each frame (in seconds)
    private int width = 1024;               // logical canvas width
    private int height = 768;               // logical canvas height
    private final List<Segment> segments = new ArrayList<>();   // array of road segments
    private Image background;               // our background image (loaded below)
    private Image sprites;                  // our sprite sheet (loaded below)
```

Manejando un Ferrari

- El automóvil del jugador se controla mediante el teclado:
- Flecha arriba / letra “w”: acelera.
- Flecha izquierda / letra “a”: gira a la izquierda.
- Flecha derecha / letra “d”: gira a la derecha.
- Flecha abajo / letra “s”: frena.
- Si no se oprime la flecha arriba ni la flecha abajo el automóvil se frena lentamente.

Estado del jugador

- Las variables que representan el estado del jugador son:
- speed. La velocidad actual.
- position. La posición actual Z en la pista. Notar que en realidad es la posición de la cámara, no del automóvil.
- playerX. La posición actual X en la carretera. Normalizada entre -1 y +1 para ser independiente del valor de roadWidth.
- Estás variables se actualizan en el método update.

Método update

- Actualiza position basada en el valor de speed.
- Actualiza playerX si la flecha izquierda o la flecha derecha están oprimidas.
- Aumenta speed si la flecha arriba está oprimida.
- Disminuye speed si la flecha abajo está oprimida.
- Disminuye speed si la flecha arriba ni la flecha abajo están oprimidas.
- Disminuye speed si playerX está afuera de la carretera.
- Para caminos rectos, el método update es sencillo.

Método update

```
private void update() {
    position = Util.increase(position, STEP * speed, trackLength);
    double dx = STEP * 2 * (speed / maxSpeed); // at top speed, should be
able to cross from left to right (-1 to 1) in 1 second
    if (keyLeft) {
        playerX = playerX - dx;
    }
    else if (keyRight) {
        playerX = playerX + dx;
    }
```

Método update

```
if (keyFaster) {
    speed = Util.accelerate(speed, acceleration, STEP);
}
else if (keySlower) {
    speed = Util.accelerate(speed, breaking, STEP);
}
else {
    speed = Util.accelerate(speed, deceleration, STEP);
}
```

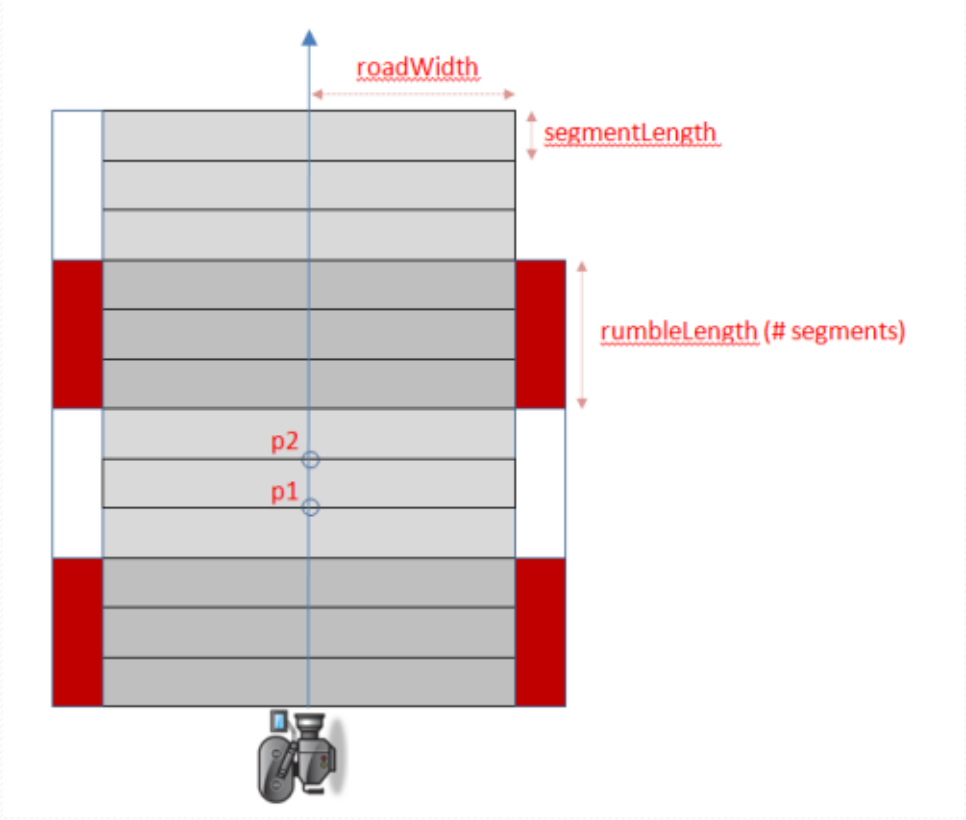
Método update

```
        if ((playerX < -1 || playerX > 1) && (speed > offRoadLimit)) {
            speed = Util.accelerate(speed, offRoadDeceleration, STEP);
        }
        playerX = Util.limit(playerX, -2, 2);      // don't ever let player go too far
out of bounds
        speed = Util.limit(speed, 0, maxSpeed);      // or exceed maxSpeed
        repaint();
    }
```

Geometría de la carretera

- La carretera está compuesta de segmentos.
- El arreglo de segmentos se construye en el método resetRoad.
- Se almacenan dos puntos por cada segmento:
- p1 es el centro del borde más cercano a la cámara.
- p2 es el centro del borde más lejano a la cámara.
- rumbleLength es la longitud de las franjas de ruido (rumble strips) a los lados de la carretera. Por default vale 3.

Carretera



Método resetRoad

```
private void resetRoad()
{
    segments.clear();
    for (int n = 0; n < 500; n++) {    // arbitrary road length
        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());
        ColorTheme color = (n / rumbleLength) % 2 == 0 ? Colors.DARK :
Colors.LIGHT;
        segments.add(new Segment(n, p1, p2, color));
    }
```

Método resetRoad

```
    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;
}
```

Método resetRoad

- p1 y p2 se inicializan solo con las coordenadas z del mundo porque la carretera es recta.
- Las coordenadas y son siempre 0.
- Las coordenadas x están escaladas entre –roadWidth y +roadWidth.
- Esto va a cambiar cuando se agreguen curvas y colinas.
- Se agregan objetos vacíos para luego almacenar las representaciones de estos puntos en coordenadas de la cámara y de la pantalla.
- Cuando el automóvil llega al final de la carretera se regresa al comienzo.

Método findSegment

- Dado un valor de z regresa el segmento correspondiente, aún si se extiende más allá de la longitud de la carretera.

```
private Segment findSegment(double z)
{
    return segments.get((int) Math.floor(z / segmentLength) % segments.size());
}
```

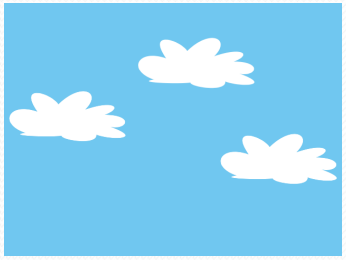
Parallax scrolling

- Es una técnica en gráficos en la que las imágenes de fondo pasan por la cámara más lentamente que las imágenes de primer plano.
- Esto crea una ilusión de profundidad en una escena 2D.

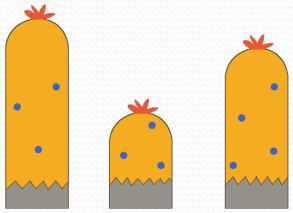
Parallax scrolling

- Ejemplo tomado de https://en.wikipedia.org/wiki/Parallax_scrolling.
- Imágenes:
  - By OhSqueezy - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=15775522
  - By OhSqueezy - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=15775493
  - By OhSqueezy - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=15775410
  - By OhSqueezy - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=15775352

Parallax scrolling

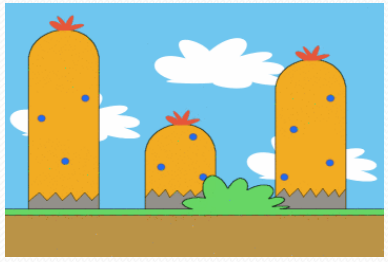


Cielo (capa trasera)



Vegetación (capa media)

Suelo (capa frontal)



Animación

Dibujando el fondo

- El método render comienza dibujando el fondo.
- El fondo se dibuja por separado en 3 capas para que, cuando se agreguen curvas y colinas, exista parallax scrolling.

```
private void render(Graphics2D g2)
{
    Render.background(g2, background, width, height, Background.SKY);
    Render.background(g2, background, width, height, Background.HILLS);
    Render.background(g2, background, width, height, Background.TREES);
        …
```

Dibujando la carretera

- Después del fondo, `render` itera sobre los segmentos:
- Proyecta los puntos `p1` y `p2` de cada segmento de coordenadas del mundo a coordenadas de la pantalla.
- Si es necesario, recorta el segmento, de lo contrario lo dibuja.

Dibujando la carretera

```
Segment baseSegment = findSegment(position);
int maxY = height;
Segment segment;
for (int n = 0; n < drawDistance; n++) {
    segment = segments.get((baseSegment.index + n) % segments.size());
    segment.looped = segment.index < baseSegment.index;
    segment.fog = Util.exponentialFog(n / drawDistance, fogDensity);
```

Dibujando la carretera

```
        Util.project(segment.p1, (playerX * roadWidth), cameraHeight,
            position - (segment.looped ? trackLength : 0), cameraDepth, width, height,
roadWidth);
        Util.project(segment.p2, (playerX * roadWidth), cameraHeight,
            position - (segment.looped ? trackLength : 0), cameraDepth, width, height,
roadWidth);
        if ((segment.p1.camera.z <= cameraDepth) ||     // behind us
            (segment.p2.screen.y >= maxY)) {   // clip by (already rendered) segment
          continue;
        }
```

Dibujando la carretera

```
    Render.segment(g2, width, lanes,
            segment.p1.screen.x, segment.p1.screen.y, segment.p1.screen.w,
            segment.p2.screen.x, segment.p2.screen.y, segment.p2.screen.w,
            segment.fog, segment.color);
    maxY = segment.p2.screen.y;
}
```

Método Util.project

- Para cada punto p1 y p2 calcula las coordenadas x y y en coordenadas de la pantalla.
- También calcula el ancho proyectado (w) del segmento y la escala (scale).

Método Util.project

```
public static void project(PointProjection p, double cameraX, double cameraY,
            double cameraZ, double cameraDepth, int width, int height, int roadWidth)
{
   p.camera.x = p.world.x - cameraX;  p.camera.y = p.world.y - cameraY;
   p.camera.z = p.world.z - cameraZ;
   p.screen.scale = cameraDepth / p.camera.z;
   p.screen.x = (int) Math.round((width / 2.0) + (p.screen.scale * p.camera.x * width /
2.0));
   p.screen.y = (int) Math.round((height / 2.0) - (p.screen.scale * p.camera.y * height /
2.0));
   p.screen.w = (int) Math.round((p.screen.scale * roadWidth * width / 2.0));
}
```

Dibujando la carretera

- Dadas las coordenadas x y y de la pantalla para p1 y p2, junto con el ancho de carretera proyectado w, el método Render.segment calcula los polígonos para dibujar el pasto, la carretera, las franjas de ruido y los separadores de carril usando el método Render.polygon.
- Ver los detalles en la clase Render.

Dibujando el automóvil

- Lo último que hace el método render es dibujar el automóvil.

```
Render.player(g2, width, height, resolution, roadWidth, sprites,
        speed / maxSpeed,
        cameraDepth / playerZ,
        width / 2.0,
        height,
        speed * (keyLeft ? -1 : keyRight ? 1 : 0),
        0);
```

Dibujando el automóvil

- El método Render.player utiliza el método drawImage de la clase Graphics2D para dibujar el sprite del automóvil.
- El sprite se escala usando la proyección vista anteriormente d / z.
- Donde z es la distancia del automóvil a la cámara guardada en la variable playerZ.
- También rebota el automóvil cuando va a altas velocidades agregando un número aleatorio basado en speed / maxSpeed.

Resultado final