B3: Curves and Sounds

CS1101S: Programming Methodology

Low Kok Lim & Martin Henz

August 27, 2021

Outline

- **Curves** (Kok-Lim)
- **Sounds** (Martin)

Outline

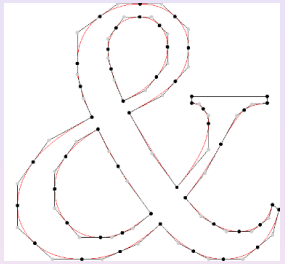
- **Curves** (Kok-Lim)
- **Sounds** (Martin)

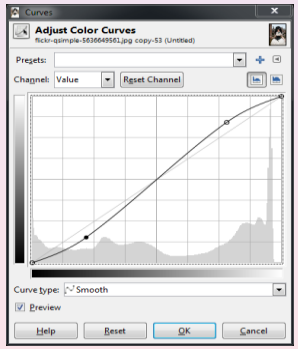
Missions & Quests on Curves

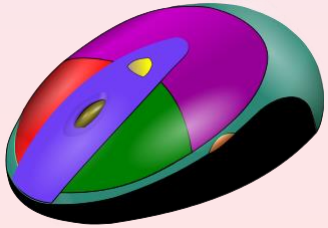
- **Mission: “Curve Introduction”**
- **Mission: “Curve Manipulation”**
- **Mission: “Beyond the First Dimension”**
- **Quest: “Cardioid Arrest”**
- **Quest: “Curvaceous Wizardry”**
- **Contest: “The Choreographer”**

Applications of Curves

- **Drawing** “vector-based”, “infinite resolution” smooth curves
- **Font** design, representation and rendering
- Smooth **animation paths** (for objects, light, camera)
- Designing **smooth functions**
  - E.g. for image color & tone adjustment
- **3D model design**, representation and rendering
  - For engineering/industrial designs, movie production and game development
- **Data fitting**
- etc.







Representation of Curves — **Explicit Form**

- **Curves in 2D**
  - The value of the **dependent variable** is given in terms of the **independent variable**
    - $y = f(x)$
    - **Example:** $y = mx + h$ (straight line)
  - Some curves cannot be expressed in explicit form
    - **Examples:** vertical straight line, circle

- **Curves in 3D**
  - Requires **two equations**
    - $y = f(x)$
    - $z = g(x)$

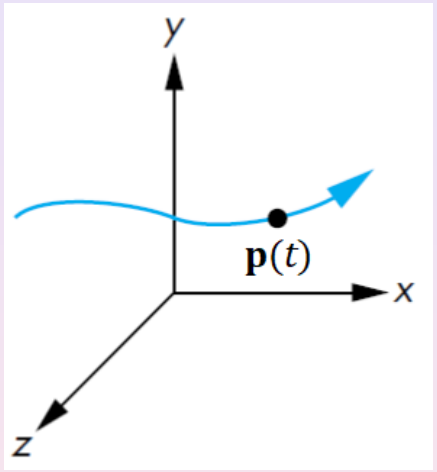
Representation of Curves — **Implicit Form**

- **Curves in 2D**
  - $f(x, y) = 0$
  - Examples:
    - $ax + by + c = 0$ (straight line)
    - $x^2 + y^2 - r^2 = 0$ (circle)
- **Curves in 3D**
  - Can be represented as the **intersection of two surfaces**
    - $f(x, y, z) = 0$
    - $g(x, y, z) = 0$
- A **drawback**: Difficult to obtain points on the curves
  - Because the equations are just membership tests

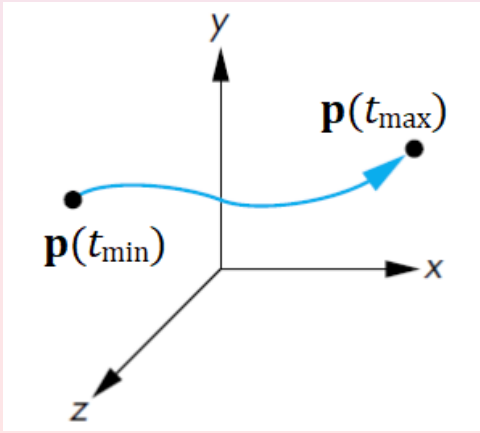
Representation of Curves — **Parametric Form**

- **Curves in 2D and 3D**
  - Each **spatial variable** for points on the curve is expressed in terms of an independent variable t, the **parameter**

$$\mathbf{p}(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} \qquad \mathbf{p}(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}$$



- A **curve segment** is defined for $t_{\min} \leq t \leq t_{\max}$
  - Often, $0 \leq t \leq 1$



Curves in Source

- Supported by the **`curve` module**
- Uses **parametric representation**
- Parameter ***t*** is within the **unit interval [0, 1]**
  - If ***C*** is a Curve, its
    - **starting** point is ***C***(0)
    - **ending** point is ***C***(1)

Specifications of Curves in Source

Curve

Curve := Number $\rightarrow$ Point

A Curve is a **function** that takes a **Number** as argument and returns a **Point**. The Number argument must be within the interval [0, 1].

Point

```
make_point : (Number, Number) → Point
x_of, y_of : Point → Number

x_of(make_point(n, m)) = n
y_of(make_point(n, m)) = m
```

Defining Curves

- **Examples:**

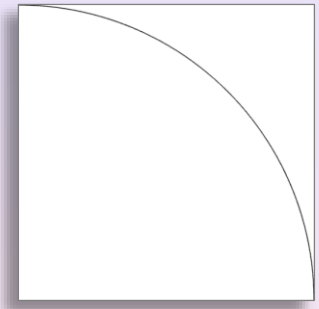
```
function unit_circle(t) {
    return make_point(math_cos(2 * math_PI * t),
                      math_sin(2 * math_PI * t));
}

function unit_line_at(y) {
    return t => make_point(t, y);
}

const unit_line = unit_line_at(0);
```

Show in Playground

Drawing Curves

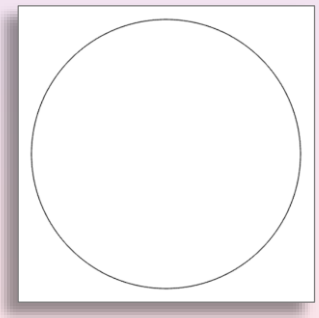
- **Examples:**

```
draw_connected(200)(unit_circle);
```

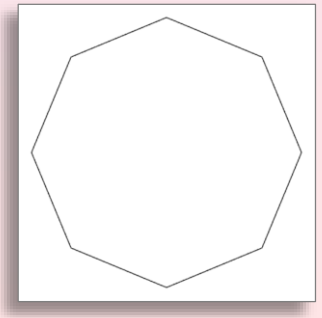


Show in Playground

```
draw_connected_full_view_proportional(200)
    (unit_circle);
```

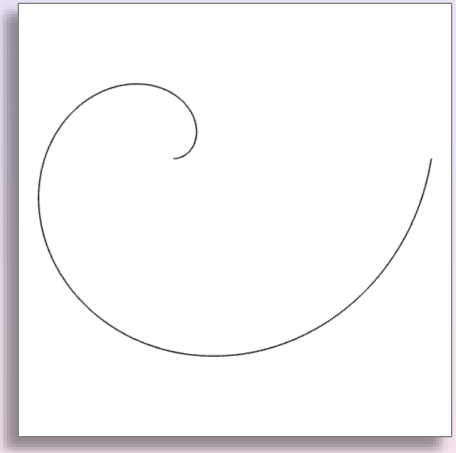


```
draw_connected_full_view_proportional(8)
    (unit_circle);
```



Example: Spiral Curves

- **Wanted:** Write a function `spiral_one` that represents a **spiral curve**
  - Uses `unit_circle`
  - One revolution only



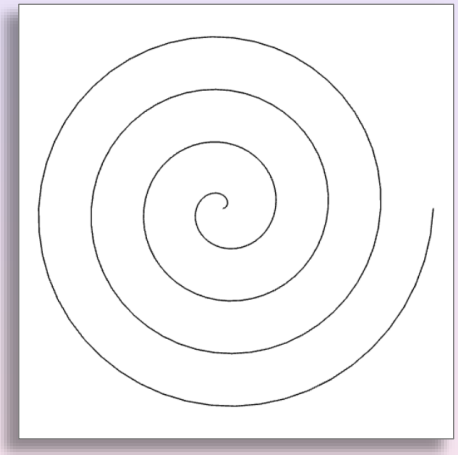
- **Solution:**

```
function spiral_one(t) {
    const p = unit_circle(t);
    return make_point(t * x_of(p), t * y_of(p));
}

draw_connected_full_view_proportional(200)(spiral_one);
```

Show in Playground

Example: Spiral Curves

- **Wanted:** Write a function `spiral` that returns a **spiral curve**
  - Uses `unit_circle`
  - Number of **revolutions** is a parameter



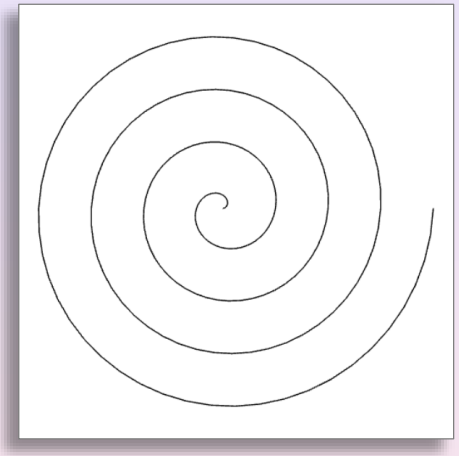
- **Attempt #1:**

```
function spiral(rev, t) {
    const p = unit_circle((t * rev) % 1);
    return make_point(t * x_of(p), t * y_of(p));
}

draw_connected_full_view_proportional(200)(spiral);
```

Show in Playground

Example: Spiral Curves

- **Wanted:** Write a function `spiral` that returns a **spiral curve**
  - Uses `unit_circle`
  - Number of **revolutions** is a parameter



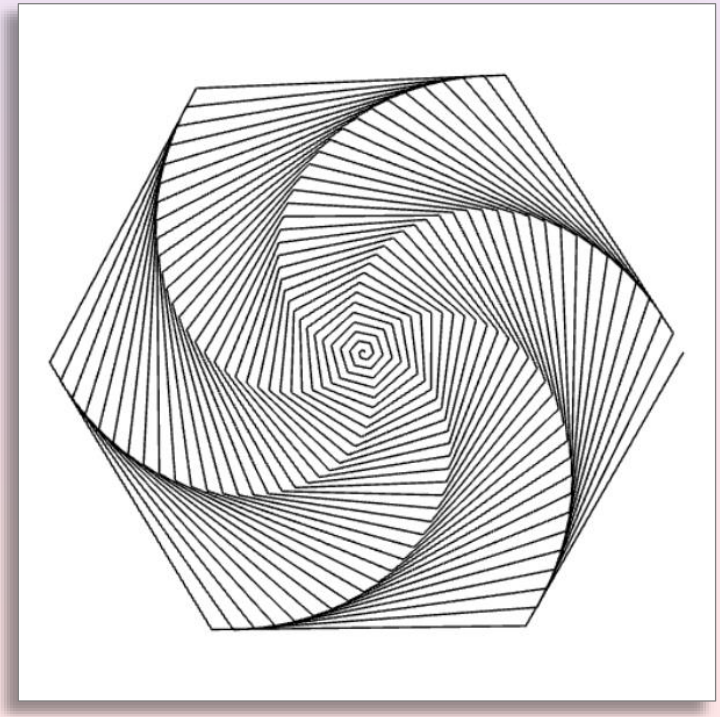
- **Attempt #2:**

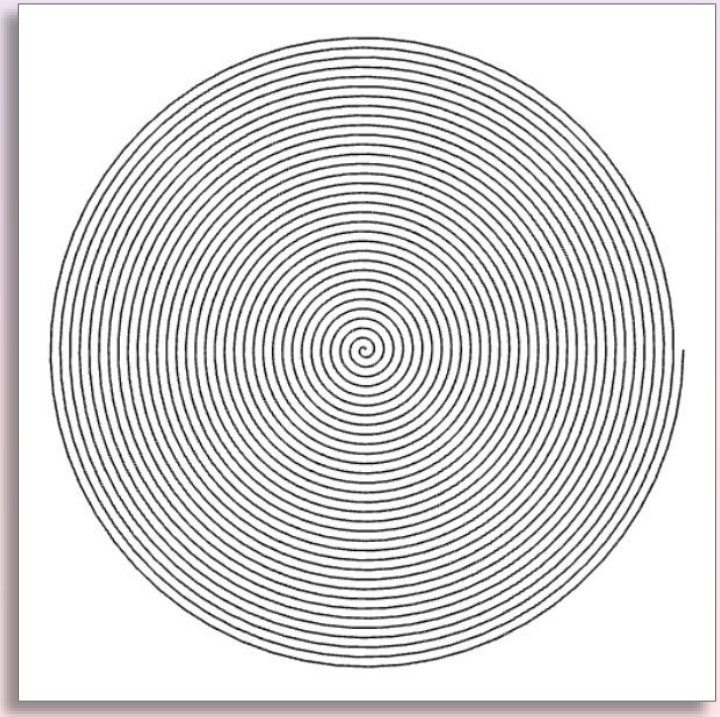
```
function spiral(rev) {
    return t => {
        const p = unit_circle((t * rev) % 1);
        return make_point(t * x_of(p), t * y_of(p));
    };
}
draw_connected_full_view_proportional(200)(spiral(4));
```

Show in Playground

Example: Spiral Curves

```
draw_connected_full_view_proportional(200)(spiral(33));

draw_connected_full_view_proportional(2000)(spiral(33));
```



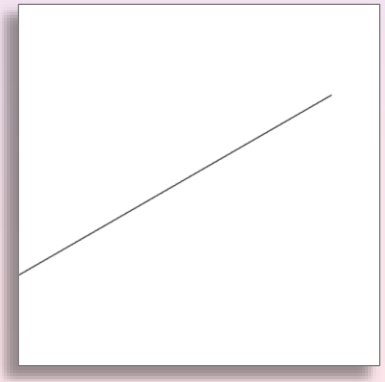


Show in Playground

Transformations on Curves

- **Example:**

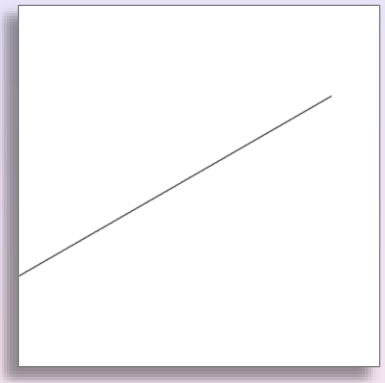
```
const rot_line =
    rotate_around_origin(0, 0, math_PI / 6)(unit_line);

const shifted_rot_line =
    translate(0, 0.25, 0)(rot_line);

draw_connected(200)(shifted_rot_line);
```



Show in Playground

Transformations on Curves

- **Example (alternative):**

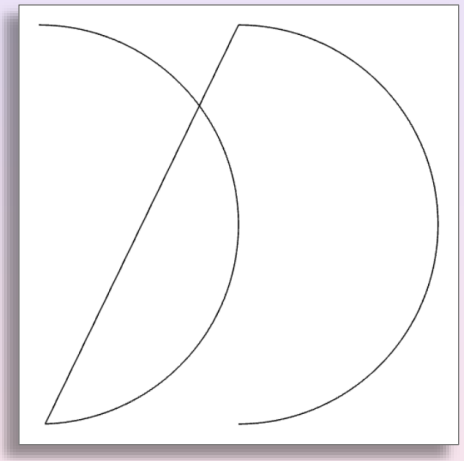


```
function compose(f, g) {
    return x => f(g(x));
}

const shift_rot =
    compose(translate(0, 0.25, 0),
            rotate_around_origin(0, 0, math_PI / 6));

const shifted_rot_line = shift_rot(unit_line);

draw_connected(200)(shifted_rot_line);
```

Show in Playground

Connecting Curves

- **Example:**



```
function connect_rigidly(curve1, curve2) {
    return t => t < 1/2
            ? curve1(2 * t)
            : curve2(2 * t - 1);
}

const result_curve =
    connect_rigidly(arc, translate(1, 0, 0)(arc));

draw_connected_full_view_proportional(200)
    (result_curve);
```

Show in Playground

Colored Curves

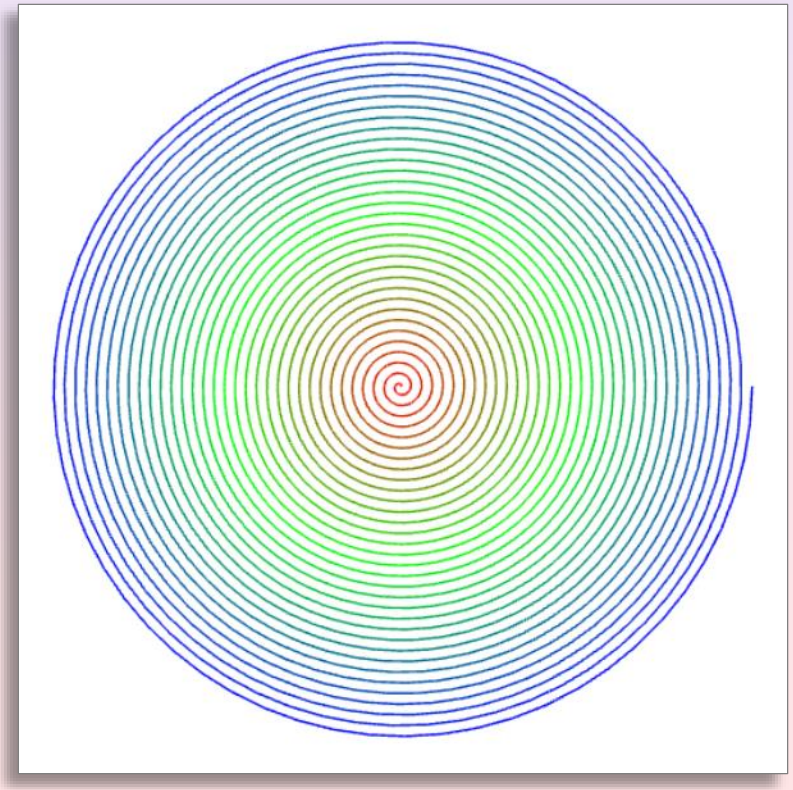
- **Example:**

```
function colorful_spiral(rev) {
    return t => {
        const p = unit_circle((t * rev) % 1);
        const R = math_max(0, 1 - 2 * t) * 255;
        const G = (1 - math_abs(1 - 2 * t)) * 255;
        const B = math_max(0, 2 * t - 1) * 255;
        return make_color_point(t * x_of(p), t * y_of(p),
                                R, G, B);
    };
}
draw_connected_full_view_proportional(2000)
    (colorful_spiral(33));
```

Show in Playground

Colored Curves

```
draw_connected_full_view_proportional(2000)
    (colorful_spiral(33));
```



Show in Playground

3D Curves

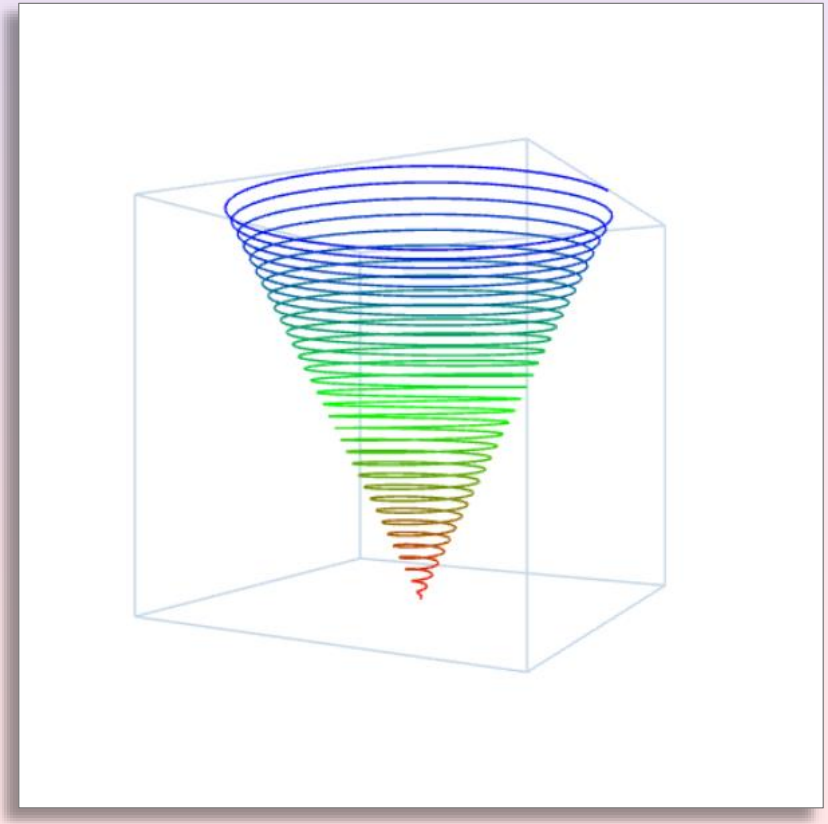
- **Example:**

```
function colorful_3D_spiral(rev) {
    return t => {
        const p = unit_circle((t * rev) % 1);
        const R = math_max(0, 1 - 2 * t) * 255;
        const G = (1 - math_abs(1 - 2 * t)) * 255;
        const B = math_max(0, 2 * t - 1) * 255;
        return make_3D_color_point(
            t * x_of(p), t * y_of(p), 2 * t, R, G, B);
    };
}
draw_3D_connected_full_view_proportional(2000)
    (colorful_3D_spiral(33));
```

Show in Playground

3D Curves

```
draw_3D_connected_full_view_proportional(2000)
    (colorful_3D_spiral(33));
```



Show in Playground

Summary

- **Functions** can provide **abstractions** for
  - compound operations
  - “**objects**” or **data**