Discussion 04

Rasterization, Splines, & Curves

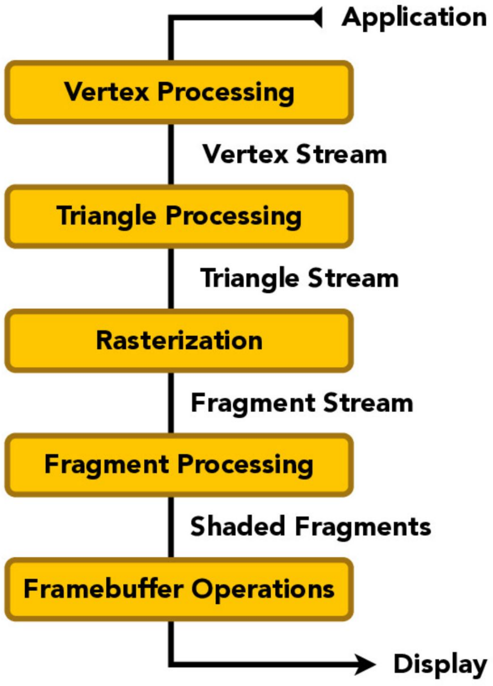
Computer Graphics and Imaging
UC Berkeley CS 184/284A

Week 4 Announcements

- Homework 2 will be released Wednesday evening
- Homework 1 is due tomorrow night.

Graphics Pipeline

Graphics Pipeline

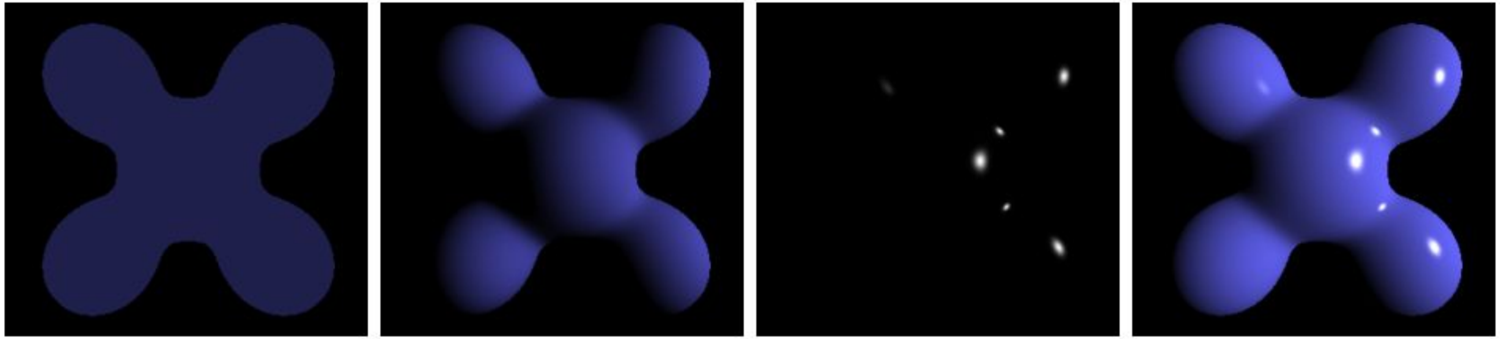


1. Vertex Processing: operations on geometry.
    a. Transformations to screen space.
2. Rasterization.
    a. Lines, triangles.
3. Fragment Processing: operations on pixels (fragments).
    a. Hidden surface removal.
    b. Per-fragment shading.

Blinn-Phong Reflection Model

Blinn-Phong Reflection Model

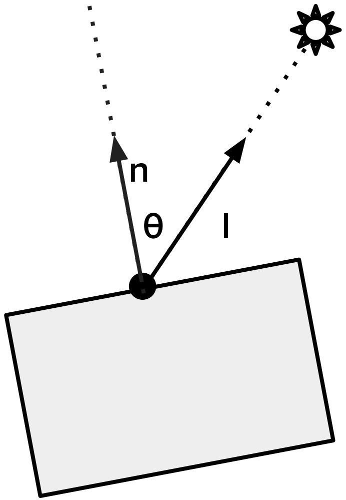
- Not physically-based.
- A material has four parameters: k_a, k_d, k_s, p (shininess).



$$k_a \, I_a \qquad k_d \, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{l}) \qquad k_s \, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{h})^p$$

Diffuse Shading

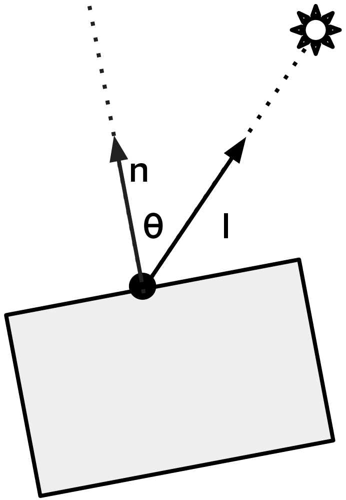
- Use normal vector, *n*, and light direction vector, *l*:



Diffuse Shading

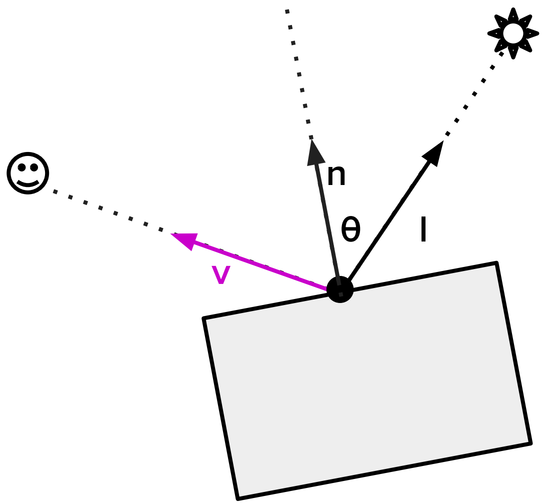
- Use normal vector, *n*, and light direction vector, *l*:

$$k_d \, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{l})$$



Specular Shading

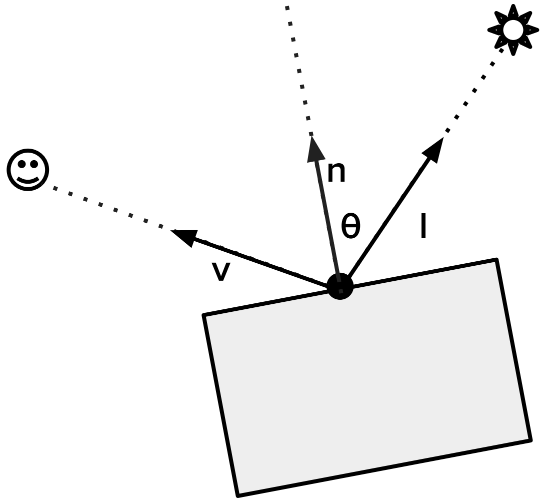
- <u>Viewing direction</u>, *v*, is in the direction of the viewer.



Specular Shading

- Viewing direction, *v*, is in the direction of the viewer.
- Calculate half-vector, *h*:

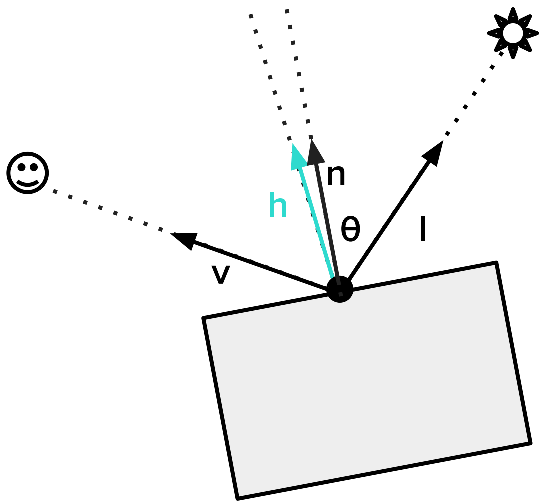
$$\begin{aligned}\mathbf{h} &= \mathrm{bisector}(\mathbf{v}, \mathbf{l}) \\ &= \frac{\mathbf{v} + \mathbf{l}}{\|\mathbf{v} + \mathbf{l}\|}\end{aligned}$$



Specular Shading

- Viewing direction, *v*, is in the direction of the viewer.
- Calculate <u>half-vector</u>, *h*:

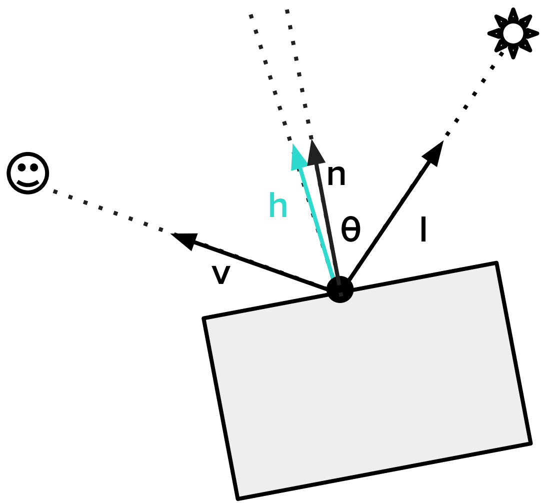
$$\mathbf{h} = \text{bisector}(\mathbf{v}, \mathbf{l})$$
$$= \frac{\mathbf{v} + \mathbf{l}}{\|\mathbf{v} + \mathbf{l}\|}$$



Specular Shading

- Viewing direction, *v*, is in the direction of the viewer.
- Calculate half-vector, *h*:

$$\mathbf{h} = \operatorname{bisector}(\mathbf{v}, \mathbf{l}) = \frac{\mathbf{v} + \mathbf{l}}{\|\mathbf{v} + \mathbf{l}\|}$$

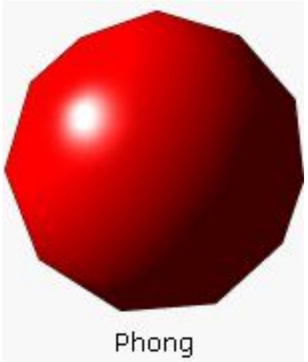


- Calculate specular shading: $k_s \, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{h})^p$

Phong Shading vs. Gouraud Shading

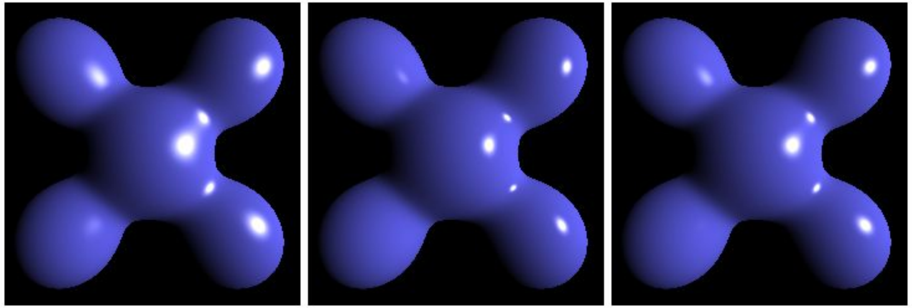
- Interpolate vertex normals per pixel →
- Compute light per pixel.
- Slow.

- Compute light per vertex →
- Interpolate vertex colors per pixel.
- Fast.



Blinn-Phong Reflection Model

- By default, Gouraud shading.
- Efficient!

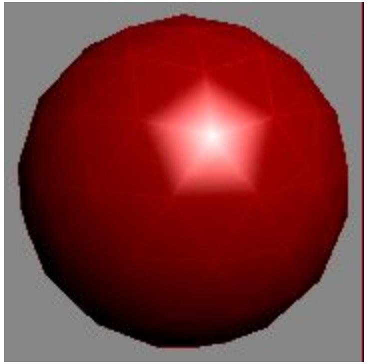


Blinn–Phong　　Phong　　Blinn–Phong (higher exponent)

A Pop Quiz.

Is Blinn-Phong shading with Gouraud shading part of

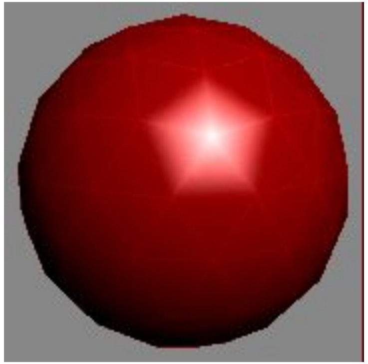
Vertex Processing or Fragment Processing?



A Pop Quiz.

Is Blinn-Phong shading with Gouraud shading part of

Vertex Processing or Fragment Processing?



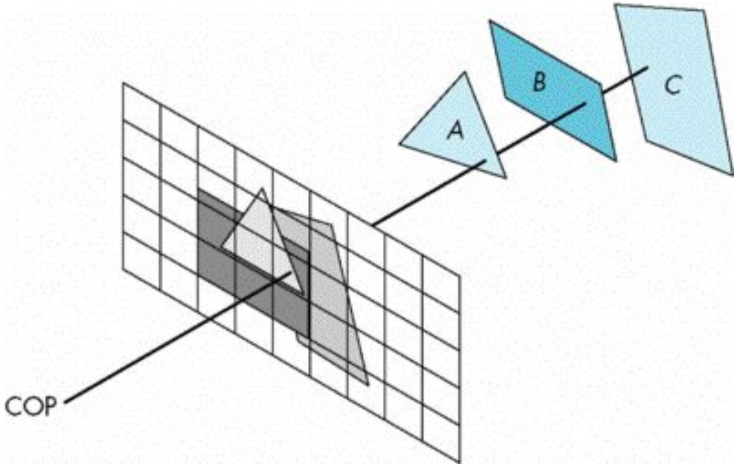
It's part of both!

- Vertex Processing: compute light per vertex.
- Fragment Processing: interpolate vertex colors to set pixel value.

Hidden Surface Removal

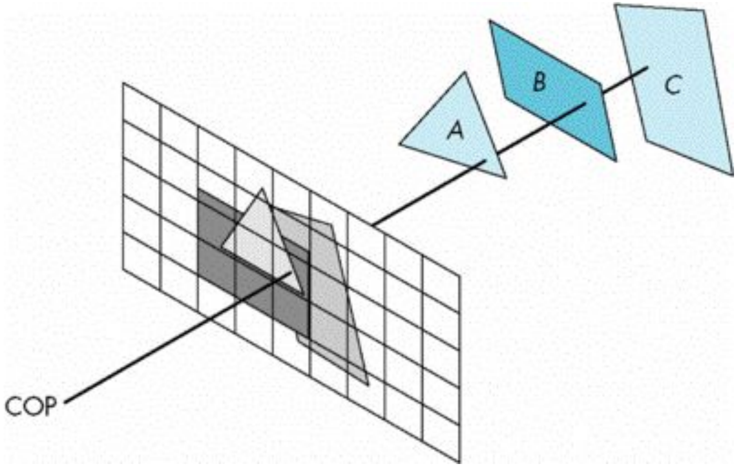
Hidden Surface Removal

- <u>Problem</u>: primitives may overlap ➔ screen should only display what is visible from the front.



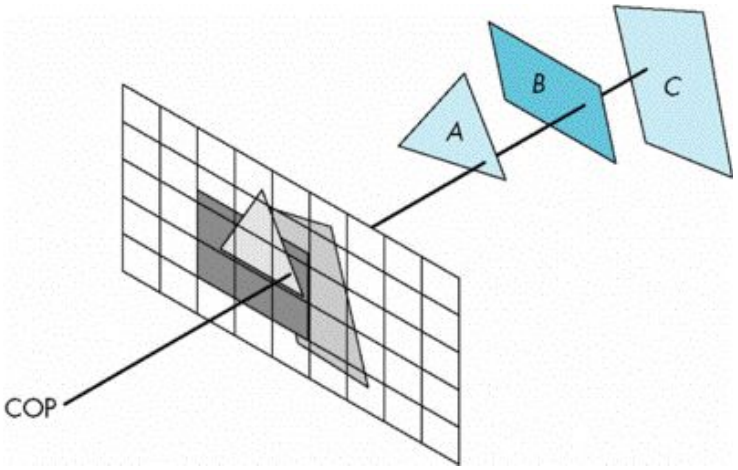
Hidden Surface Removal

- <u>Problem</u>: primitives may overlap → screen should only display what is visible from the front.
- <u>Solution</u>: track the depth (z-value) of fragments →



Hidden Surface Removal

- <u>Problem</u>: primitives may overlap ➔ screen should only display what is visible from the front.
- <u>Solution</u>: track the depth (z-value) of fragments ➔
  during fragment blending, pixel takes on value of <u>closest fragment</u>.



```
void zBufferAlgorithm(const ______________________________________ triangles,
                            ______________________________________ framebuffer,
                            ______________________________________ zbuffer) {
    for (const Triangle& T : triangles) {
        for (const Sample& sample : T.samples) {
            int x = sample.x;
            int y = sample.y;
            float z = sample.z;

            if (x >= 0 && x < WIDTH && y >= 0 && y < HEIGHT) {
                if (______________________________________) {
                    framebuffer[x][y] = sample.color;
                    ______________________________________;
                }
            }
        }
    }
}
```

```
void zBufferAlgorithm(const std::vector<Triangle>& __________ triangles,
                            ______________________________ framebuffer,
                            ______________________________ zbuffer) {
    for (const Triangle& T : triangles) {
        for (const Sample& sample : T.samples) {
            int x = sample.x;
            int y = sample.y;
            float z = sample.z;

            if (x >= 0 && x < WIDTH && y >= 0 && y < HEIGHT) {
                if (______________________________) {
                    framebuffer[x][y] = sample.color;
                    ______________________________;
                }
            }
        }
    }
}
```

```
void zBufferAlgorithm(const std::vector<Triangle>&_____________ triangles,
                           std::vector<std::vector<Color>>&____ framebuffer,
                           ____________________________________ zbuffer) {
    for (const Triangle& T : triangles) {
        for (const Sample& sample : T.samples) {
            int x = sample.x;
            int y = sample.y;
            float z = sample.z;

            if (x >= 0 && x < WIDTH && y >= 0 && y < HEIGHT) {
                if (_______________________________________) {
                    framebuffer[x][y] = sample.color;
                    _______________________________________;
                }
            }
        }
    }
}
```

```
void zBufferAlgorithm(const std::vector<Triangle>&________ triangles,
                            std::vector<std::vector<Color>>&_ framebuffer,
                            std::vector<std::vector<float>>&_ zbuffer) {
    for (const Triangle& T : triangles) {
        for (const Sample& sample : T.samples) {
            int x = sample.x;
            int y = sample.y;
            float z = sample.z;

            if (x >= 0 && x < WIDTH && y >= 0 && y < HEIGHT) {
                if (______________________________________) {
                    framebuffer[x][y] = sample.color;
                    ______________________________________;
                }
            }
        }
    }
}
```

```
void zBufferAlgorithm(const std::vector<Triangle>&              triangles,
                            std::vector<std::vector<Color>>&    framebuffer,
                            std::vector<std::vector<float>>&    zbuffer) {
    for (const Triangle& T : triangles) {
        for (const Sample& sample : T.samples) {
            int x = sample.x;
            int y = sample.y;
            float z = sample.z;

            if (x >= 0 && x < WIDTH && y >= 0 && y < HEIGHT) {
                if (z < zbuffer[x][y]) {
                    framebuffer[x][y] = sample.color;
                    zbuffer[x][y] = z;
                }
            }
        }
    }
}
```

```
void zBufferAlgorithm(const std::vector<Triangle>&                      triangles,
                            std::vector<std::vector<Color>>&             framebuffer,
                            std::vector<std::vector<float>>&             zbuffer) {

    for (const Triangle& T : triangles) {
        for (const Sample& sample : T.samples) {
            int x = sample.x;
            int y = sample.y;
            float z = sample.z;

            if (x >= 0 && x < WIDTH && y >= 0 && y < HEIGHT) {
                if (z < zbuffer[x][y]) {
                    framebuffer[x][y] = sample.color;
                    zbuffer[x][y] = z;
                }
            }
        }
    }
}
```

5. Prior to running this algorithm, what should the Z-buffer values be initialized to?

```
void zBufferAlgorithm(const std::vector<Triangle>&                    triangles,
                            std::vector<std::vector<Color>>&          framebuffer,
                            std::vector<std::vector<float>>&          zbuffer) {

    for (const Triangle& T : triangles) {
        for (const Sample& sample : T.samples) {
            int x = sample.x;
            int y = sample.y;
            float z = sample.z;

            if (x >= 0 && x < WIDTH && y >= 0 && y < HEIGHT) {
                if (z < zbuffer[x][y]) {
                    framebuffer[x][y] = sample.color;
                    zbuffer[x][y] = z;
                }
            }
        }
    }
}
```

5. Prior to running this algorithm, what should the Z-buffer values be initialized to?

Infinity! `std::numeric_limits<float>::infinity()`

Cubic Hermite Interpolation

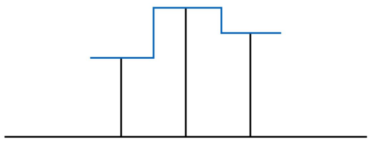
Cubic Hermite Interpolation

Interpolation: combines discrete points (explicit geometry) into a shape.

Why Cubic Hermite interpolation?

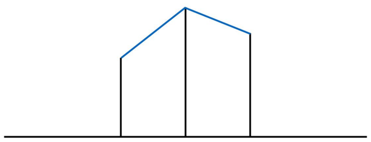
- We want something that's **continuous** (unlike nearest neighbor interpolation) and has a **continuous derivative** (unlike linear interpolation)

Nearest Neighbor Interpolation



Problem: values not continuous

Linear Interpolation



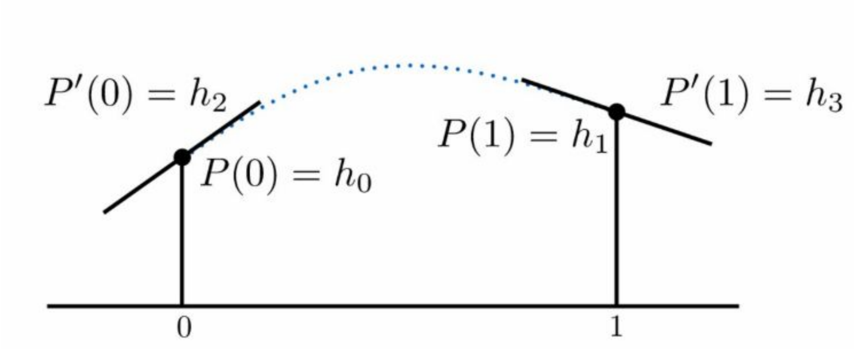
Problem: derivatives not continuous

Cubic Hermite Interpolation

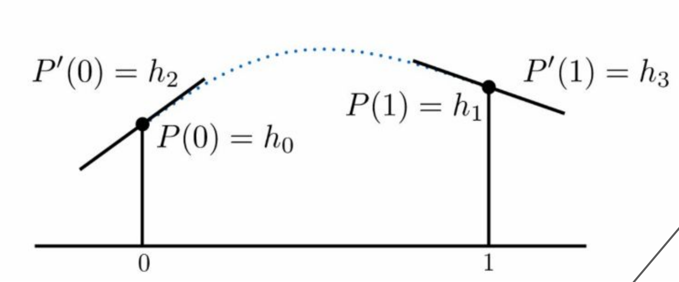
Input: the **values (P) & derivatives (P')** at selected endpoints

Output: a **cubic polynomial** that interpolates between them

Solution: a **weighted sum** of **Hermite basis functions**



Hermite Basis Functions

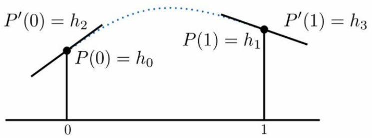


can simply plug in constraints!

$$P(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} H_0(t) & H_1(t) & H_2(t) & H_3(t) \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

t^3	$H_0(t) = 2t^3 - 3t^2 + 1$
t^2	$H_1(t) = -2t^3 + 3t^2$
t	$H_2(t) = t^3 - 2t^2 + t$
1	$H_3(t) = t^3 - t^2$
Basis functions for cubic polynomials	**Hermite basis functions for cubic polynomials**

Math Deepdive: Derivation



$$P(t) = h_0\, H_0(t) + h_1\, H_1(t) + h_2\, H_2(t) + h_3\, H_3(t)$$

Cubic polynomial

$$P(t) = a\,t^3 + b\,t^2 + c\,t + d$$
$$P'(t) = 3a\,t^2 + 2b\,t + c$$

Set up constraint equations

$$P(0) = h_0 = d$$
$$P(1) = h_1 = a + b + c + d$$
$$P'(0) = h_2 = c$$
$$P'(1) = h_3 = 3a + 2b + c$$

Solve for Polynomial Coefficients

$$h_0 = d$$
$$h_1 = a + b + c + d$$
$$h_2 = c$$
$$h_3 = 3a + 2b + c$$

$$\begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

$$P(t) = a\,t^3 + b\,t^2 + c\,t + d$$
$$= \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$
$$= \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

$$P(t) = a\,t^3 + b\,t^2 + c\,t + d$$
$$= \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}$$
$$= \begin{bmatrix} 2t^3 - 3t^2 + 1 \\ -2t^3 + 3t^2 \\ t^3 - 2t^2 + t \\ t^3 - t^2 \end{bmatrix}^T \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

Catmull Rom Interpolation

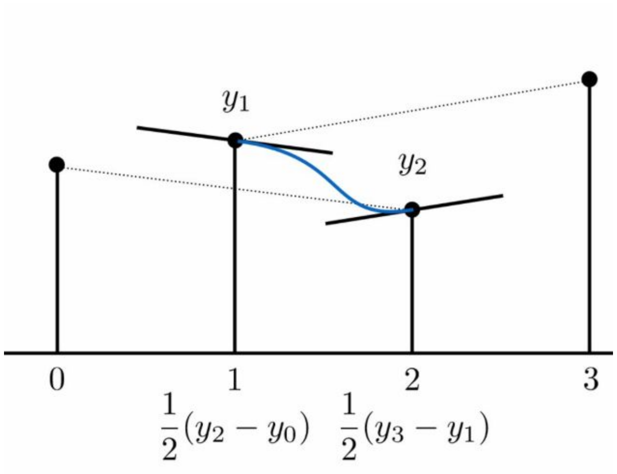
Catmull Rom Interpolation

Input: sequence of points

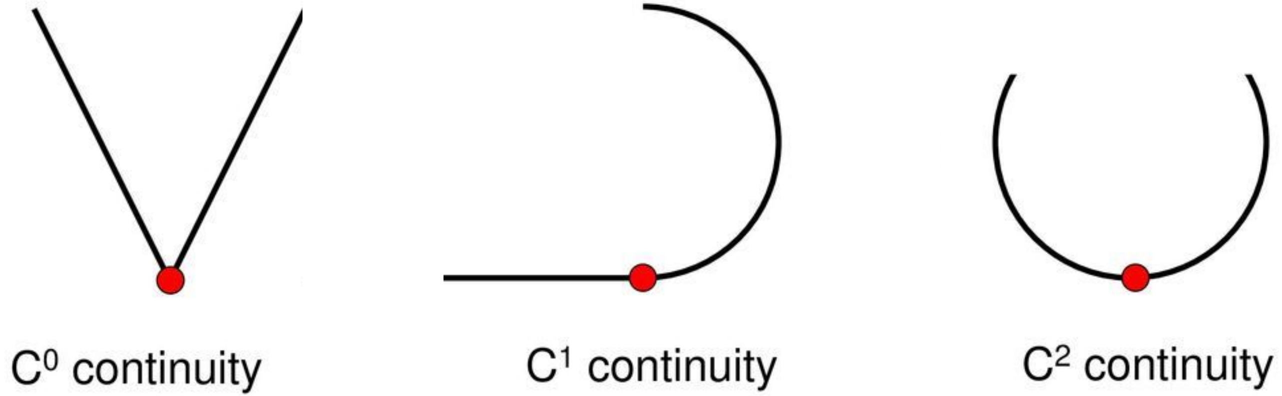
1. Calculate the slopes between alternating points
2. Use Hermite interpolation

Output: spline with C1 continuity

Procedure: make slope using previous and after values, then use Hermite Interpolation.



Splicing curves together



Ends of segments meet

Ends of segments meet

Equal tangent vectors for both segments

Ends of segments meet

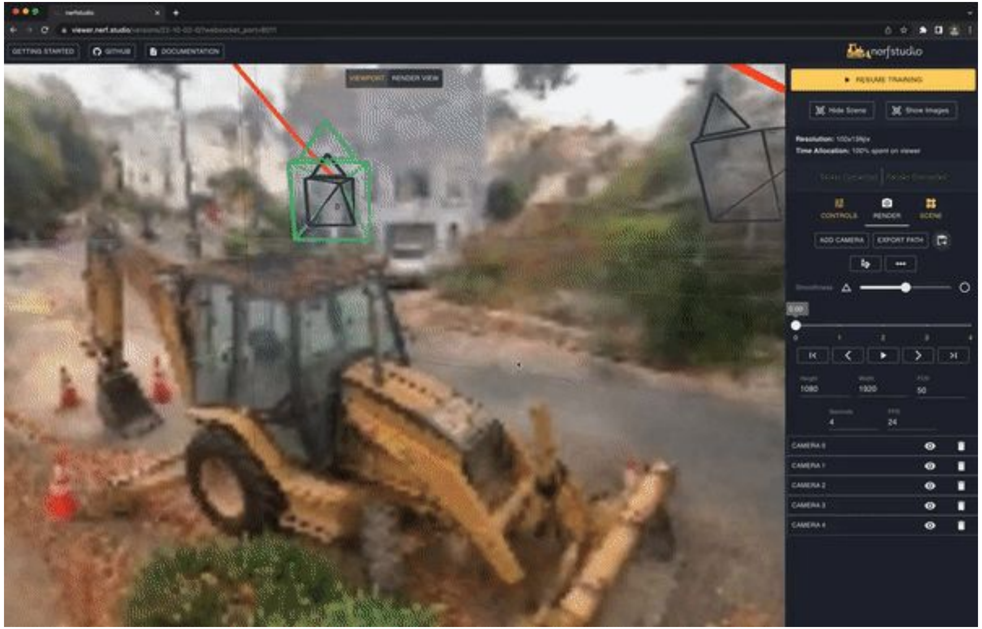
Equal tangent vectors for both segments

Same second derivative at point

Catmull Rom can be used to make nice camera trajectories!

Here's an example with Nerfstudio.

(1) The keyframes we want the camera path to follow.



(2) The code making this possible!

https://threejs.org/docs/#api/en/extras/curves/SplineCurve

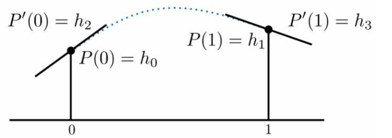
Curve →

SplineCurve

Create a smooth 2d spline curve from a series of points. Internally this uses Interpolations.CatmullRom to create the curve.

(3) The final render!

Math Deepdive: Derivation



$$P(t) = h_0\,H_0(t) + h_1\,H_1(t) + h_2\,H_2(t) + h_3\,H_3(t)$$

Cubic polynomial

$$P(t) = a\,t^3 + b\,t^2 + c\,t + d$$
$$P'(t) = 3a\,t^2 + 2b\,t + c$$

Set up constraint equations

$$P(0) = h_0 = d$$
$$P(1) = h_1 = a + b + c + d$$
$$P'(0) = h_2 = c$$
$$P'(1) = h_3 = 3a + 2b + c$$

Solve for Polynomial Coefficients

$$h_0 = d$$
$$h_1 = a + b + c + d$$
$$h_2 = c$$
$$h_3 = 3a + 2b + c$$

$$\begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

$$P(t) = a\,t^3 + b\,t^2 + c\,t + d$$
$$= \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$
$$= \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

$$P(t) = a\,t^3 + b\,t^2 + c\,t + d$$
$$= \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}$$
$$= \begin{bmatrix} 2t^3 - 3t^2 + 1 \\ -2t^3 + 3t^2 \\ t^3 - 2t^2 + t \\ t^3 - t^2 \end{bmatrix}^T \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

1. List all degree 2 polynomials satisfying: $f(0) = 1$, $f(1) = 2$, $f(2) = 5$.

1. List all degree 2 polynomials satisfying: $f(0) = 1$, $f(1) = 2$, $f(2) = 5$.

Solution: Let $f(t) = at^2 + bt + c$ be the degree 2 polynomial. The system of constraints we get is:

$$\begin{aligned} c &= 1 \\ a + b + c &= 2 \\ 4a + 2b + c &= 5 \end{aligned}$$

Solving the system yields a unique solution $a = 1, b = 0, c = 1$, so $f(t) = t^2 + 1$ is the only degree 2 polynomial.

1. List all degree 2 polynomials satisfying: $f(0) = 1$, $f(1) = 2$, $f(2) = 5$.

2. How many degree 3 polynomials satisfy the given constraints?

1. List all degree 2 polynomials satisfying: $f(0) = 1$, $f(1) = 2$, $f(2) = 5$.

2. How many degree 3 polynomials satisfy the given constraints?

Solution: There are infinitely many such polynomials. Intuitively, three constraints on four coefficients (since a degree 3 polynomial has four coefficients) leave one free parameter, so there must be infinitely many degree 3 solutions.

3. Suppose we have a list of constraints:

$$f(0) = p_0,\ f'(0) = d_0,\ f(1) = p_1,\ f'(1) = d_1,\ \ldots,\ f(k) = p_k,\ f'(k) = d_k.$$

Since we have $2(k+1)$ constraints (one function and one derivative condition per point), the unique interpolating polynomial must have degree $2k+1$.

For a function f, what are the tradeoffs when either

- solving for a single degree $2k+1$ polynomial, versus
- taking the point and derivative constraints at i and $i-1$ for $i = 1, \ldots, k$ and using them to fit k cubic Hermite splines?

3. Suppose we have a list of constraints:

$$f(0) = p_0,\ f'(0) = d_0,\ f(1) = p_1,\ f'(1) = d_1,\ \ldots,\ f(k) = p_k,\ f'(k) = d_k.$$

Since we have $2(k+1)$ constraints (one function and one derivative condition per point), the unique interpolating polynomial must have degree $2k+1$.

For a function f, what are the tradeoffs when either

- solving for a single degree $2k+1$ polynomial, versus
- taking the point and derivative constraints at i and $i-1$ for $i = 1, \ldots, k$ and using them to fit k cubic Hermite splines?

Solution: If we solve for a single high-degree polynomial, it will have infinitely many continuous derivatives. However, it may exhibit unpredictable behavior between the control points, and furthermore, changing a single constraint will affect the entire curve.

If we solve for k cubic Hermite splines, then the resulting curve will be continuous and have a continuous first derivative. The curve will have infinitely many continuous derivatives within each segment but may have a discontinuous second derivative at the control points. However, changing a single constraint will only affect two of the cubic splines: those that have the control point as an endpoint.

4. Consider a cubic polynomial $f(t) = at^3 + bt^2 + ct + d$ that satisfies the following conditions:

$$
\begin{aligned}
f(0) &= f_0, \\
f(1) &= f_1, \\
f''(0) &= f_0'', \\
f''(1) &= f_1''.
\end{aligned}
$$

Write the matrix that, when inverted and applied to the vector $(f_0, f_1, f_0'', f_1'')^T$, allows you to recover the coefficients a, b, c, and d of the polynomial.

4. Consider a cubic polynomial $f(t) = at^3 + bt^2 + ct + d$ that satisfies the following conditions:

$$\begin{aligned} f(0) &= f_0, \\ f(1) &= f_1, \\ f''(0) &= f_0'', \\ f''(1) &= f_1''. \end{aligned}$$

Write the matrix that, when inverted and applied to the vector $(f_0, f_1, f_0'', f_1'')^T$, allows you to recover the coefficients $a, b, c,$ and d of the polynomial.

Solution:

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 2 & 0 & 0 \\ 6 & 2 & 0 & 0 \end{pmatrix}^{-1} \begin{pmatrix} f_0 \\ f_1 \\ f_0'' \\ f_1'' \end{pmatrix}$$

5. Given the numerical inverse of the matrix:

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 0 & 0 & -1/6 & 1/6 \\ 0 & 0 & 1/2 & 0 \\ -1 & 1 & -1/3 & -1/6 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_0'' \\ f_1'' \end{pmatrix}$$

what are the **basis polynomials** that allow expressing $f(t)$ in terms of f_0, f_1, f_0'', and f_1''?

5. Given the numerical inverse of the matrix:

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 0 & 0 & -1/6 & 1/6 \\ 0 & 0 & 1/2 & 0 \\ -1 & 1 & -1/3 & -1/6 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_0'' \\ f_1'' \end{pmatrix}$$

what are the **basis polynomials** that allow expressing $f(t)$ in terms of f_0, f_1, f_0'', and f_1''?

Recall:

$$P(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} H_0(t) & H_1(t) & H_2(t) & H_3(t) \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

t^3	$H_0(t) = 2t^3 - 3t^2 + 1$
t^2	$H_1(t) = -2t^3 + 3t^2$
t	$H_2(t) = t^3 - 2t^2 + t$
1	$H_3(t) = t^3 - t^2$
Basis functions for cubic polynomials	**Hermite basis functions for cubic polynomials**

5. Given the numerical inverse of the matrix:

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 0 & 0 & -1/6 & 1/6 \\ 0 & 0 & 1/2 & 0 \\ -1 & 1 & -1/3 & -1/6 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_0'' \\ f_1'' \end{pmatrix}$$

what are the **basis polynomials** that allow expressing $f(t)$ in terms of f_0, f_1, f_0'', and f_1''?

Recall:

$$P(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} H_0(t) & H_1(t) & H_2(t) & H_3(t) \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

Basis functions for cubic polynomials	Hermite basis functions for cubic polynomials
t^3	$H_0(t) = 2t^3 - 3t^2 + 1$
t^2	$H_1(t) = -2t^3 + 3t^2$
t	$H_2(t) = t^3 - 2t^2 + t$
1	$H_3(t) = t^3 - t^2$

Solution:

The basis polynomials for this problem are:

$$G_0(t) = -t + 1, \quad G_1(t) = t, \quad G_2(t) = -\frac{t^3}{6} + \frac{t^2}{2} - \frac{t}{3}, \quad G_3(t) = \frac{t^3}{6} - \frac{t}{6}.$$

Recall:

$$P(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} H_0(t) & H_1(t) & H_2(t) & H_3(t) \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

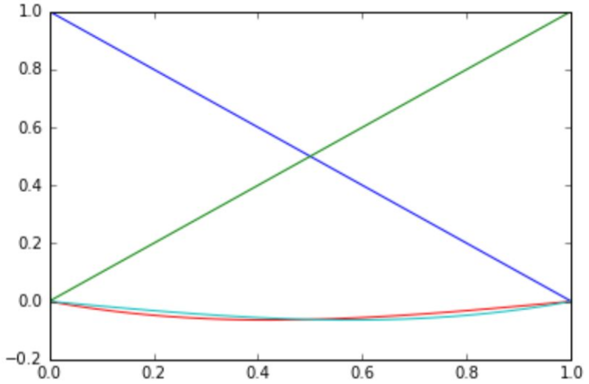
Basis functions for cubic polynomials	Hermite basis functions for cubic polynomials
t^3	$H_0(t) = 2t^3 - 3t^2 + 1$
t^2	$H_1(t) = -2t^3 + 3t^2$
t	$H_2(t) = t^3 - 2t^2 + t$
1	$H_3(t) = t^3 - t^2$

Solution:

The basis polynomials for this problem are:

$$G_0(t) = -t + 1, \quad G_1(t) = t, \quad G_2(t) = -\frac{t^3}{6} + \frac{t^2}{2} - \frac{t}{3}, \quad G_3(t) = \frac{t^3}{6} - \frac{t}{6}.$$

Here's a plot of these functions:



These basis polynomials form a set of functions that allow us to express any cubic polynomial satisfying the given constraints. The interpolating function is given by:

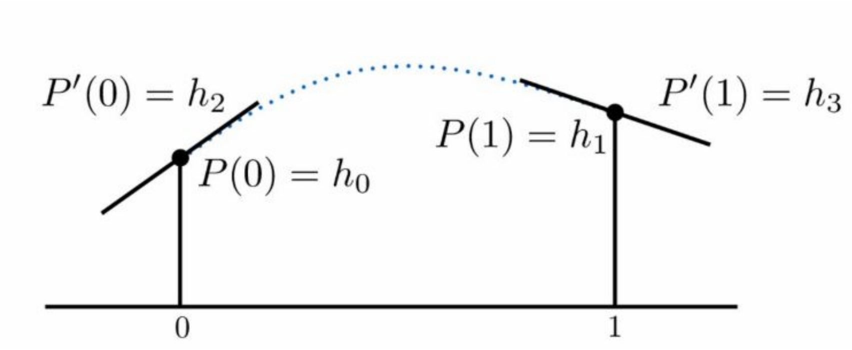
$$f(t) = f_0 G_0(t) + f_1 G_1(t) + f_0'' G_2(t) + f_1'' G_3(t).$$

$f(t)$ matches the given function values and second derivatives at $t = 0$ and $t = 1$.

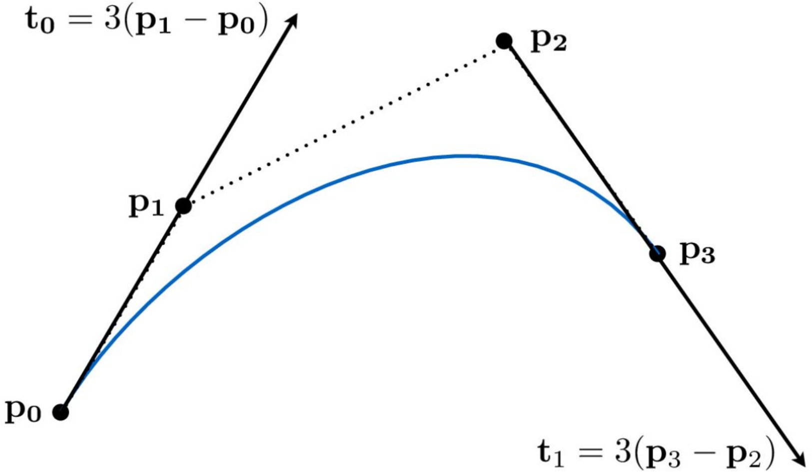
Bezier Curves

Cubic Bezier Take 1: Hermite interpolation

Hermite: Specify derivatives directly

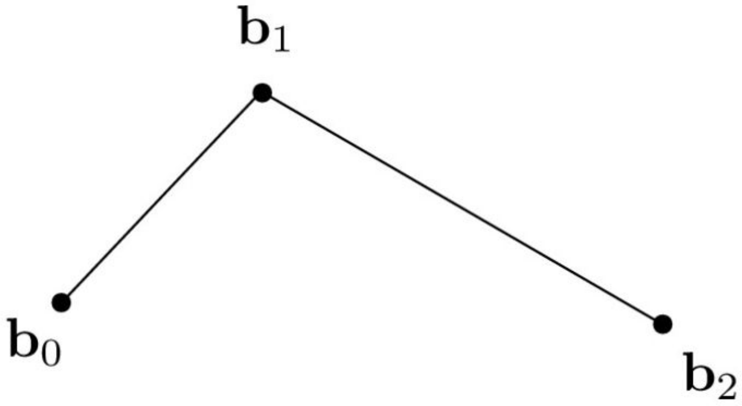


Cubic Bezier: Specify via control points



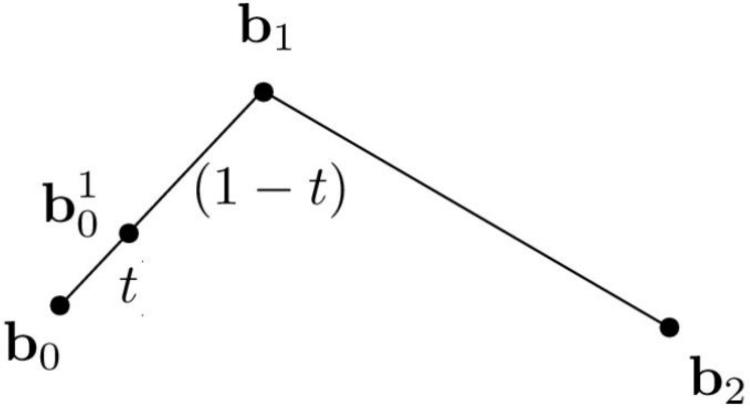
(Brief aside) de Casteljau's algorithm

Example: Three points for quadratic Bezier curves



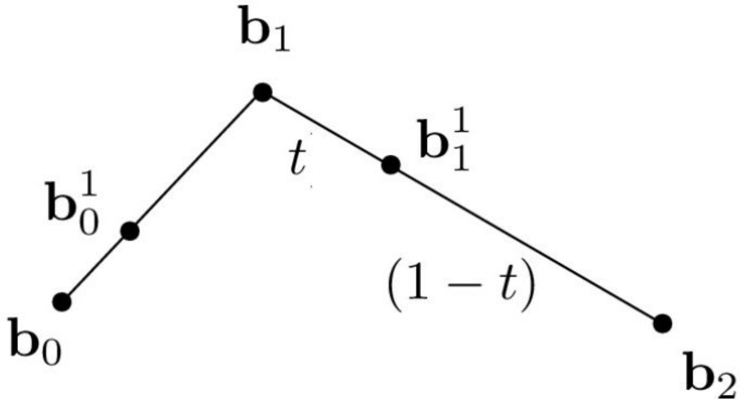
(Brief aside) de Casteljau's algorithm

Example: Three points for quadratic Bezier curves



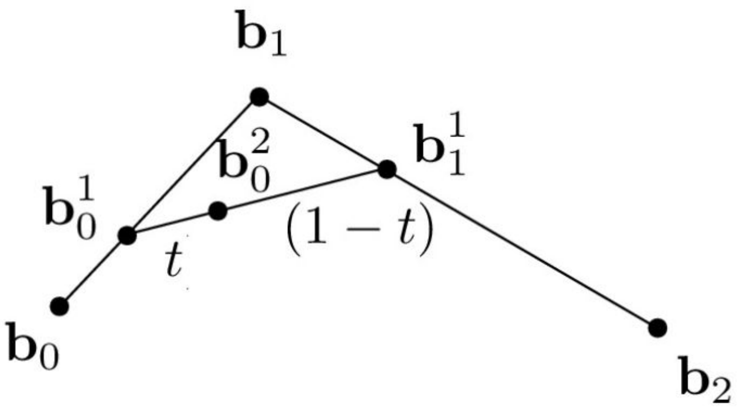
(Brief aside) de Casteljau's algorithm

Example: Three points for quadratic Bezier curves



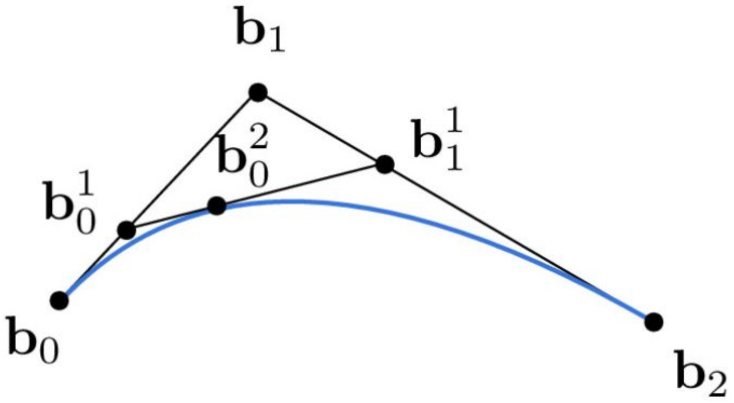
(Brief aside) de Casteljau's algorithm

Example: Three points for quadratic Bezier curves

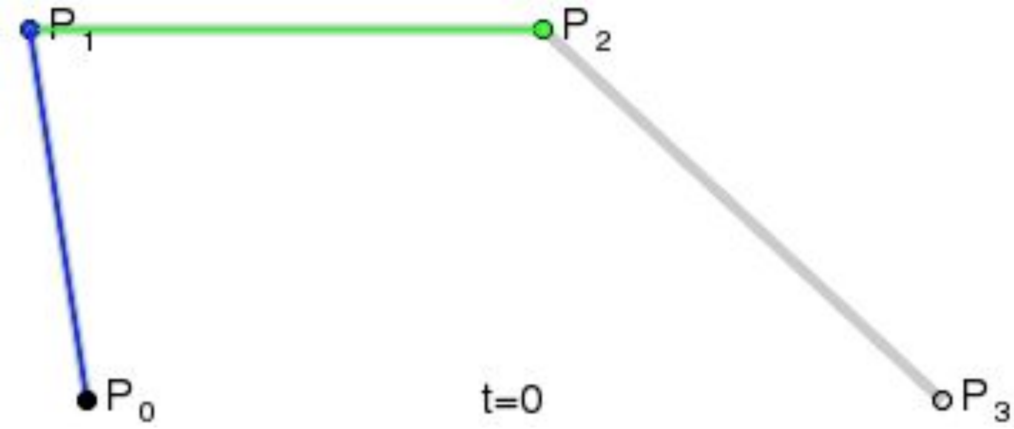


(Brief aside) de Casteljau's algorithm

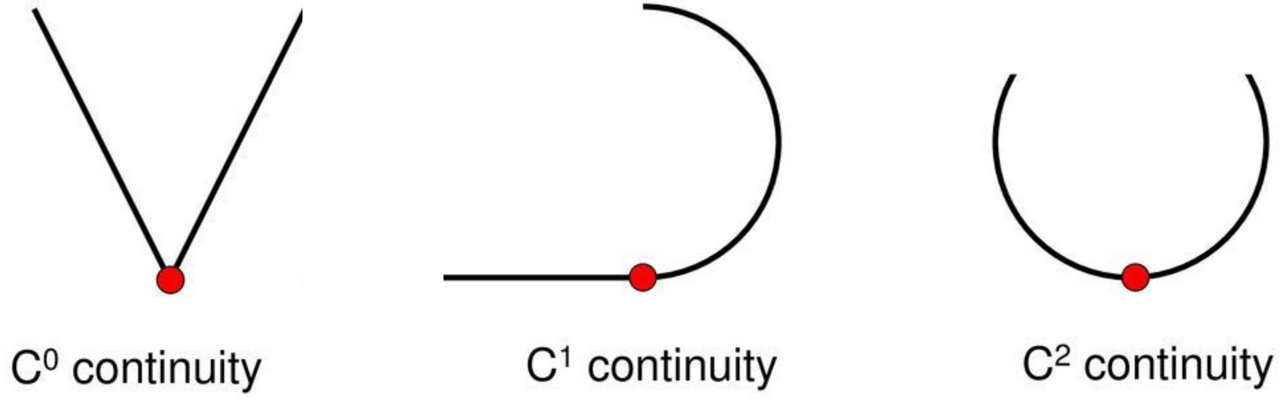
Example: Three points for quadratic Bezier curves



Cubic Bezier Take 2: de Casteljau's algorithm



Splicing curves together



Ends of segments meet

Ends of segments meet

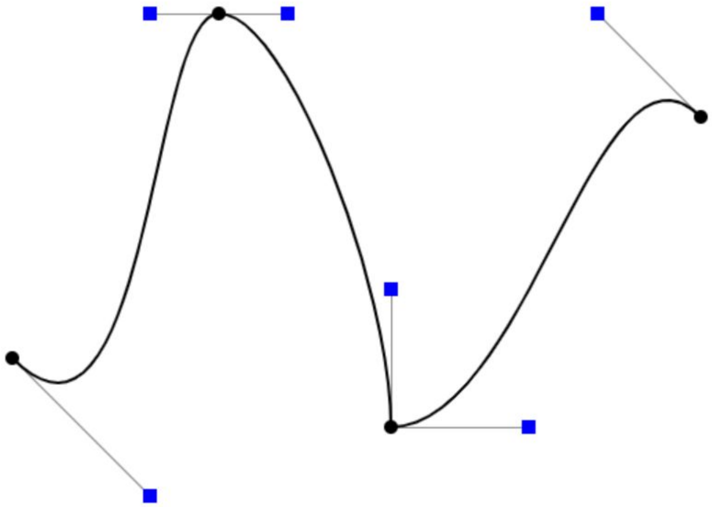
Equal tangent vectors for both segments

Ends of segments meet

Equal tangent vectors for both segments

Same second derivative at point

Piecewise Bezier curves demo



https://math.hws.edu/eck/cs424/notes2013/canvas/bezier.html

Let’s Take Attendance.

- Be sure to select Week 4 and input your TA’s secret word 😉
- Any feedback? Let us know!