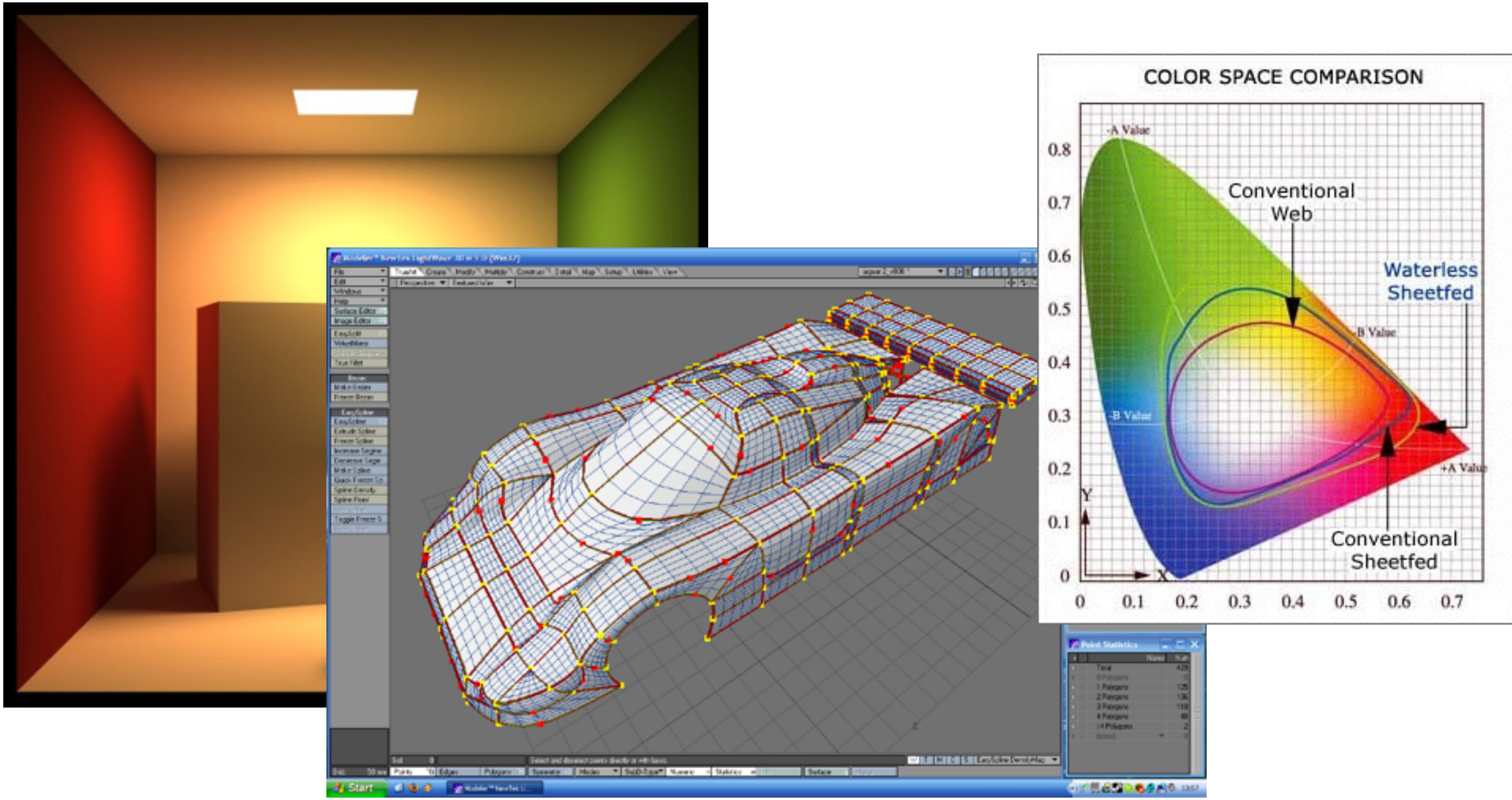


Lecture 8 : Introduction to curves (with a focus mostly on parametric representations, in 2D)

Tuesday October 5th 2021

Logistics

- Your homework #2 is due Thursday Oct 7th
- We're working on grading HW#1, grades *hopefully* will be posted by Thursday.
- There's a poll on Piazza (actually, two of them) regarding the time/format of the midterm. Please participate if you can.

Today's lecture

- An introduction to curves, and more specifically *parametric curves* in 2D
    - Motivation, uses and applications
    - Mathematical representations
    - Curve properties (tangent direction, arc length, etc)
    - Quick notes on animation
- We will closely follow the notation and exposition of Chapter 15 of Fundamentals of Computer Graphics (accessible via Canvas through this link)

What *are* curves? (for our purposes)

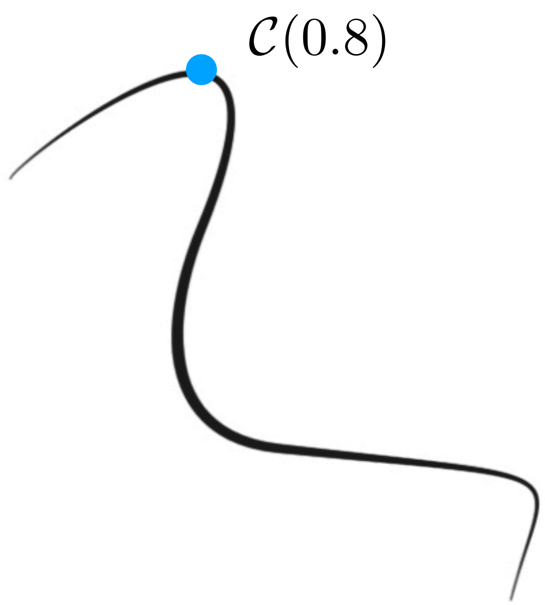
- Intuitively: the points traced by the tip of a drawing instrument when we move it on the 2D writing plane

Mathematically this is a <u>one dimensional object!</u>
Although for drawing purposes we might endow it with some thickness, curves are conceptually infinitesimally thin

Imagine what a curve would look like in 3D
The path of a 3D pen?
The shape of a bent wire?

What *are* curves? (for our purposes)

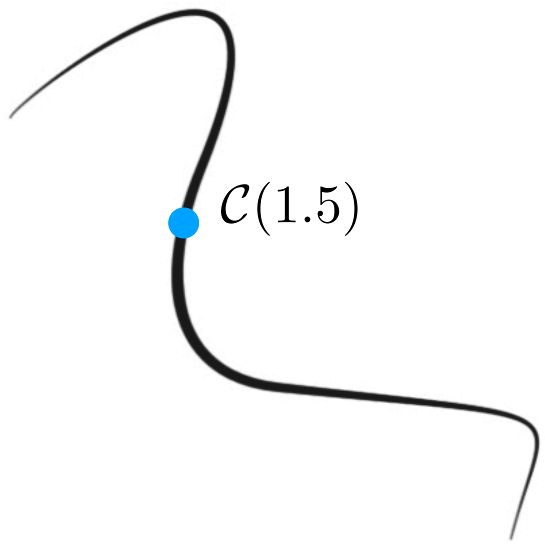
- Mathematically: a curve **may** be defined as the map of an interval into the 2D (or 3D) space
- To make this more concrete, consider the curve as a function $\mathcal{C}(t) : [a, b] \to \mathbf{R}^2$



*One possible intuitive interpretation: Consider "**t**" to be the time instance when the drawing instrument traces a given point of the curve*

What *are* curves? (for our purposes)

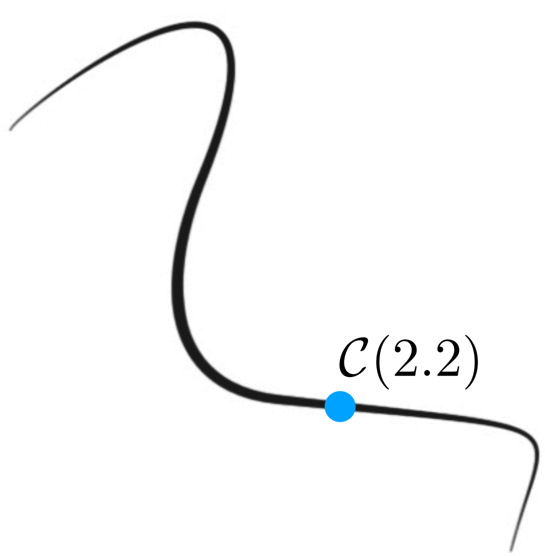
- Mathematically: a curve **may** be defined as the map of an interval into the 2D (or 3D) space
- To make this more concrete, consider the curve as a function $\mathcal{C}(t) : [a, b] \rightarrow \mathbf{R}^2$



*One possible intuitive interpretation: Consider "**t**" to be the time instance when the drawing instrument traces a given point of the curve*

What *are* curves? (for our purposes)

- Mathematically: a curve **may** be defined as the map of an interval into the 2D (or 3D) space
- To make this more concrete, consider the curve as a function $\mathcal{C}(t) : [a, b] \rightarrow \mathbf{R}^2$



*One possible intuitive interpretation: Consider "**t**" to be the time instance when the drawing instrument traces a given point of the curve*

Curve representations

- The representation of a curve as a function from an interval to the 2D (or 3D) space is what we call a *parametric* curve representation

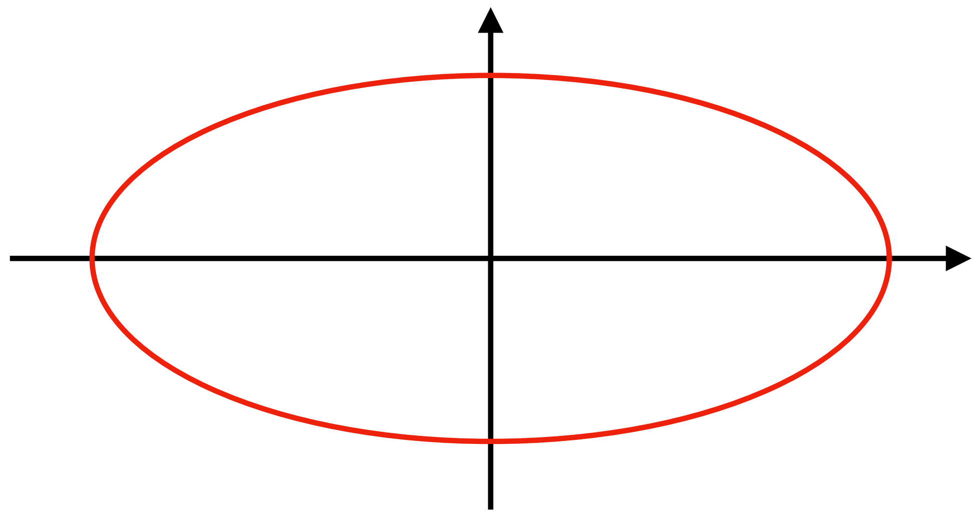
$$\mathcal{C}(t) : [a, b] \rightarrow \mathbf{R}^2$$

"t" is called the *parameter* of the curve (more on this later)

- However, this is not the only way we can (or choose) to represent curves …

Curve representations

- Alternative: An *implicit curve representation* $f(x, y) = 0$



Example: An ellipse

$$f(x, y) = \frac{x^2}{\alpha^2} + \frac{y^2}{\beta^2} - 1 = 0$$

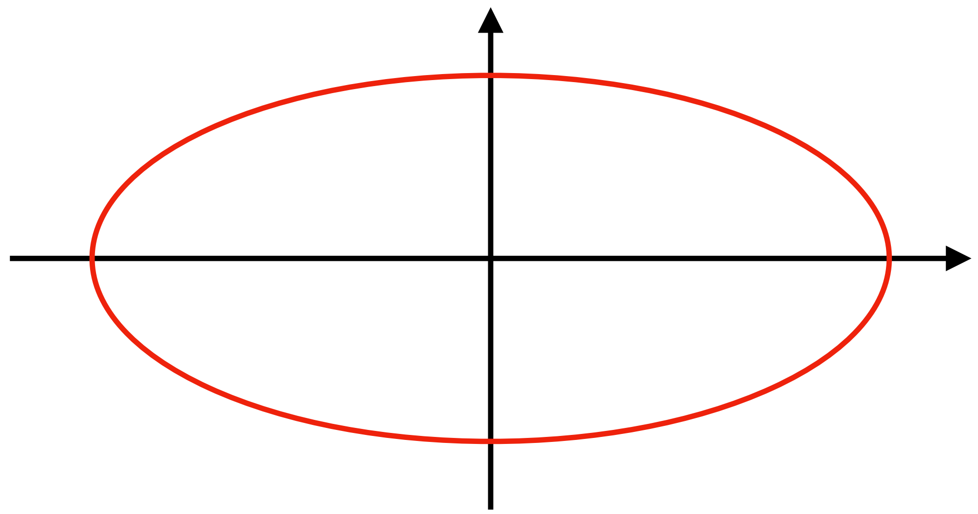
The implicit representation is the set of points (x,y) that satisfies the equation f(x,y)=0 (for an appropriate definition of f)

$$\mathcal{C}(t) = \begin{pmatrix} \alpha \cos(t) \\ \beta \sin(t) \end{pmatrix}, \quad t \in [0, 2\pi]$$

The very same curve can also admit an equivalent parametric representation, too!

Curve representations

- Benefits/disadvantages of implicit representations?



$$f(x,y) = \frac{x^2}{\alpha^2} + \frac{y^2}{\beta^2} - 1 = 0$$

+ Easy to check if a point is inside or outside (check the sign of f(x,y) …)

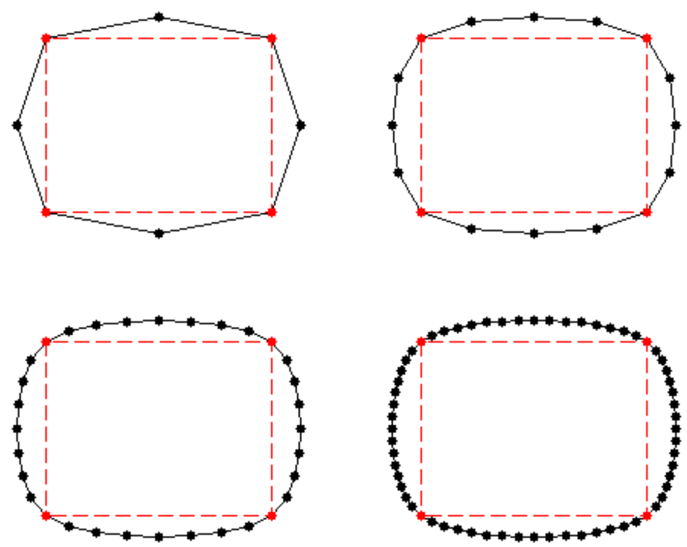
+ There are easy ways to compute geometric properties of interest (closest point on the curve, distance to the curve, etc; these are important, but outside the scope of CS559)

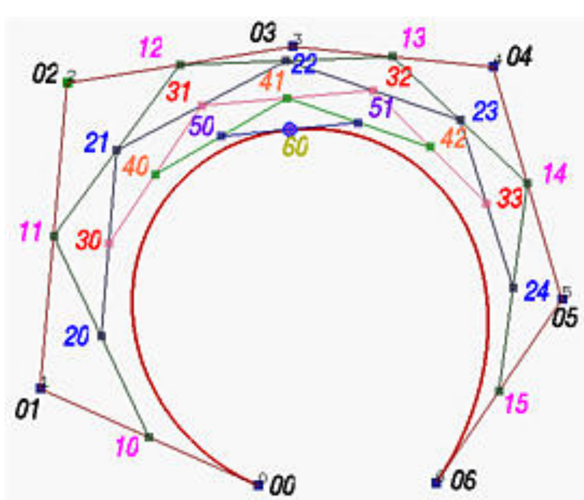
- Problematic if we want to represent <u>open</u> curves

- Hard to come up with an explicit formula for f(x,y) for all but very simple curves (but, we can construct <u>individual values</u> of a possible f(x,y); again, beyond our scope)

Curve representations

- Alternative: *Subdivision curves*





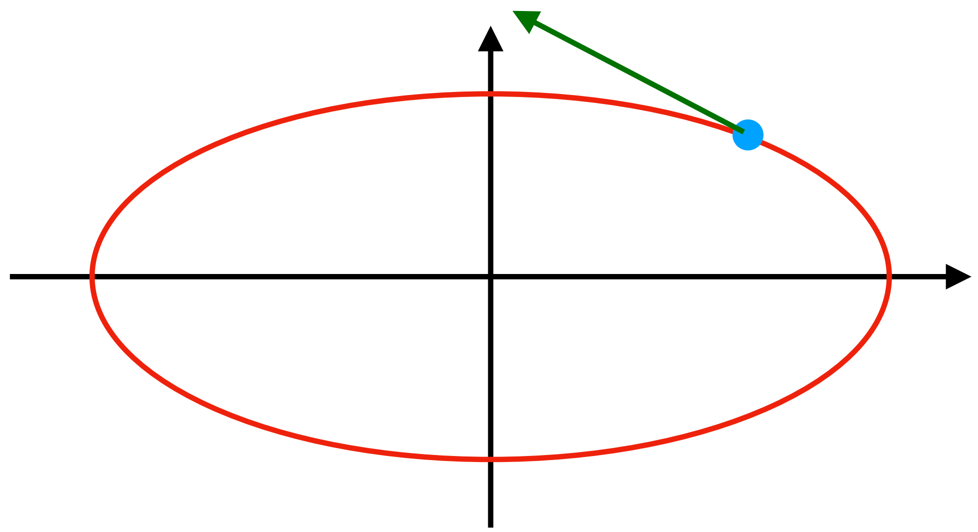
Subdivision curves start as a simple (but not very detailed) geometric object, such as a polygon or line-segment chain and provide a *rule for increasing the detail and complexity of this object.* *(i.e. adding more vertices and edges)*

The subdivision curve is the "limit" of Applying this detail-boosting rule infinite times.

We will talk about the process of subdivision, for select *parametric* curves, but this type of representation has much more depth and range of applications than we'll cover

Curve properties

- It is easy to compute the *tangent vector* of a parametric curve!



$$\mathcal{C}(t) = \left(\begin{array}{c} \alpha\cos(t) \\ \beta\sin(t) \end{array} \right), \quad t \in [0, 2\pi]$$

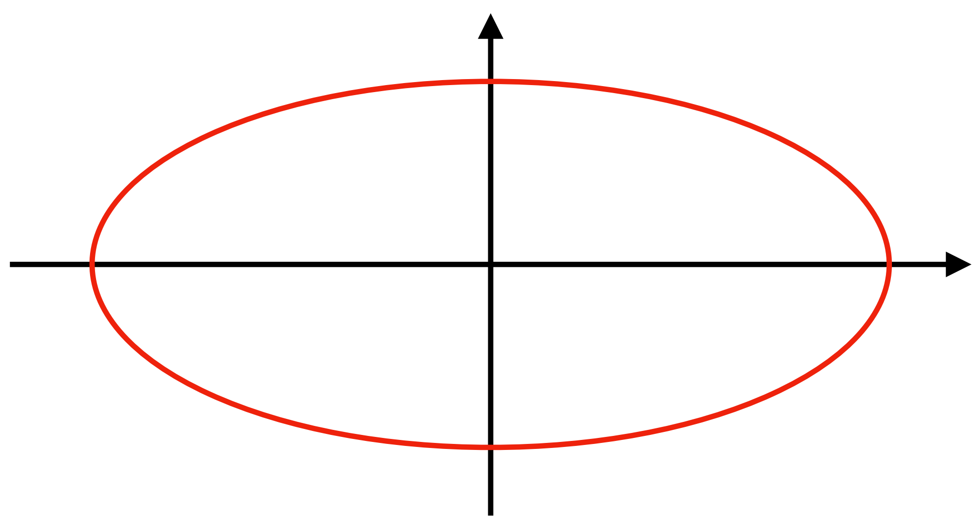
The <u>derivative</u> C'(t) is a parametric description of the <u>tangent vector!</u>

$$\mathcal{C}'(t) = \left(\begin{array}{c} -\alpha\sin(t) \\ \beta\cos(t) \end{array} \right), \quad t \in [0, 2\pi]$$

Intuitively: C'(t) is the <u>velocity vector</u> for the tip of our drawing instrument (if "t" is interpreted as the time when our pen goes over each point) The <u>speed</u> (length drawn per unit time) is the magnitude |C'(t)|

Curve properties

- It is easy to compute the *length* of a parametric curve



$$\mathcal{C}(t) = \begin{pmatrix} \alpha\cos(t) \\ \beta\sin(t) \end{pmatrix}, \quad t \in [0, 2\pi]$$

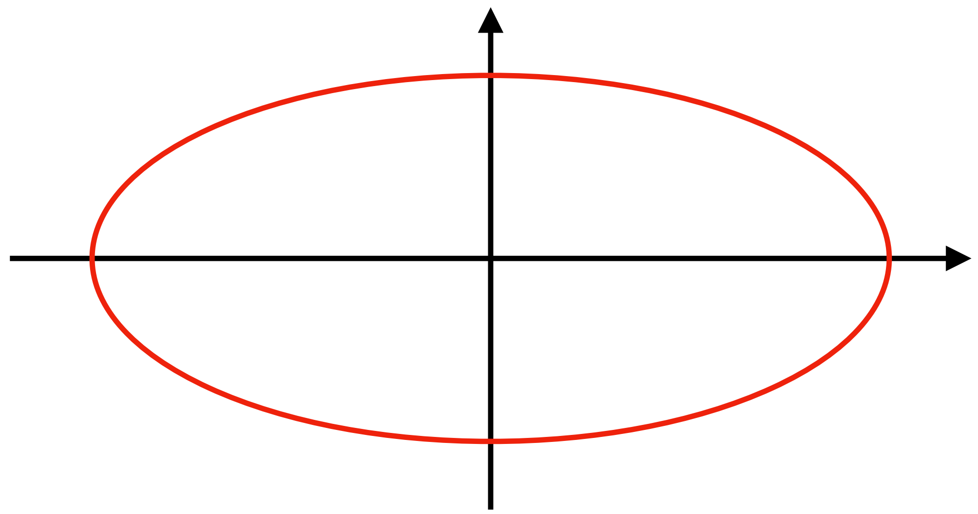
The length is obtained by integrating the speed over the parameter integral

$$L = \int_0^{2\pi} |\mathcal{C}'(t)| dt = \int_0^{2\pi} \sqrt{\alpha^2 \sin^2(t) + \beta^2 \cos^2(t)} dt$$

(exercise: check that for a circle with radius R, L=2πR)

Parameterization (and re-parameterization)

- For *parametric curves*, the choice of the parameter is not unique!



$$\mathcal{C}(t) = \begin{pmatrix} \alpha\cos(t) \\ \beta\sin(t) \end{pmatrix}, \quad t \in [0, 2\pi]$$

$$\mathcal{C}(t) = \begin{pmatrix} \alpha\cos(2t^2) \\ \beta\sin(2t^2) \end{pmatrix}, \quad t \in [0, \sqrt{\pi}]$$

$$\mathcal{C}(t) = \begin{pmatrix} \alpha\cos(2\pi t) \\ \beta\sin(2\pi t) \end{pmatrix}, \quad t \in [0, 1]$$

All these parametric representations trace out exactly the same curve!

Difference: the different formulas trace out the curve <u>at different speed!</u>

Parameterization (and re-parameterization)

- For *parametric curves*, the choice of the parameter is not unique!

There is a special parameterization called arc-length parameterization that satisfies the property |C'(t)|=1

For arc-length parameterized curves, the parameter can be interpreted to be the length of the curve traversed up to the current point

We can convert to arc-length parameterization via change of variables

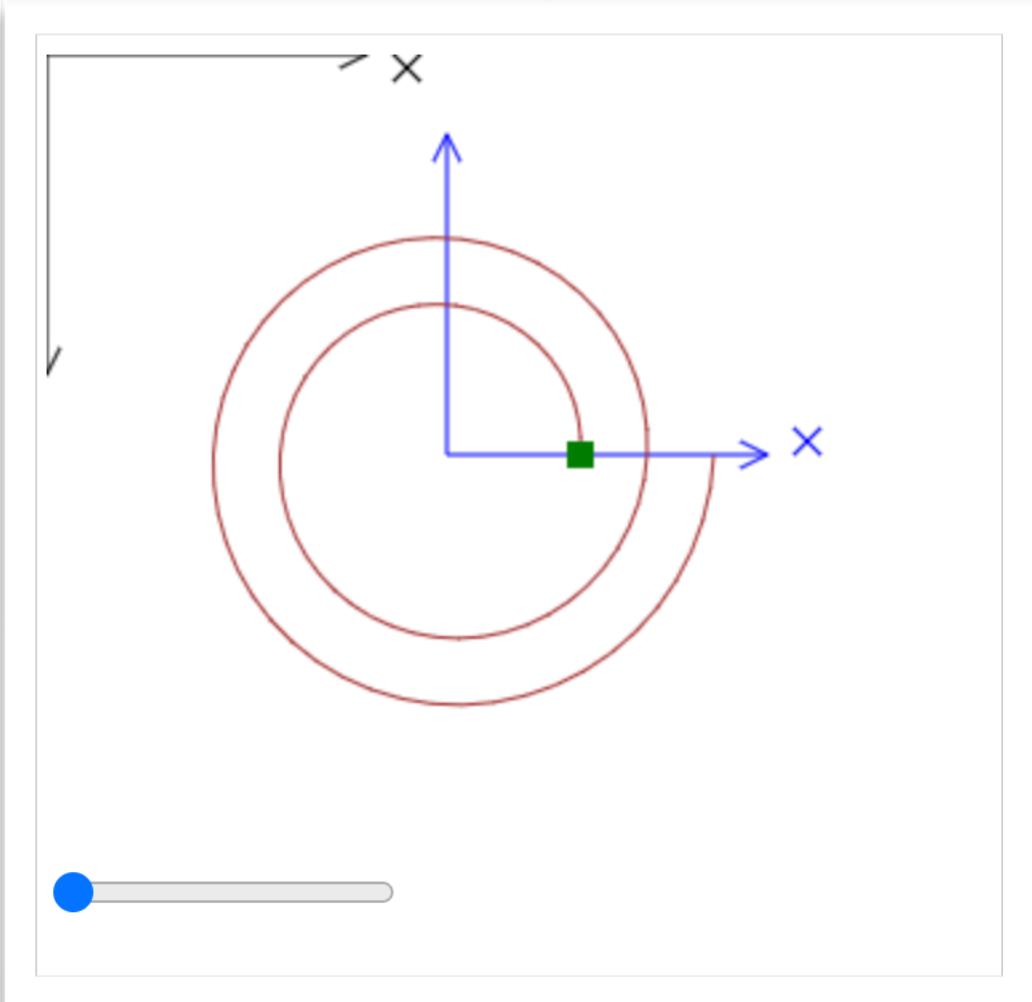
$$s = \int_0^t |\mathcal{C}'(\tau)| d\tau$$

(we'll see an example once we go to 3D …)

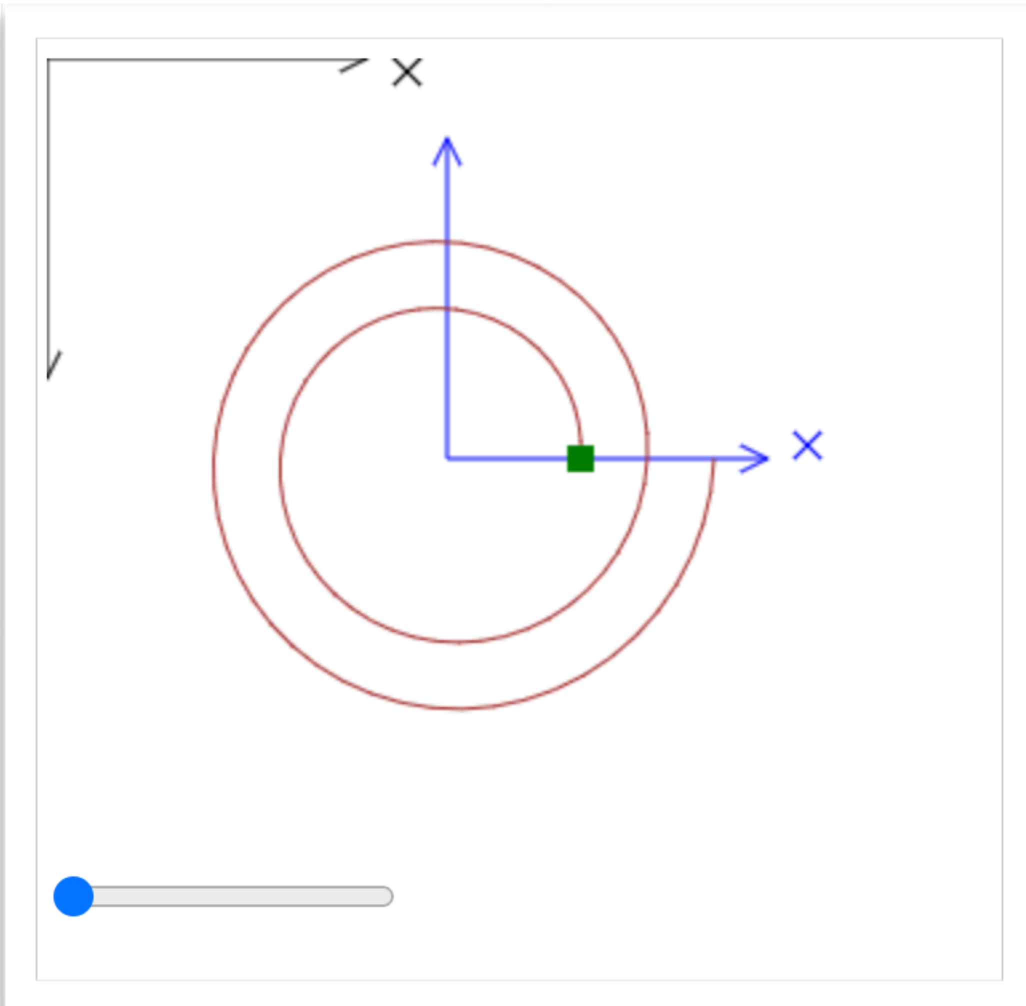
Why parametric curves?

- Creating more complex shapes than built-in objects
- Creating animations
  (by having objects "trace" the curve)
- Using derived properties (tangents/normals) that facilitate the creation of more complex scenes or appearances

Implementation example : A spiral curve



Implementation example : A spiral curve



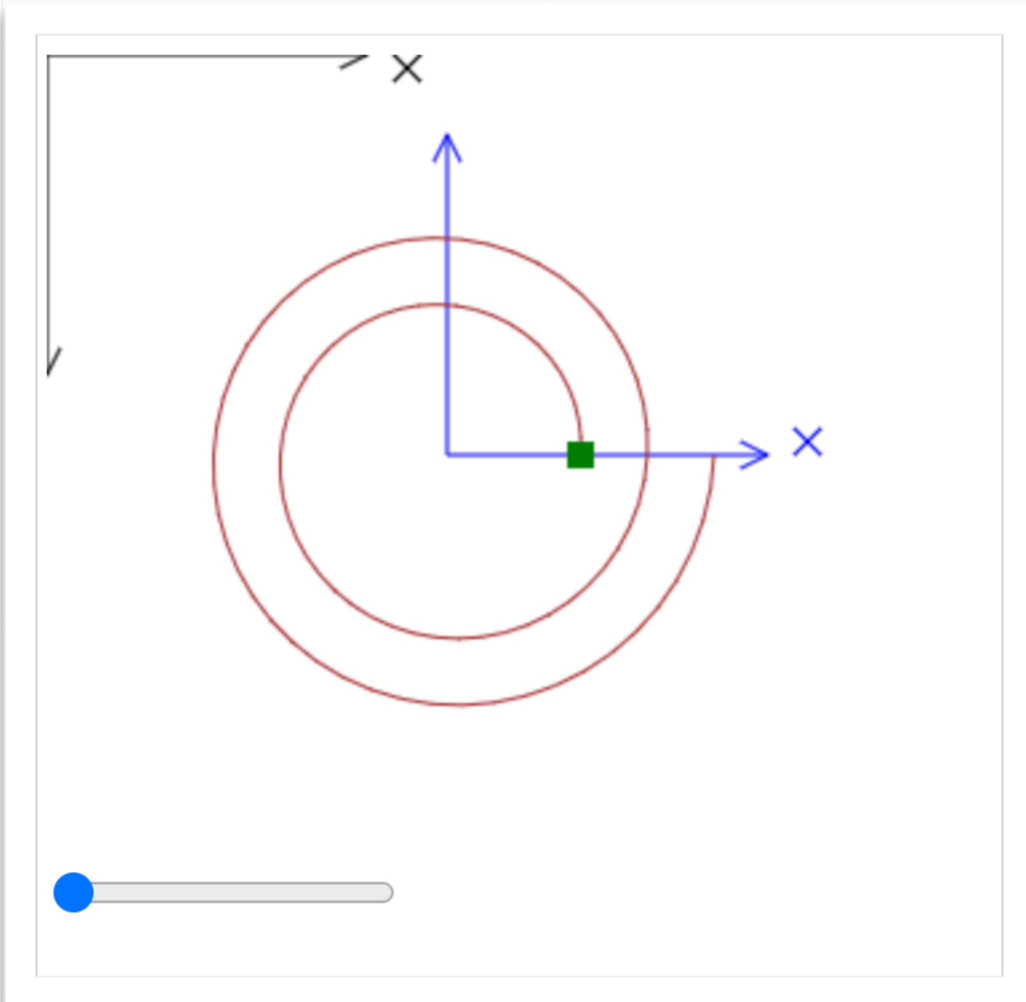
Curve

$$\mathcal{C}(t) = \begin{pmatrix} R(t)\cos(2\pi t) \\ R(t)\sin(2\pi t) \end{pmatrix}, \quad t \in [0,2]$$

$$R(t) = \alpha t + \beta$$

(all expressions relative to the <u>blue</u> system!)

Implementation example : A spiral curve



Curve

$$\mathcal{C}(t) = \begin{pmatrix} R(t)\cos(2\pi t) \\ R(t)\sin(2\pi t) \end{pmatrix}, \quad t \in [0,2]$$

$$R(t) = \alpha t + \beta$$

Tangent

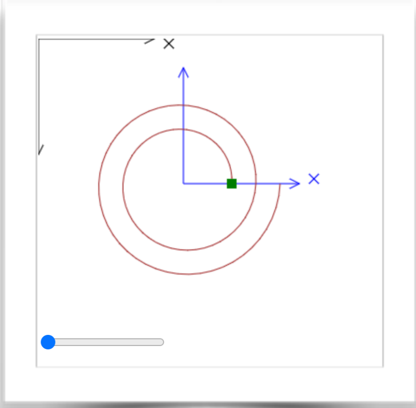
$$\mathcal{C}'(t) = \begin{pmatrix} R'(t)\cos(2\pi t) - R(t)2\pi\sin(2\pi t) \\ R'(t)\sin(2\pi t) + R(t)2\pi\cos(2\pi t) \end{pmatrix}$$

(all expressions relative to the blue system!)

Implementation example : A spiral curve

jsbin.com/ziletiq

Week5/Demo0



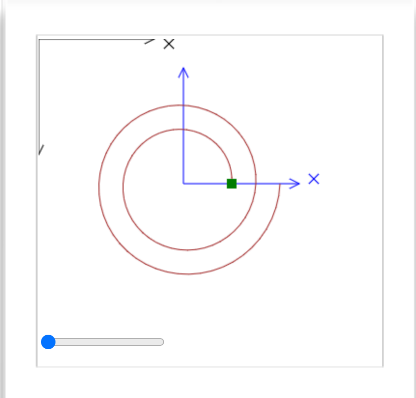
JavaScript

```
function setup() {
    var canvas = document.getElementById('myCanvas');
    var context = canvas.getContext('2d');
    var slider1 = document.getElementById('slider1');
    slider1.value = -25;

    function draw() {
    canvas.width = canvas.width;
    // use the sliders to get the angles
    var tParam = slider1.value*0.01;

    function moveToTx(loc,Tx)
    {var res=vec2.create(); vec2.transformMat3(res,loc,Tx); context.moveTo(res[0],res[1]);}

    function lineToTx(loc,Tx)
    {var res=vec2.create(); vec2.transformMat3(res,loc,Tx); context.lineTo(res[0],res[1]);}

    function drawObject(color,Tx) {
        context.beginPath();
        context.fillStyle = color;
        moveToTx([-5,-5],Tx);
        lineToTx([-5,5],Tx);
        lineToTx([5,5],Tx);
        lineToTx([5,-5],Tx);
        context.closePath();
        context.fill();
    }

[...]
```

Implementation example : A spiral curve

jsbin.com/ziletiq

Week5/Demo0



JavaScript

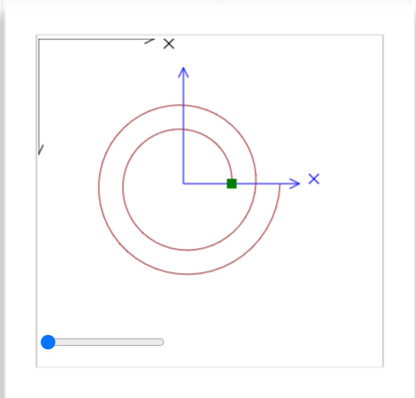
```
function setup() {
    var canvas = document.getElementById('myCanvas');
    var context = canvas.getContext('2d');
    var slider1 = document.getElementById('slider1');
    slider1.value = -25;

    function draw() {
    canvas.width = canvas.width;
    // use the sliders to get the angles
    var tParam = slider1.value*0.01;

    function moveToTx(loc,Tx)
    {var res=vec2.create(); vec2.transformMat3(res,loc,Tx); context.moveTo(res[0],res[1]);}

    function lineToTx(loc,Tx)
    {var res=vec2.create(); vec2.transformMat3(res,loc,Tx); context.lineTo(res[0],res[1]);}

    function drawObject(color,Tx) {
        context.beginPath();
        context.fillStyle = color;
        moveToTx([-5,-5],Tx);
        lineToTx([-5,5],Tx);
        lineToTx([5,5],Tx);
        lineToTx([5,-5],Tx);
        context.closePath();
        context.fill();
    }

[...]
```

Passing "vectors" to the drawing calls

Implementation example : A spiral curve

jsbin.com/ziletiq

Week5/Demo0



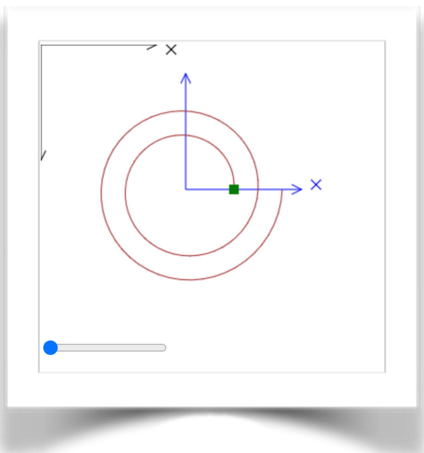
JavaScript

```
[...]
    function drawAxes(color,Tx) {
        context.strokeStyle=color;
        context.beginPath();
        // Axes
        moveToTx([120,0],Tx);lineToTx([0,0],Tx);lineToTx([0,120],Tx);
        // Arrowheads
        moveToTx([110,5],Tx);lineToTx([120,0],Tx);lineToTx([110,-5],Tx);
        moveToTx([5,110],Tx);lineToTx([0,120],Tx);lineToTx([-5,110],Tx);
        // X-label
        moveToTx([130,0],Tx);lineToTx([140,10],Tx);
        moveToTx([130,10],Tx);lineToTx([140,0],Tx);
        context.stroke();
    }

    var Rstart = 50.0;
    var Rslope = 25.0;
    var Cspiral = function(t) {
        var R = Rslope * t + Rstart;
        var x = R * Math.cos(2.0 * Math.PI * t);
        var y = R * Math.sin(2.0 * Math.PI * t);
        return [x,y];
    }

    function drawTrajectory(t_begin,t_end,intervals,C,Tx,color) {
        context.strokeStyle=color;
        context.beginPath();
            moveToTx(C(t_begin),Tx);
            for(var i=1;i<=intervals;i++){
        var t=((intervals-i)/intervals)*t_begin+(i/intervals)*t_end;
        lineToTx(C(t),Tx);
            }
            context.stroke();
    }

[...]
```

Implementation example : A spiral curve

jsbin.com/ziletiq

Week5/Demo0



$$\mathcal{C}(t) = \begin{pmatrix} R(t)\cos(2\pi t) \\ R(t)\sin(2\pi t) \end{pmatrix}, \quad t \in [0,2]$$

$$R(t) = \alpha t + \beta$$

JavaScript

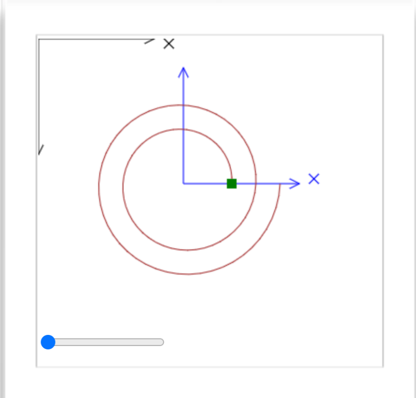
```
[...]
    function drawAxes(color,Tx) {
        context.strokeStyle=color;
        context.beginPath();
        // Axes
        moveToTx([120,0],Tx);lineToTx([0,0],Tx);lineToTx([0,120],Tx);
        // Arrowheads
        moveToTx([110,5],Tx);lineToTx([120,0],Tx);lineToTx([110,-5],Tx);
        moveToTx([5,110],Tx);lineToTx([0,120],Tx);lineToTx([-5,110],Tx);
        // X-label
        moveToTx([130,0],Tx);lineToTx([140,10],Tx);
        moveToTx([130,10],Tx);lineToTx([140,0],Tx);
        context.stroke();
    }

    var Rstart = 50.0;
    var Rslope = 25.0;
    var Cspiral = function(t) {
        var R = Rslope * t + Rstart;
        var x = R * Math.cos(2.0 * Math.PI * t);
        var y = R * Math.sin(2.0 * Math.PI * t);
        return [x,y];
    }

    function drawTrajectory(t_begin,t_end,intervals,C,Tx,color) {
        context.strokeStyle=color;
        context.beginPath();
            moveToTx(C(t_begin),Tx);
            for(var i=1;i<=intervals;i++){
        var t=((intervals-i)/intervals)*t_begin+(i/intervals)*t_end;
        lineToTx(C(t),Tx);
            }
            context.stroke();
    }

[...]
```

Implementation of the parametric curve

Implementation example : A spiral curve

jsbin.com/ziletiq

Week5/Demo0



JavaScript

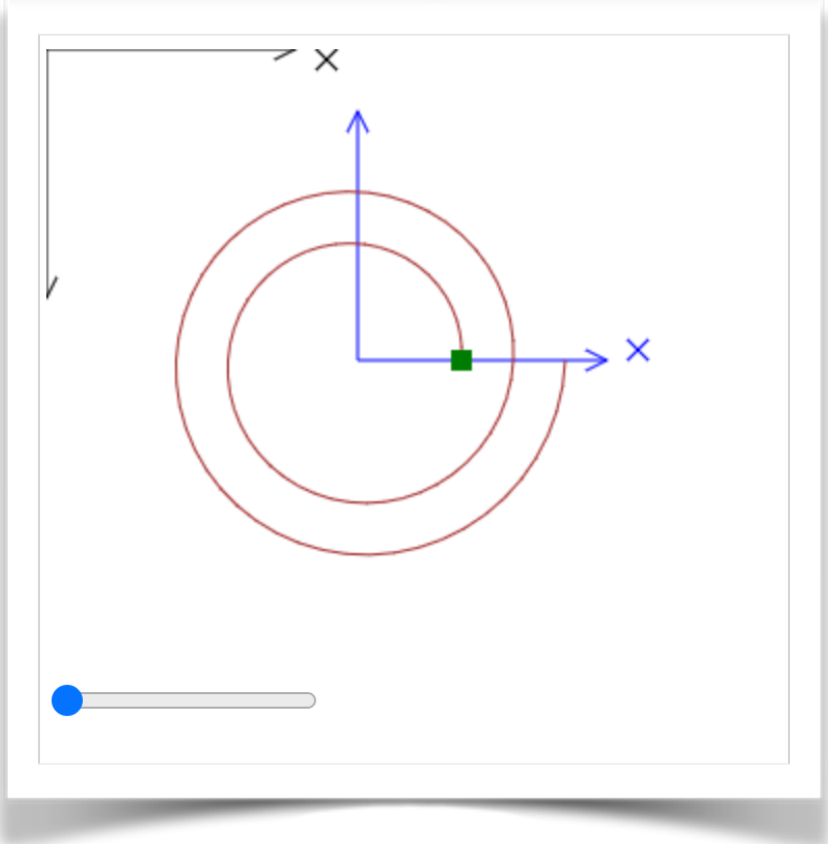
```
[...]
    function drawAxes(color,Tx) {
        context.strokeStyle=color;
        context.beginPath();
        // Axes
        moveToTx([120,0],Tx);lineToTx([0,0],Tx);lineToTx([0,120],Tx);
        // Arrowheads
        moveToTx([110,5],Tx);lineToTx([120,0],Tx);lineToTx([110,-5],Tx);
        moveToTx([5,110],Tx);lineToTx([0,120],Tx);lineToTx([-5,110],Tx);
        // X-label
        moveToTx([130,0],Tx);lineToTx([140,10],Tx);
        moveToTx([130,10],Tx);lineToTx([140,0],Tx);
        context.stroke();
    }

    var Rstart = 50.0;
    var Rslope = 25.0;
    var Cspiral = function(t) {
        var R = Rslope * t + Rstart;
        var x = R * Math.cos(2.0 * Math.PI * t);
        var y = R * Math.sin(2.0 * Math.PI * t);
        return [x,y];
    }

    function drawTrajectory(t_begin,t_end,intervals,C,Tx,color) {
        context.strokeStyle=color;
        context.beginPath();
            moveToTx(C(t_begin),Tx);
            for(var i=1;i<=intervals;i++){
        var t=((intervals-i)/intervals)*t_begin+(i/intervals)*t_end;
        lineToTx(C(t),Tx);
            }
            context.stroke();
    }

[...]
```

Drawing curve as a sequence of (small) line segments!

Implementation example : A spiral curve

jsbin.com/ziletiq

Week5/Demo0



JavaScript

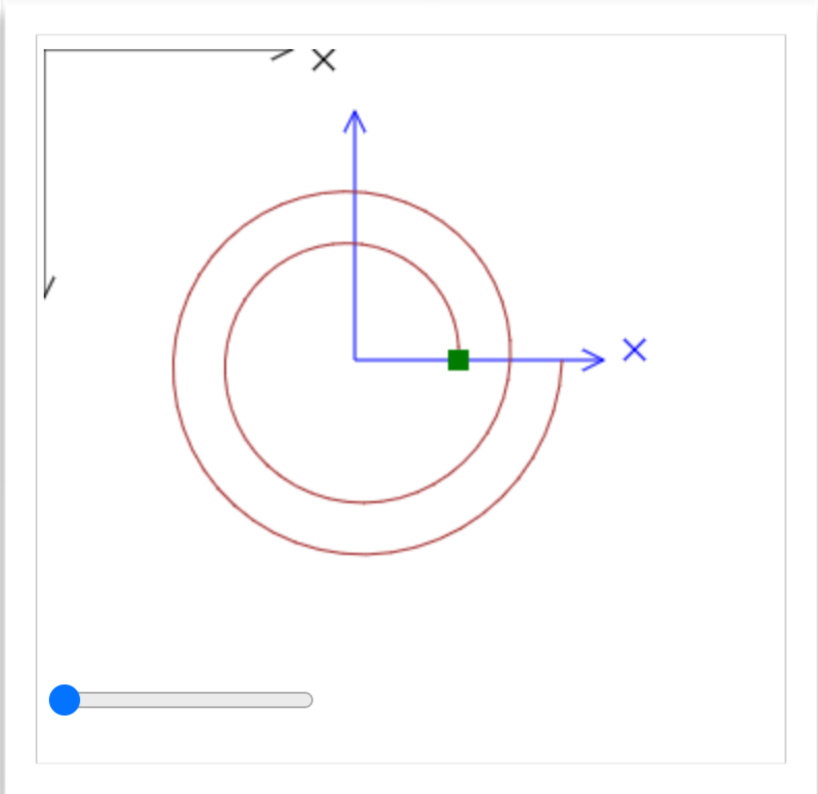
```
[...]
    function drawTrajectory(t_begin,t_end,intervals,C,Tx,color) {
        context.strokeStyle=color;
        context.beginPath();
            moveToTx(C(t_begin),Tx);
            for(var i=1;i<=intervals;i++){
        var t=((intervals-i)/intervals)*t_begin+(i/intervals)*t_end;
        lineToTx(C(t),Tx);
            }
            context.stroke();
    }

    // make sure you understand these

    drawAxes("black", mat3.create());

    var Tblue_to_canvas = mat3.create();
    mat3.fromTranslation(Tblue_to_canvas,[150,150]);
    mat3.scale(Tblue_to_canvas,Tblue_to_canvas,[1,-1]); // Flip the Y-axis
    drawAxes("blue",Tblue_to_canvas);

    drawTrajectory(0.0,2.0,100,Cspiral,Tblue_to_canvas,"brown");
    var Tgreen_to_blue = mat3.create();
    mat3.fromTranslation(Tgreen_to_blue,Cspiral(tParam));
    var Tgreen_to_canvas = mat3.create();
    mat3.multiply(Tgreen_to_canvas, Tblue_to_canvas, Tgreen_to_blue);
    drawObject("green",Tgreen_to_canvas);
    }

    slider1.addEventListener("input",draw);
    draw();
```

Using transforms to draw relatively to a better "centered" system

Implementation example : A spiral curve

jsbin.com/ziletiq

Week5/Demo0



JavaScript

```
[...]
    function drawTrajectory(t_begin,t_end,intervals,C,Tx,color) {
        context.strokeStyle=color;
        context.beginPath();
            moveToTx(C(t_begin),Tx);
            for(var i=1;i<=intervals;i++){
        var t=((intervals-i)/intervals)*t_begin+(i/intervals)*t_end;
        lineToTx(C(t),Tx);
            }
            context.stroke();
    }

    // make sure you understand these

    drawAxes("black", mat3.create());

    var Tblue_to_canvas = mat3.create();
    mat3.fromTranslation(Tblue_to_canvas,[150,150]);
    mat3.scale(Tblue_to_canvas,Tblue_to_canvas,[1,-1]); // Flip the Y-axis
    drawAxes("blue",Tblue_to_canvas);

    drawTrajectory(0.0,2.0,100,Cspiral,Tblue_to_canvas,"brown");
    var Tgreen_to_blue = mat3.create();
    mat3.fromTranslation(Tgreen_to_blue,Cspiral(tParam));
    var Tgreen_to_canvas = mat3.create();
    mat3.multiply(Tgreen_to_canvas, Tblue_to_canvas, Tgreen_to_blue);
    drawObject("green",Tgreen_to_canvas);
    }

    slider1.addEventListener("input",draw);
    draw();
```

Draw extent of the curve, and animate an object translating along it