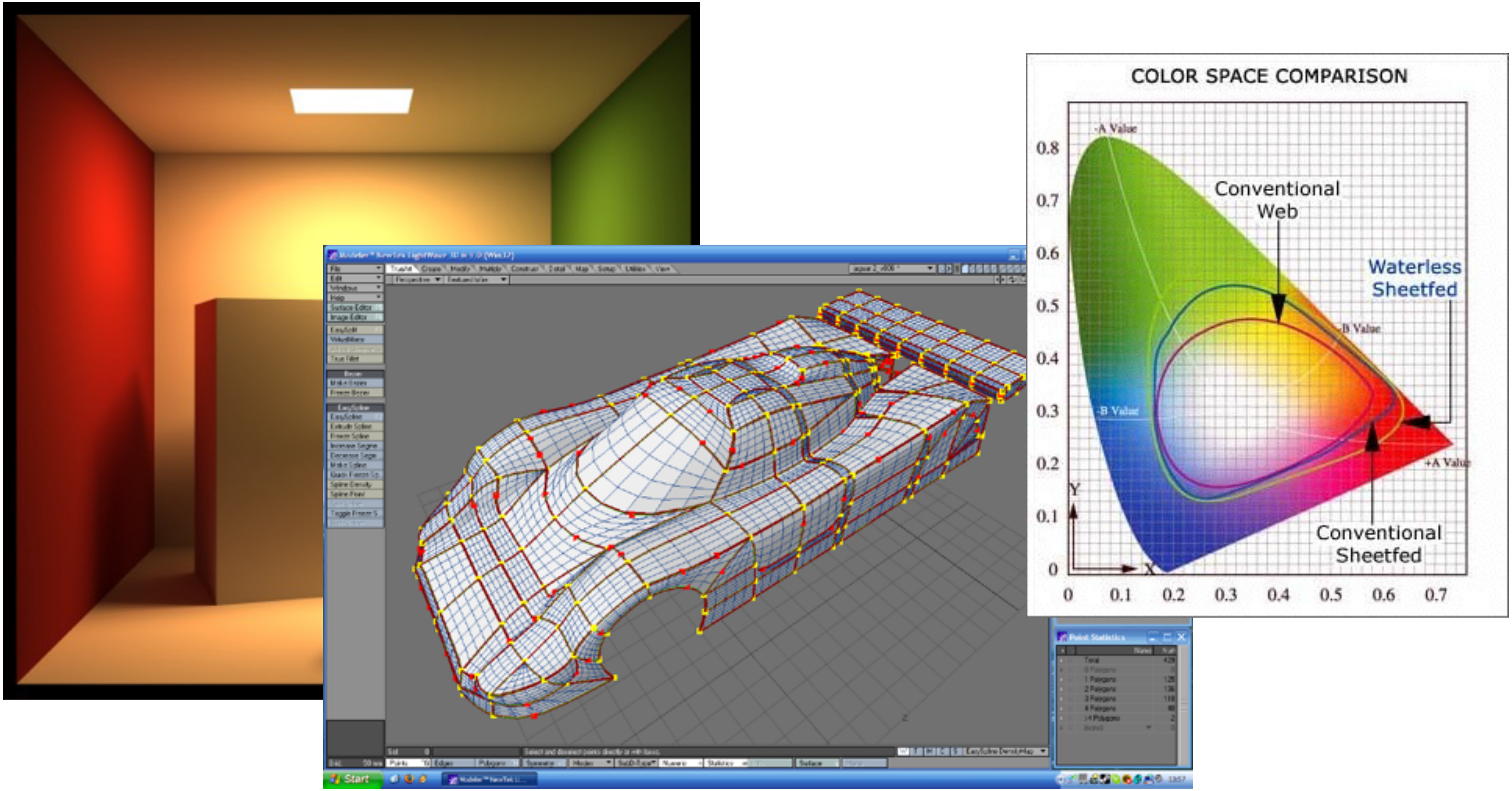


Lecture 11 : More parametric cubics (implementation, basis functions, non-interpolating curves, subdivision)

Thursday October 14th 2020

Logistics

- Programming Assignment #3 has been released as of yesterday - it is due on Friday Oct 22nd
- Regrade requests
  - No time constraint to raise an issue (we can revisit grading issues at any time)
  - Private posting on Piazza probably best way to "file" a regrade request; during office hours, Carter Sifferman is the best-positioned member of the instructional staff to discuss grading with you.

Today's lecture

- More on cubics and parameterization
    - Practical implementation of Hermite curves (in code!)
    - Basis functions
    - Other interpolating curves: Bezier (in detail), Natural Cubics (brief exposition)
    - Approximating (non-interpolating) curves; B-Splines
    - Subdivision

(Recap) Cubic Hermite curves

jsbin.com/fawudah

Week6/Demo0

- The first example of a *cubic* parametric curve

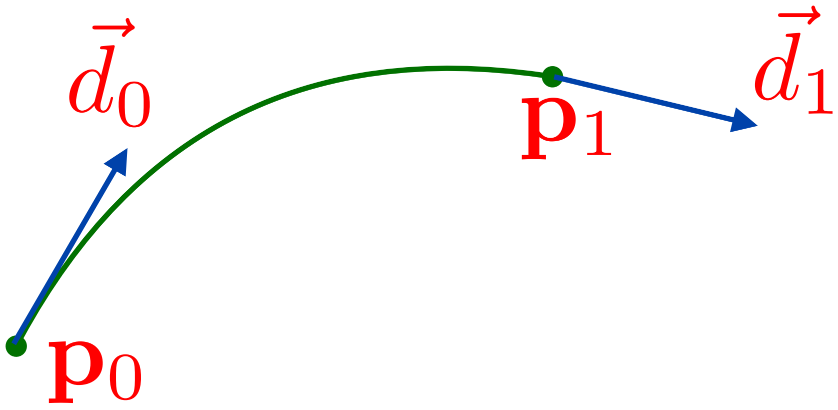
$$\mathbf{f}(u) = \underbrace{\begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix}}_{\mathbf{u}} \underbrace{\begin{bmatrix} a_0 & b_0 \\ a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \end{bmatrix}}_{\mathbf{A}}$$

- Can be used to control both location of endpoints, as well as the direction of the tangent
- Can be easily manipulated to build piecewise-Hermit curves with C1 continuity
- Allows *local control* in conjunction with C1 continuity

Hermite cubics

jsbin.com/fawudah

Week6/Demo0



- We specify
  - Beginning and ending *positions* $\boldsymbol{p_0}, \boldsymbol{p_1}$
  - Beginning and ending *tangents* $\boldsymbol{d_0}, \boldsymbol{d_1}$
- As before the curve is written (using the *basis matrix*)

$$\mathbf{f}(u) = \mathbf{uBP}$$

Correction from Tuesday's slides!

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3 & -2 & 3 & -1 \\ 2 & 1 & -2 & 1 \end{bmatrix} \qquad \mathbf{P} = \begin{bmatrix} p_0 \\ d_0 \\ p_1 \\ d_1 \end{bmatrix}$$

Basis functions?

- Standard parametric form : $\mathbf{f}(u) = \mathbf{uBP}$
- We can multiply u and B to get a *vector of basis functions* $\mathbf{b}(u) = \mathbf{uB} = \left[\begin{array}{cccc} b_0(u) & b_1(u) & b_2(u) & b_3(u) \end{array}\right]$
- The rows of the matrix **P** contain *"control points"* (they might actually be other than "points", e.g. tangent vectors, but we call all such points or vectors "control points" for uniformity) $\mathbf{P} = \left[\begin{array}{c} p_0 \\ p_1 \\ p_2 \\ p_3 \end{array}\right]$
- Multiplying out **b** and **P** we get:

$$\mathbf{f}(u) = \mathbf{b}(u)\mathbf{P} = \left[\begin{array}{cccc} b_0(u) & b_1(u) & b_2(u) & b_3(u) \end{array}\right] \left[\begin{array}{c} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{array}\right] = \sum_{k=0}^{3} b_k(u)\mathbf{p}_k$$

Basis functions?

- Standard parametric form : $\mathbf{f}(u) = \mathbf{uBP}$
- We can multiply u and B to get a *vector of basis functions* $\mathbf{b}(u) = \mathbf{uB} = \left[\begin{array}{cccc} b_0(u) & b_1(u) & b_2(u) & b_3(u) \end{array}\right]$
- The curve is written as a linear combination of the *control points:*

$$\mathbf{f}(u) = \sum_{k=0}^{3} b_k(u)\mathbf{p}_k$$

The basis functions are the ones you will implement in practice (in code!)

- *Similarly, for the derivative:*

$$\mathbf{f}'(u) = \sum_{k=0}^{3} b'_k(u)\mathbf{p}_k$$

Basis functions?

jsbin.com/fawudah

Week6/Demo0

- For Hermite :

$$\mathbf{b}(u) = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3 & -2 & 3 & -1 \\ 2 & 1 & -2 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} b_0(u) & b_1(u) & b_2(u) & b_3(u) \end{bmatrix}$$

- Basis functions :

$$\begin{aligned} b_0(u) &= 2u^3 - 3u^2 + 1 \\ b_1(u) &= u^3 - 2u^2 + u \\ b_2(u) &= -2u^3 + 3u^2 \\ b_3(u) &= u^3 - u^2 \end{aligned}$$

- Curve formula :

$$\mathbf{f}(u) = \sum_{k=0}^{3} b_k(u)\mathbf{p}_k$$

Basis functions?

jsbin.com/fawudah

Week6/Demo0

- For Hermite :

$$\mathbf{b}(u) = \begin{bmatrix} \boxed{1} & u & u^2 & u^3 \end{bmatrix} \begin{bmatrix} \boxed{1} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3 & -2 & 3 & -1 \\ 2 & 1 & -2 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \boxed{b_0(u)} & b_1(u) & b_2(u) & b_3(u) \end{bmatrix}$$

- Basis functions :

$$\boxed{b_0(u)} = 2u^3 - 3u^2 + \boxed{1}$$
$$b_1(u) = u^3 - 2u^2 + u$$
$$b_2(u) = -2u^3 + 3u^2$$
$$b_3(u) = u^3 - u^2$$

- Curve formula :

$$\mathbf{f}(u) = \sum_{k=0}^{3} b_k(u)\mathbf{p}_k$$

Basis functions?

jsbin.com/fawudah

Week6/Demo0

- For Hermite :

$$\mathbf{b}(u) = \begin{bmatrix} 1 & u & \boxed{u^2} & u^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \boxed{-3} & -2 & 3 & -1 \\ 2 & 1 & -2 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \boxed{b_0(u)} & b_1(u) & b_2(u) & b_3(u) \end{bmatrix}$$

- Basis functions :

$$\boxed{b_0(u)} = 2u^3 \boxed{-3u^2} + 1$$
$$b_1(u) = u^3 - 2u^2 + u$$
$$b_2(u) = -2u^3 + 3u^2$$
$$b_3(u) = u^3 - u^2$$

- Curve formula :

$$\mathbf{f}(u) = \sum_{k=0}^{3} b_k(u)\mathbf{p}_k$$

Basis functions?

jsbin.com/fawudah

Week6/Demo0

- For Hermite :

$$\mathbf{b}(u) = \begin{bmatrix} 1 & u & u^2 & \boxed{u^3} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3 & -2 & 3 & -1 \\ \boxed{2} & 1 & -2 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \boxed{b_0(u)} & b_1(u) & b_2(u) & b_3(u) \end{bmatrix}$$

- Basis functions :

$$\boxed{b_0(u)} = \boxed{2u^3} - 3u^2 + 1$$
$$b_1(u) = u^3 - 2u^2 + u$$
$$b_2(u) = -2u^3 + 3u^2$$
$$b_3(u) = u^3 - u^2$$

- Curve formula :

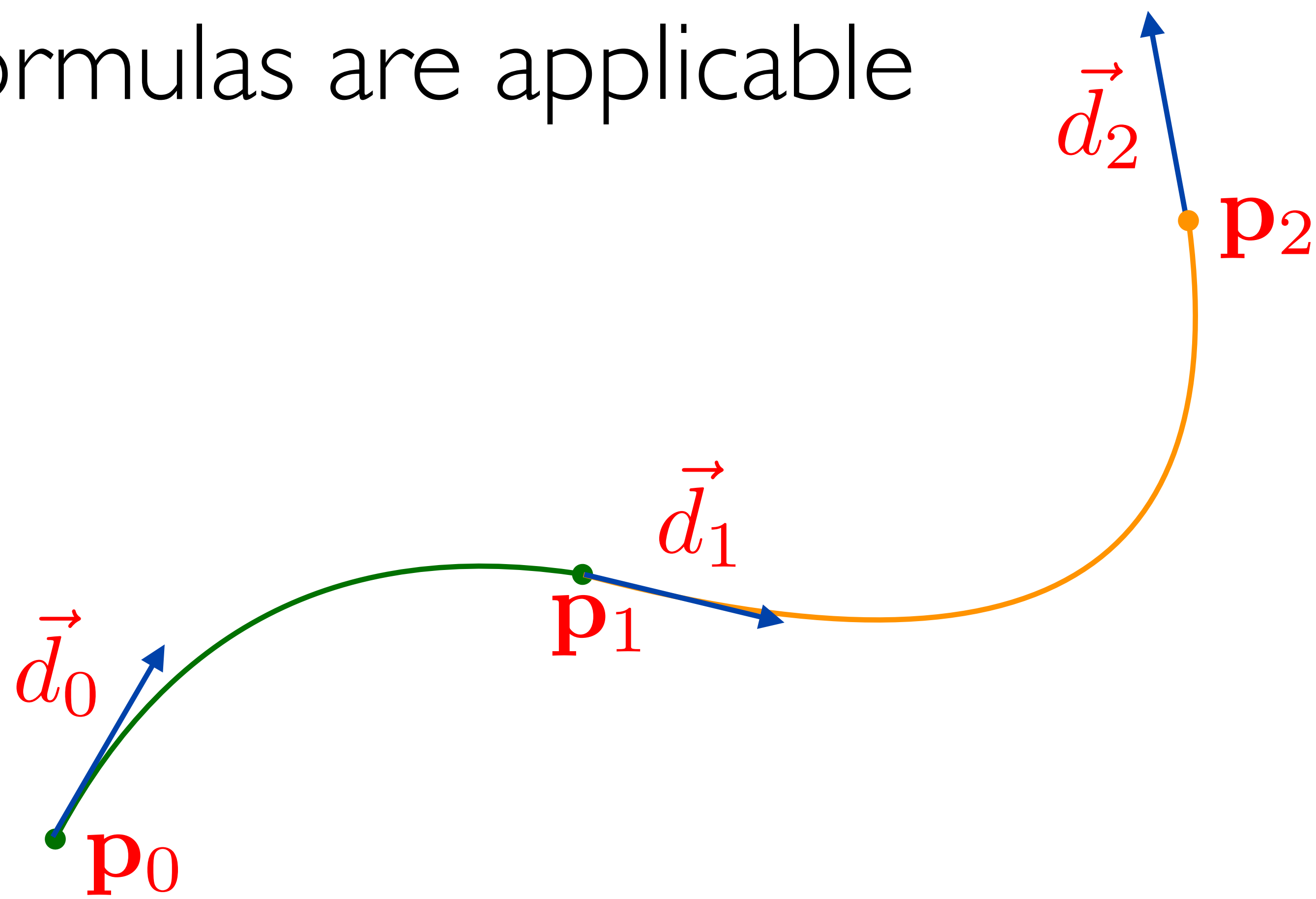
$$\mathbf{f}(u) = \sum_{k=0}^{3} b_k(u)\mathbf{p}_k$$

Hermite - Sample Implementation

jsbin.com/fawudah

Week6/Demo0

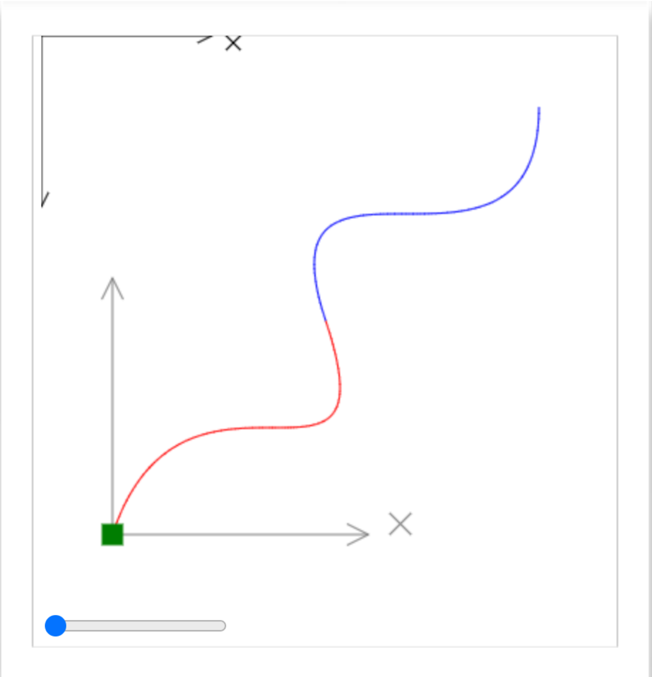
- Two Hermite curves, joined with C1 continuity
- Defined via 3 pairs of (location, tangent) control points, the middle one shared by the two curves
- The parametric interval for each piece of the curve is "translated" to the canonical interval [0,1], so that the previous formulas are applicable



Implementation example : 2-piece Hermite

jsbin.com/fawudah

Week6/Demo0



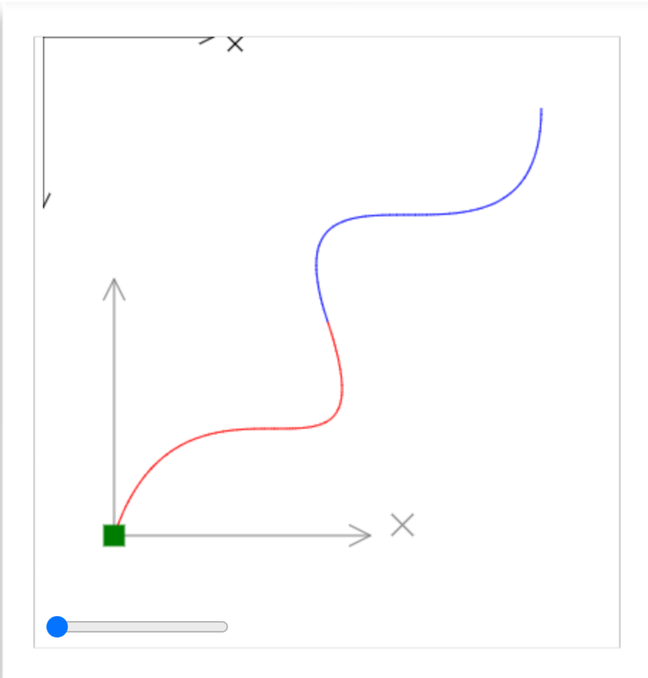
JavaScript

```
[...]
    var Hermite = function(t) {
        return [
        2*t*t*t-3*t*t+1,
        t*t*t-2*t*t+t,
        -2*t*t*t+3*t*t,
        t*t*t-t*t
        ];
    }

    function Cubic(basis,P,t){
        var b = basis(t);
        var result=vec2.create();
        vec2.scale(result,P[0],b[0]);
        vec2.scaleAndAdd(result,result,P[1],b[1]);
        vec2.scaleAndAdd(result,result,P[2],b[2]);
        vec2.scaleAndAdd(result,result,P[3],b[3]);
        return result;
    }

    var p0=[0,0];
    var d0=[1,3];
    var p1=[1,1];
    var d1=[-1,3];
    var p2=[2,2];
    var d2=[0,3];

    var P0 = [p0,d0,p1,d1]; // First two points and tangents
    var P1 = [p1,d1,p2,d2]; // Last two points and tangents

    var C0 = function(t_) {return Cubic(Hermite,P0,t_);};
    var C1 = function(t_) {return Cubic(Hermite,P1,t_);};

    var Ccomp = function(t) {
        if (t<1){
            var u = t;
            return C0(u);
        } else {
            var u = t-1.0;
            return C1(u);
        }
    }

[...]
```

Implementation example : 2-piece Hermite

jsbin.com/fawudah

Week6/Demo0



$$b_0(u) = 2u^3 - 3u^2 + 1$$
$$b_1(u) = u^3 - 2u^2 + u$$
$$b_2(u) = -2u^3 + 3u^2$$
$$b_3(u) = u^3 - u^2$$

Implementation of the basis functions

*Hermite(t) essentially returns the row vector **b**(u)*

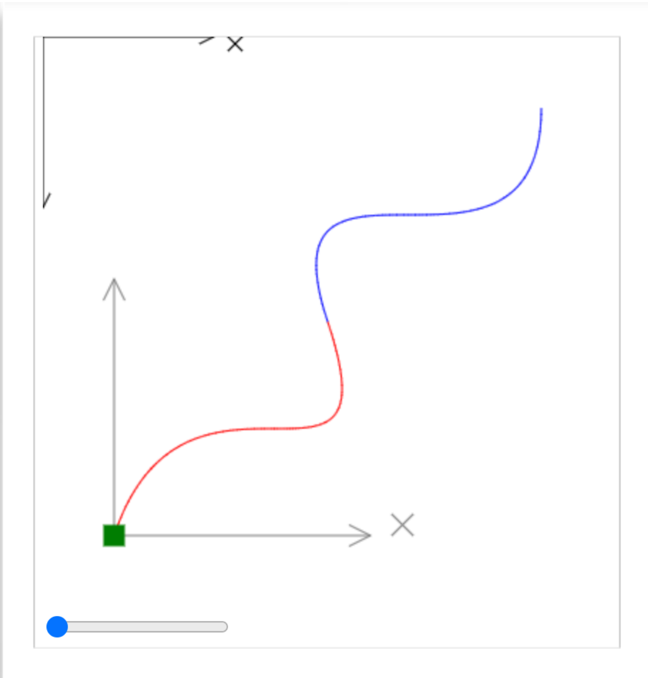
JavaScript

```
[...]
    var Hermite = function(t) {
        return [
        2*t*t*t-3*t*t+1,
        t*t*t-2*t*t+t,
        -2*t*t*t+3*t*t,
        t*t*t-t*t
        ];
    }

    function Cubic(basis,P,t){
        var b = basis(t);
        var result=vec2.create();
        vec2.scale(result,P[0],b[0]);
        vec2.scaleAndAdd(result,result,P[1],b[1]);
        vec2.scaleAndAdd(result,result,P[2],b[2]);
        vec2.scaleAndAdd(result,result,P[3],b[3]);
        return result;
    }

    var p0=[0,0];
    var d0=[1,3];
    var p1=[1,1];
    var d1=[-1,3];
    var p2=[2,2];
    var d2=[0,3];

    var P0 = [p0,d0,p1,d1]; // First two points and tangents
    var P1 = [p1,d1,p2,d2]; // Last two points and tangents

    var C0 = function(t_) {return Cubic(Hermite,P0,t_);};
    var C1 = function(t_) {return Cubic(Hermite,P1,t_);};

    var Ccomp = function(t) {
        if (t<1){
            var u = t;
            return C0(u);
        } else {
            var u = t-1.0;
            return C1(u);
        }
    }
[...]
```

Implementation example : 2-piece Hermite

jsbin.com/fawudah

Week6/Demo0



$$\mathbf{f}(u) = \sum_{k=0}^{3} b_k(u)\mathbf{p}_k$$

Computing the curve as a "weighted average" of control points (implemented using glMatrix calls for scaling and adding vectors)

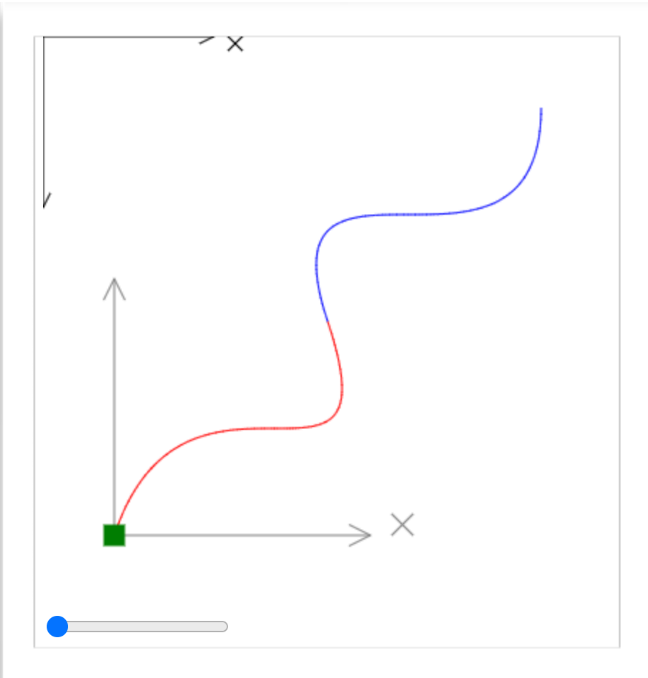
JavaScript

```
[...]
    var Hermite = function(t) {
        return [
         2*t*t*t-3*t*t+1,
         t*t*t-2*t*t+t,
         -2*t*t*t+3*t*t,
         t*t*t-t*t
         ];
    }

    function Cubic(basis,P,t){
        var b = basis(t);
        var result=vec2.create();
        vec2.scale(result,P[0],b[0]);
        vec2.scaleAndAdd(result,result,P[1],b[1]);
        vec2.scaleAndAdd(result,result,P[2],b[2]);
        vec2.scaleAndAdd(result,result,P[3],b[3]);
        return result;
    }

    var p0=[0,0];
    var d0=[1,3];
    var p1=[1,1];
    var d1=[-1,3];
    var p2=[2,2];
    var d2=[0,3];

    var P0 = [p0,d0,p1,d1]; // First two points and tangents
    var P1 = [p1,d1,p2,d2]; // Last two points and tangents

    var C0 = function(t_) {return Cubic(Hermite,P0,t_);};
    var C1 = function(t_) {return Cubic(Hermite,P1,t_);};

    var Ccomp = function(t) {
        if (t<1){
            var u = t;
            return C0(u);
        } else {
            var u = t-1.0;
            return C1(u);
        }
    }
[...]
```

Implementation example : 2-piece Hermite

jsbin.com/fawudah

Week6/Demo0



Defining two curves $C_0(t)$ and $C_1(t)$ that use respectively the first two (or last two) pass-through points & tangents

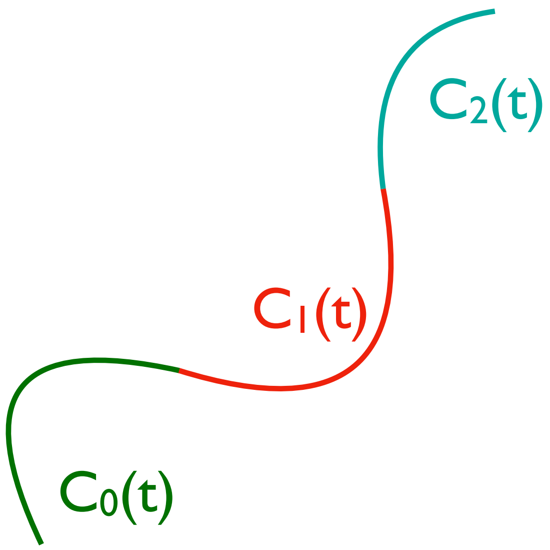
(note that both curves assume a parameter that takes value in the range [0,1]!)

JavaScript

```
[...]
    var Hermite = function(t) {
        return [
        2*t*t*t-3*t*t+1,
        t*t*t-2*t*t+t,
        -2*t*t*t+3*t*t,
        t*t*t-t*t
        ];
    }

    function Cubic(basis,P,t){
        var b = basis(t);
        var result=vec2.create();
        vec2.scale(result,P[0],b[0]);
        vec2.scaleAndAdd(result,result,P[1],b[1]);
        vec2.scaleAndAdd(result,result,P[2],b[2]);
        vec2.scaleAndAdd(result,result,P[3],b[3]);
        return result;
    }

    var p0=[0,0];
    var d0=[1,3];
    var p1=[1,1];
    var d1=[-1,3];
    var p2=[2,2];
    var d2=[0,3];

    var P0 = [p0,d0,p1,d1]; // First two points and tangents
    var P1 = [p1,d1,p2,d2]; // Last two points and tangents

    var C0 = function(t_) {return Cubic(Hermite,P0,t_);};
    var C1 = function(t_) {return Cubic(Hermite,P1,t_);};

    var Ccomp = function(t) {
        if (t<1){
            var u = t;
            return C0(u);
        } else {
            var u = t-1.0;
            return C1(u);
        }
    }

[...]
```

Piecewise parametric curves

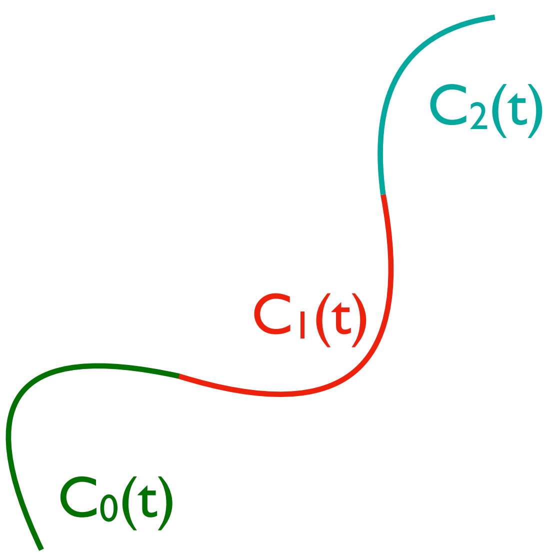
- It can be convenient to define different extents of a parametric curve using different formulas (rather than attempting to over-complicate a *single* formula)



$$\mathcal{C}(t) = \begin{cases} \mathcal{C}_0(t), & t \in [t_0, t_1] \\ \mathcal{C}_1(t), & t \in [t_1, t_2] \\ \vdots & \\ \mathcal{C}_{N-1}(t), & t \in [t_{N-1}, t_N] \end{cases}$$

Piecewise parametric curves

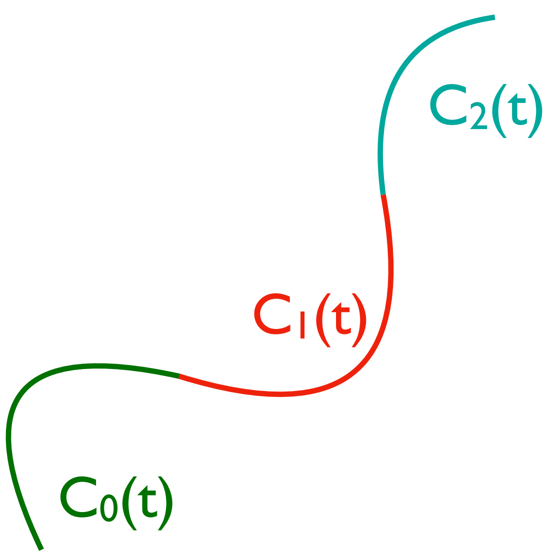
- It can be convenient to define different extents of a parametric curve using different formulas (rather than attempting to over-complicate a *single* formula)



$$\mathcal{C}(t) = \begin{cases} \mathcal{C}_0(t), \quad t \in [t_0, t_1] & = \tilde{\mathcal{C}}_0(u), \text{ where } u = \dfrac{t - t_0}{t_1 - t_0} \\ \mathcal{C}_1(t), \quad t \in [t_1, t_2] & = \tilde{\mathcal{C}}_1(u), \text{ where } u = \dfrac{t - t_2}{t_2 - t_1} \\ \vdots & \\ \mathcal{C}_{N-1}(t), \quad t \in [t_{N-1}, t_N] & = \tilde{\mathcal{C}}_{N-1}(u), \text{ where } u = \dfrac{t - t_N}{t_{N-1} - t_N} \end{cases}$$

Piecewise parametric curves

- It can be convenient to define different extents of a parametric curve using different formulas (rather than attempting to over-complicate a *single* formula)



Using this auxiliary variable “u” (that is defined in a different way in each distinct interval), we can build the composite curves using components (the functions with the “tilde”) that are always parameterized on the interval [0,1]

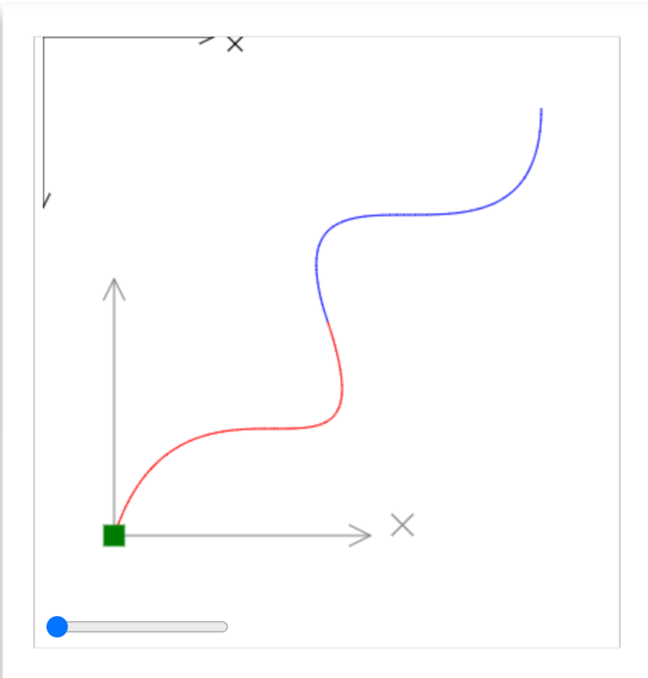
Hazard/Point of Caution: If the components are designed to join with C1-continuity, this re-parameterization might drop the continuity to G1, if the intervals are not equal in length!

$$\mathcal{C}(t) = \begin{cases} \mathcal{C}_0(t), & t \in [t_0, t_1] \quad = \tilde{\mathcal{C}}_0(u), \text{ where } u = \dfrac{t - t_1}{t_1 - t_0} \\ \mathcal{C}_1(t), & t \in [t_1, t_2] \quad = \tilde{\mathcal{C}}_1(u), \text{ where } u = \dfrac{t - t_2}{t_2 - t_1} \\ \vdots & \\ \mathcal{C}_{N-1}(t), & t \in [t_{N-1}, t_N] \quad = \tilde{\mathcal{C}}_{N-1}(u), \text{ where } u = \dfrac{t - t_N}{t_{N-1} - t_N} \end{cases}$$

Implementation example : 2-piece Hermite

jsbin.com/fawudah

Week6/Demo0



Building a composite curve over the parameter interval [0,2], by using an auxiliary "u" parameter, and pieces that are defined over [0,1] …

JavaScript

```
[...]
    var Hermite = function(t) {
        return [
        2*t*t*t-3*t*t+1,
        t*t*t-2*t*t+t,
        -2*t*t*t+3*t*t,
        t*t*t-t*t
        ];
    }

    function Cubic(basis,P,t){
        var b = basis(t);
        var result=vec2.create();
        vec2.scale(result,P[0],b[0]);
        vec2.scaleAndAdd(result,result,P[1],b[1]);
        vec2.scaleAndAdd(result,result,P[2],b[2]);
        vec2.scaleAndAdd(result,result,P[3],b[3]);
        return result;
    }

    var p0=[0,0];
    var d0=[1,3];
    var p1=[1,1];
    var d1=[-1,3];
    var p2=[2,2];
    var d2=[0,3];

    var P0 = [p0,d0,p1,d1]; // First two points and tangents
    var P1 = [p1,d1,p2,d2]; // Last two points and tangents

    var C0 = function(t_) {return Cubic(Hermite,P0,t_);};
    var C1 = function(t_) {return Cubic(Hermite,P1,t_);};

    var Ccomp = function(t) {
        if (t<1){
            var u = t;
            return C0(u);
        } else {
            var u = t-1.0;
            return C1(u);
        }
    }
[...]
```

Derivatives?

jsbin.com/bodorun

Week6/Demo1

- Still using Hermite basis matrix :

$$\mathbf{b}'(u) = \begin{bmatrix} 0 & 1 & 2u & 3u^2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3 & -2 & 3 & -1 \\ 2 & 1 & -2 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} b'_0(u) & b'_1(u) & b'_2(u) & b'_3(u) \end{bmatrix}$$

- Derivatives of basis functions :

$$\begin{aligned} b'_0(u) &= 6u^2 - 6u \\ b'_1(u) &= 3u^2 - 4u + 1 \\ b'_2(u) &= -6u^2 + 6u \\ b'_3(u) &= 3u^2 - 2u \end{aligned}$$

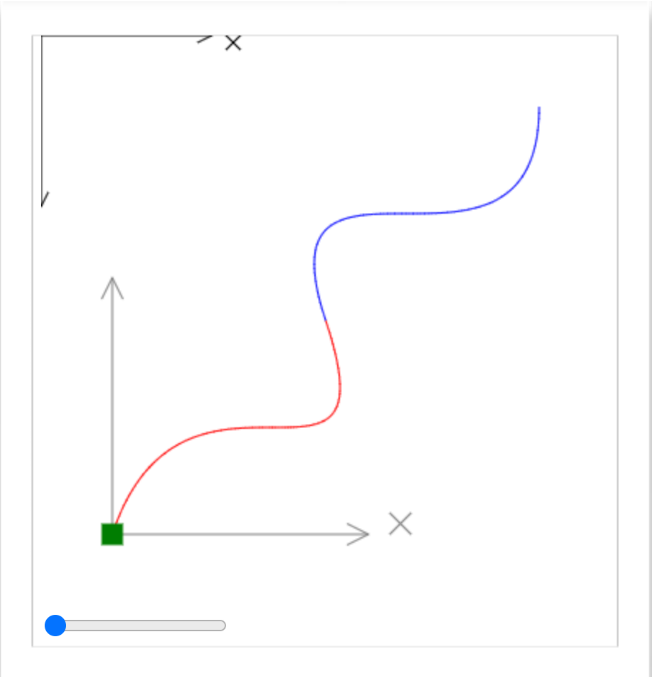
- Tangent formula :

$$\mathbf{f}'(u) = \sum_{k=0}^{3} b'_k(u)\mathbf{p}_k$$

Implementation example : 2-piece Hermite

jsbin.com/bodorun

Week6/Demo1



$$b'_0(u) = 6u^2 - 6u$$
$$b'_1(u) = 3u^2 - 4u + 1$$
$$b'_2(u) = -6u^2 + 6u$$
$$b'_3(u) = 3u^2 - 2u$$

Expressions for the derivatives are built exactly the same as expressions for the curve (the only difference is we're taking weighted averages of <u>differentiated</u> basis functions)

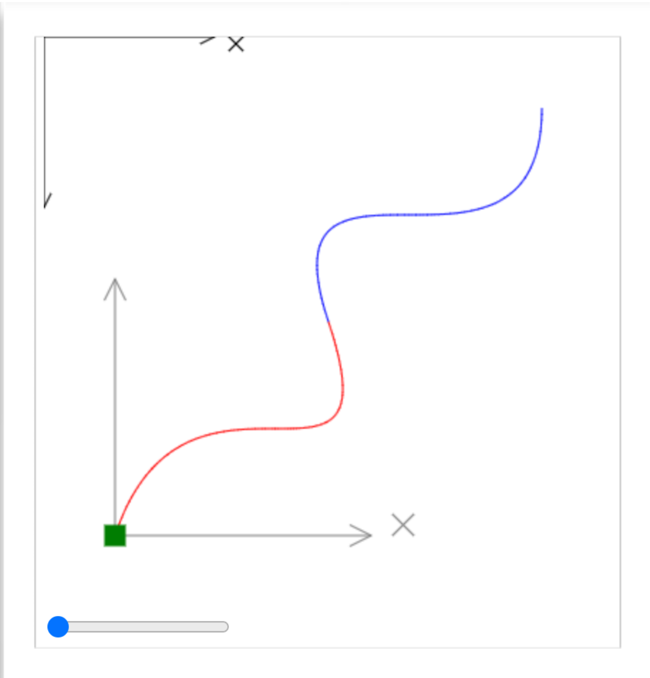
JavaScript

```
[...]
    var Hermite = function(t) {
        return [
        2*t*t*t-3*t*t+1,
        t*t*t-2*t*t+t,
        -2*t*t*t+3*t*t,
        t*t*t-t*t
        ];
    }

    var HermiteDerivative = function(t) {
        return [
        6*t*t-6*t,
        3*t*t-4*t+1,
        -6*t*t+6*t,
        3*t*t-2*t
        ];
    }

    function Cubic(basis,P,t){
        var b = basis(t);
        var result=vec2.create();
        vec2.scale(result,P[0],b[0]);
        vec2.scaleAndAdd(result,result,P[1],b[1]);
        vec2.scaleAndAdd(result,result,P[2],b[2]);
        vec2.scaleAndAdd(result,result,P[3],b[3]);
        return result;
    }

    var p0=[0,0];
    var d0=[1,3];
    var p1=[1,1];
    var d1=[-1,3];
    var p2=[2,2];
    var d2=[0,3];

    var P0 = [p0,d0,p1,d1]; // First two points and tangents
    var P1 = [p1,d1,p2,d2]; // Last two points and tangents

    var C0 = function(t_) {return Cubic(Hermite,P0,t_);};
    var C1 = function(t_) {return Cubic(Hermite,P1,t_);};

    var C0prime = function(t_) {return Cubic(HermiteDerivative,P0,t_);};
    var C1prime = function(t_) {return Cubic(HermiteDerivative,P1,t_);};
[...]
```

Implementation example : 2-piece Hermite

jsbin.com/bodorun

Week6/Demo1



… and the composite tangent is built via the auxiliary parameter as before.

JavaScript

```
[...]
    var p0=[0,0];
    var d0=[1,3];
    var p1=[1,1];
    var d1=[-1,3];
    var p2=[2,2];
    var d2=[0,3];

    var P0 = [p0,d0,p1,d1]; // First two points and tangents
    var P1 = [p1,d1,p2,d2]; // Last two points and tangents

    var C0 = function(t_) {return Cubic(Hermite,P0,t_);};
    var C1 = function(t_) {return Cubic(Hermite,P1,t_);};

    var C0prime = function(t_) {return Cubic(HermiteDerivative,P0,t_);};
    var C1prime = function(t_) {return Cubic(HermiteDerivative,P1,t_);};

    var Ccomp = function(t) {
        if (t<1){
            var u = t;
            return C0(u);
        } else {
            var u = t-1.0;
            return C1(u);
        }
    }

    var Ccomp_tangent = function(t) {
        if (t<1){
            var u = t;
            return C0prime(u);
        } else {
            var u = t-1.0;
            return C1prime(u);
        }
    }
[...]
```

B-splines

Flash preview will revisit!

jsbin.com/viwinoj

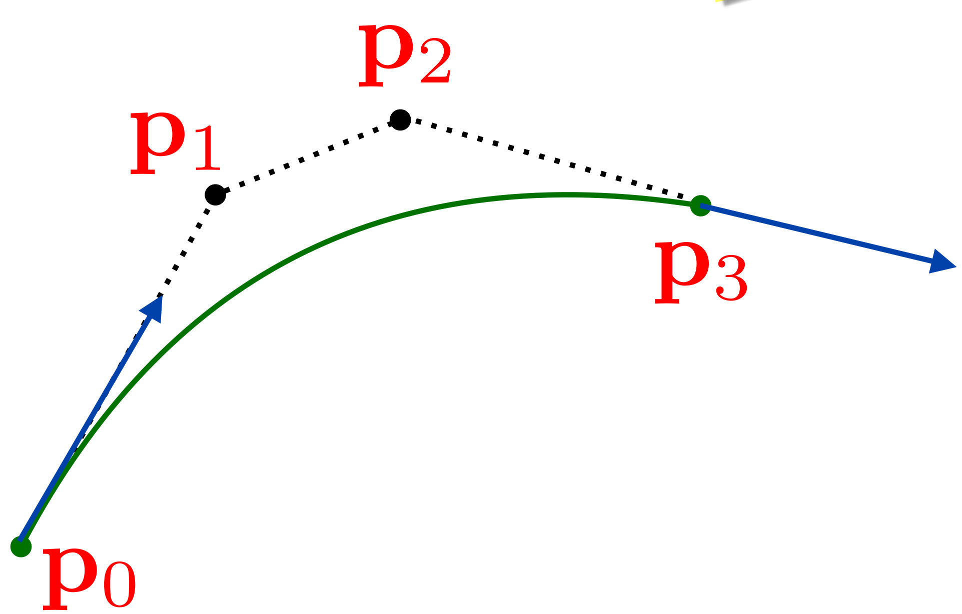
Week6/Demo2

- Using 4 control points (p_0,p_1,p_2,p_3) it creates a curve that *approximates the arc from* p_1->p_2 (but doesn't necessarily go through either point)
- A sequence of curves that respectively use points (p_0,p_1,p_2,p_3), (p_1,p_2,p_3,p_4), (p_2,p_3,p_4,p_5) etc will join with C2-continuity! (but will only approximate the points)
- If passing through a point is needed, duplicate it in the sequence of control pts! (but C2-continuity is lost)
- Correction from notes …

$$\mathbf{B} = \frac{1}{6}\begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

Bezier cubics (and subdivision)

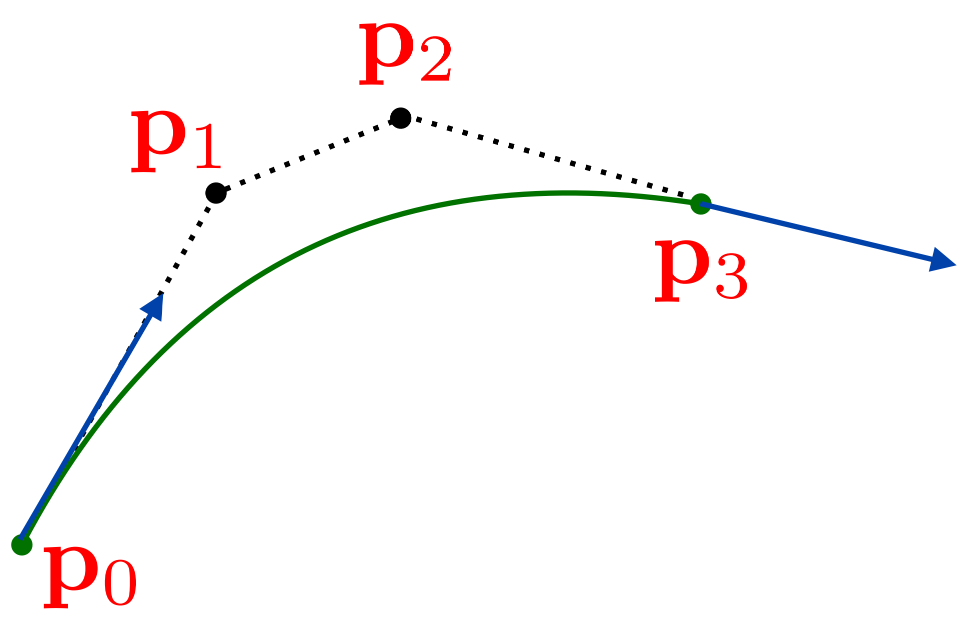
Flash preview
will revisit!



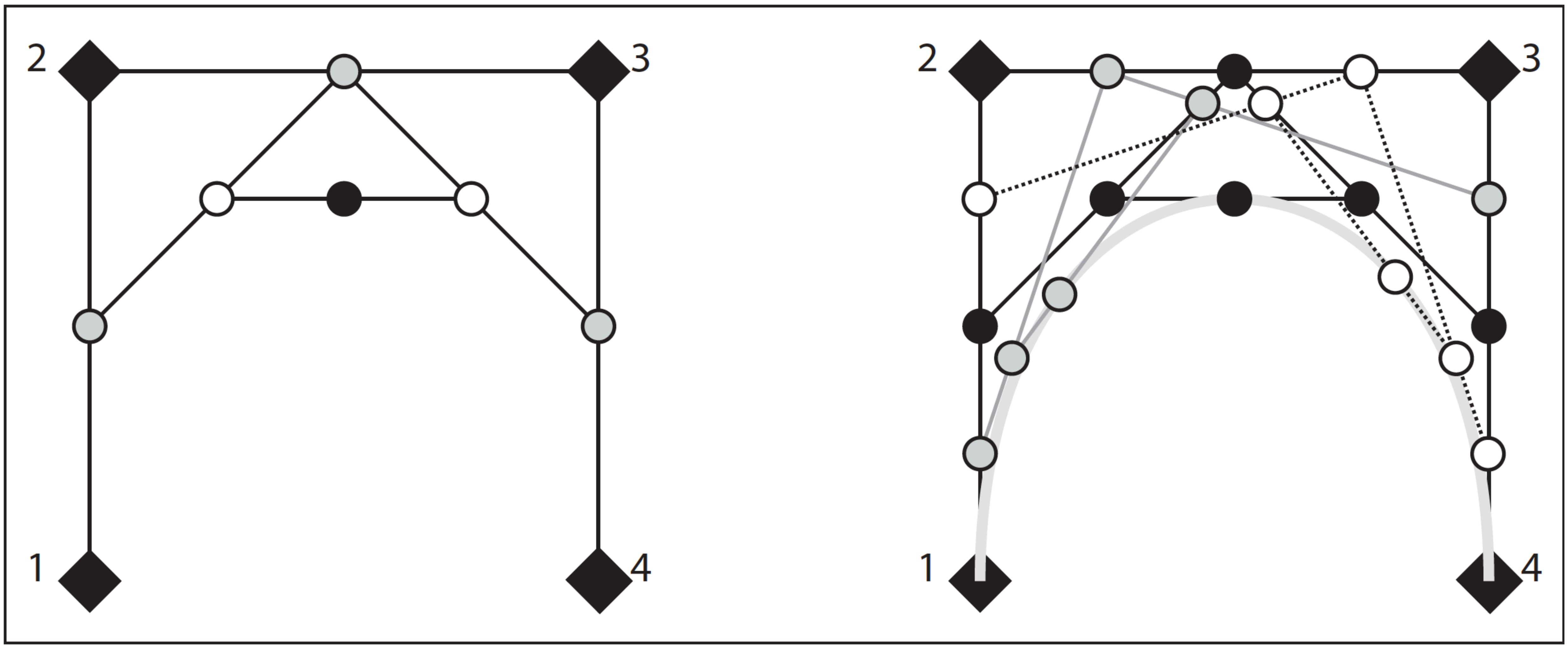
- Conceptually, a "variant" of Hermite!
- Initial gradient set to $\mathbf{f}'(0) = \gamma(\mathbf{p}_1 - \mathbf{p}_0)$
- Final gradient set to $\mathbf{f}'(1) = \gamma(\mathbf{p}_3 - \mathbf{p}_2)$
- Special value $\gamma=3$ guarantees the *interpolation property*(this is when we speak of Bezier curves)

Bezier cubics (and subdivision)

Flash preview
will revisit!



- Can be constructed via *De Casteljau Subdivision*



Natural cubics

Flash preview
will revisit!

- Specify up to 2nd derivative at origin *C(0),C'(0),C''(0)* and just position *C(1)* at end
- We can *evaluate* first/second derivative at end of interval, and create the next spline to match!
- Yields C2-continuity; loses local control