CSE 167 (FA21)
Computer Graphics:
Curves
Albert Chern
Vector graphics

Low-level geometry representation
- lines, triangles
- Easy to store, easy to render
- Hard to make, edit, design

High-level representation
- Generates lines & triangles mathematically with small number of control points
- Splines and subdivisions
Curves

Computer fonts
Curves

Design
Curves

Surface of revolution
Curves

Surface patch
Curves

Swept surface / volume
Curves

Key frame animation
Curves

Camera path
Overview

• Parametric curves
  ▶ Curves are functions
• Splines
  ▶ Bézier curves
  ▶ B-spline, NURBS
• Subdivisions
  ▶ Subdivision Bézier curves
• Next: surfaces
Parametric curves
Parametric curves

- A curve is described by a function mapping from 1D to 2D (or 3D)

\[ f: [a, b] \rightarrow \mathbb{R}^2 \]

\[ t \mapsto f(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} \]
Example

- Straight line

\[ f(t) = p_0 + td \]
Example

- Circle

\[ f(t) = \begin{bmatrix} \cos(t) \\ \sin(t) \end{bmatrix} \]
Example

- Circle

\[ f(t) = \begin{bmatrix} \frac{1-t^2}{1+t^2} \\ \frac{2t}{1+t^2} \end{bmatrix} \]

\[
\begin{bmatrix}
\frac{1-t^2}{1+t^2} \\
\frac{2t}{1+t^2}
\end{bmatrix} \sim
\begin{bmatrix}
1 - t^2 \\
2t \\
1 + t^2
\end{bmatrix}
\]
A piecewise defined curve is given by:

\[ f(t) = \begin{cases} 
  f_0(t) & t_0 \leq t < t_1 \\
  f_1(t) & t_1 \leq t < t_2 \\
  f_2(t) & t_2 \leq t \leq t_3 
\end{cases} \]

We may want some continuity condition.
Derivatives of curve

- Position $f(t)$
- Velocity $f'(t)$
- Acceleration $f''(t)$

- If $f'(t) \neq 0$

\[
\frac{f'(t)}{|f'(t)|} \text{ is the tangent}
\]
Continuity of curve

- A curve $f$ is called $C^0$ if $f$ is continuous
- A curve $f$ is called $C^1$ if $f'$ is continuous
- A curve $f$ is called $C^2$ if $f''$ is continuous
- and so on.
- $\cdots \subset C^2 \subset C^1 \subset C^0$
- Do $C^1$, $C^2$, $\cdots$ imply visual smoothness of the curve?
Continuity of curve

- The passenger can feel smooth and continuous acceleration, but the trajectory can still feature cusps and kinks!
Continuity of curve

- The function can be analytic \( f(t) = \begin{bmatrix} t - \sin t \\ 1 - \cos t \end{bmatrix} \)
- But the trajectory is not a smooth curve
Geometric continuity of curve

- To ensure smoothness of curve, add one extra condition: \( f'(t) \neq 0 \)
- Curve with non-vanishing first derivative is called **regular curve**
- A curve \( f \) is called \( G^1 \) if it is \( C^1 \) and \( f'(t) \neq 0 \)  
  - continuous tangent
- A curve \( f \) is called \( G^2 \) if it is \( C^2 \) and \( f'(t) \neq 0 \)  
  - continuous curvature
- and so on
Take away

- Mathematical description of curve
  - A curve is a position-valued function of one variable
- Circles can be drawn using quadratic polynomials with values in homogeneous coordinate
- $C^k$ continuity directly measures the continuity of the $k$-th derivative of the function
- $G^k$ continuity adds an extra condition that the first derivative is non-vanishing
Spline curves
Splines

- The curves (functions) are generated by a small number of “control points” through some algorithm or mathematical formula.
- Known as **spline curves** or **splines**.
The term comes from flexible spline devices used by shipbuilders and draftsmen to draw smooth shapes.

Spline consists of a long strip fixed in a few points that relaxes to form a smooth curve passing through those points.
Polynomial curves

• In computations, the spline curves are almost always parametrized by **polynomials**.

• Evaluation of polynomials only involve addition, subtraction, and multiplication.

  - **Linear** \( f(t) = p_0 + td \)
  
  - **Quadratic** \( f(t) = at^2 + bt + c \)
Polynomial curves

- Quadratic

\[ f(t) = at^2 + bt + c \]

- Quadratic into homogeneous coordinate

\[
\begin{bmatrix}
\frac{1-t^2}{1+t^2} \\
\frac{2t}{1+t^2} \\
1
\end{bmatrix} \sim \begin{bmatrix}
1 - t^2 \\
2t \\
1 + t^2
\end{bmatrix} = \begin{bmatrix}
-1 \\
0 \\
1
\end{bmatrix} t^2 + \begin{bmatrix}
0 \\
1 \\
0
\end{bmatrix} t + \begin{bmatrix}
1
\end{bmatrix}
\]

circle, ellipse, hyperbola
Polynomial curves

- Designs consisting of lines and circular arcs (common in architecture) use quadratic polynomial curves in homogeneous coordinates.
Polynomial curves

- Linear \( f(t) = p_0 + td \)
- Quadratic \( f(t) = at^2 + bt + c \)
- Cubic \( f(t) = at^3 + bt^2 + ct + d \)
Cubic polynomials are approximate solutions to the elastic curves
Spline control

- With higher degree polynomial, we have more variety of curves
- But the behavior of higher degree polynomials are hard to control

Keyword: Runge’s phenomenon
Spline control

- In spline interpolation, we usually use **piecewise** lower order polynomial
  - The behavior of lower order polynomials are easy to predict (most commonly used degree is cubic)
  - Piecewise definition allows large variety of shapes

3 pieces of cubic polynomials
Spline control

Two common methods for controlling splines

- Bézier curve
  - Curve passes through a subset of control points at the junctions

- Basis spline (B-spline)
  - Curve don’t pass through the control points
  - Transition is smoother
Spline control

- $p_0, p_1, p_2, p_3$ control the red curve
- $p_3, p_4, p_5, p_6$ control the blue curve

- $p_0, p_1, p_2, p_3$ control the red curve
- $p_1, p_2, p_3, p_4$ control the blue curve
Two common methods for controlling splines

- Bézier curve
- Basis spline (B-spline)
Bézier Curves
Bézier curves introduction

- A curve drawing algorithm developed by physicist/mathematician Paul de Casteljau in 1959.
  - Higher order extension of linear interpolation
- Popularized by Pierre Bézier (1960s) for designing curves for the bodywork of Renault cars.
  - Provides intuitive control over curve with control points
  - Endpoints are interpolated (passed through by the curve)
  - Intermediate points are approximated
Common usage

- Common usage: cubic Bézier curve
  - 4 control points

blogs.sitepointstatic.com/examples/tech/canvas-curves/bezier-curve.html
Bézier curves overview

- **De Casteljau algorithm**
  - Algorithmic introduction (intuitive, historical)

- **Bernstein polynomial**
  - Explicit formula (analysis)

- **Cubic Blossom**
  - Mathematical abstraction (for generalization)
De Casteljau algorithm

• Recursive linear interpolation

• Let us abbreviate linear interpolation between two points by “lerp”

```cpp
point lerp(point p, point q, float t ){
    return (1 − t)p + tq;
}
```

• In GLSL, GLM, the lerp function is called “mix” (glm::mix)

De Casteljau algorithm

- Given
  - a few (say 4) points \( p_0, p_1, p_2, p_3 \)
  - a value \( t \) between 0 and 1. For example \( t = 0.25 \)
De Casteljau algorithm

• Given
  ▶ a few (say 4) points $p_0, p_1, p_2, p_3$
  ▶ a value $t$ between 0 and 1. For example $t = 0.25$

• Linear interpolate

$q_0(t) = \text{lerp}(p_0, p_1, t)$
$q_1(t) = \text{lerp}(p_1, p_2, t)$
$q_2(t) = \text{lerp}(p_2, p_3, t)$
De Casteljau algorithm

\[
\begin{align*}
  r_0(t) &= \operatorname{lerp}(q_0, q_1, t) \\
  r_1(t) &= \operatorname{lerp}(q_1, q_2, t)
\end{align*}
\]
De Casteljau algorithm

\[ f(t) = \text{lerp}(r_0, r_1, t) \]

https://www.jasondavies.com/animated-bezier/
How do we translate the procedure to code?
De Casteljau algorithm

- Algorithmically, given an array of points, and a value $t$
De Casteljau algorithm

- Algorithmically, given an array of points, and a value $t$
- Repeatedly lerp and replace the original point (save memory)

$q_0 = \text{lerp}(p_0, p_1, t)$
De Casteljau algorithm

- Algorithmically, given an array of points, and a value $t$
- Repeatedly lerp and replace the original point (save memory)

$q_0 \rightarrow p_1 \rightarrow p_2 \rightarrow p_3$

$q_1 = \text{lerp}(p_1, p_2, t)$
De Casteljau algorithm

- Algorithmically, given an array of points, and a value $t$
- Repeatedly lerp and replace the original point (save memory)

$q_2 = \text{lerp}(p_2, p_3, t)$
De Casteljau algorithm

- Algorithmically, given an array of points, and a value $t$
- Repeatedly lerp and replace the original point (save memory)
De Casteljau algorithm

- Algorithmically, given an array of points, and a value $t$
- Repeatedly lerp and replace the original point (save memory)

$q_0 \quad q_1 \quad q_2 \quad p_3$

$r_0 = \text{lerp}(q_0, q_1, t)$
De Casteljau algorithm

- Algorithmically, given an array of points, and a value $t$
- Repeatedly lerp and replace the original point (save memory)

$$r_1 = \text{lerp}(q_1, q_2, t)$$
De Casteljau algorithm

- Algorithmically, given an array of points, and a value $t$
- Repeatedly lerp and replace the original point (save memory)
De Casteljau algorithm

- Algorithmically, given an array of points, and a value $t$
- Repeatedly lerp and replace the original point (save memory)

\[
\begin{align*}
    & \quad r_0 \quad r_1 \quad q_2 \quad p_3 \\
    f &= \text{lerp}(r_0, r_1, t)
\end{align*}
\]
De Casteljau algorithm

• Algorithmically, given an array of points, and a value $t$
• Repeatedly lerp and replace the original point (save memory)
De Casteljau pseudocode

```cpp
Point deCasteljau( std::vector<Point> p_input, float t )
{
    std::vector<Point> p = p_input; // make a copy
    int n = the array size of p
    for i = 1,…, n-1
        for j = 1,…, n-i
            p[ j ] = lerp(p[ j ], p[ j+1 ], t )
    end for
    end for
    return p[ 0 ]
}
```
Bézier curves overview

- **De Casteljau algorithm**
  - Algorithmic introduction (intuitive, historical)

- **Bernstein polynomial**
  - Explicit formula (analysis)

- **Cubic Blossom**
  - Mathematical abstraction (for generalization)
Bézier Curves
Bernstein polynomials
Bézier curve explicit form

• Call $s = 1 - t$, so $\text{lerp}(p_0, p_1, t) = sp_0 + tp_1$

\[
\begin{align*}
    p_0 & \quad \rightarrow \quad sp_0 + tp_1 \\
    p_1 & \quad \rightarrow \quad sp_1 + tp_2 \\
    p_2 & \quad \rightarrow \quad sp_2 + tp_3 \\
    p_3 & \quad \rightarrow \quad s^2p_0 + 2stp_1 + t^2p_2 \\
    & \quad \rightarrow \quad s^2p_1 + 2stp_2 + t^2p_3 \\
    & \quad \rightarrow \quad s^3p_0 + 3s^2tp_1 + 3st^2p_2 + t^3p_3
\end{align*}
\]
Bézier curve explicit form

- Call \( s = 1 - t \), so \( \text{lerp}(\mathbf{p}_0, \mathbf{p}_1, t) = s\mathbf{p}_0 + t\mathbf{p}_1 \)
- In general, the Bézier curve constructed from \( \mathbf{p}_0, \ldots, \mathbf{p}_n \) is given by

\[
f(t) = \sum_{k=0}^{n} \binom{n}{k} s^k t^{n-k} \mathbf{p}_k = \sum_{k=0}^{n} \binom{n}{k} (1 - t)^k t^{n-k} \mathbf{p}_k \]

\[
\mathbf{B}_k^{(n)}(t) \quad \text{Bernstein polynomial}
\]
Bézier curve explicit form

\[ f(t) = \sum_{k=0}^{n} \binom{n}{k} (1 - t)^k t^{n-k} p_k \]

\[ \binom{n}{k} = \frac{n!}{k!(n-k)!} \]

\[ B_k^{(n)}(t) \] Bernstein polynomial

- For \( n = 1 \) (2 points)

\[ f(t) = B_0^{(1)}(t) p_0 + B_1^{(1)}(t) p_1 \]

\[ B_0^{(1)}(t) = -t + 1 \]

\[ B_1^{(1)}(t) = t \]
Bézier curve explicit form

- For \( n = 2 \) (3 points)

\[
f(t) = B_0^{(2)}(t)p_0 + B_1^{(2)}(t)p_1 + B_2^{(2)}(t)p_2
\]

\[
B_0^{(2)}(t) = t^2 - 2t + 1
\]

\[
B_1^{(2)}(t) = -2t^2 + 2t
\]

\[
B_2^{(2)}(t) = t^2
\]
Bézier curve explicit form

- For \( n = 3 \) (4 points)

\[
f(t) = B^{(3)}_0(t)p_0 + B^{(3)}_1(t)p_1 + B^{(3)}_2(t)p_2 + B^{(3)}_3(t)p_3
\]

\[
B^{(3)}_0(t) = -t^3 + 3t^2 - 3t + 1
\]
\[
B^{(3)}_1(t) = 3t^3 - 6t^2 + 3t
\]
\[
B^{(3)}_2(t) = -3t^3 + 3t^2
\]
\[
B^{(3)}_3(t) = t^3
\]
Bézier curve matrix form

- For \( n = 3 \) (4 points)

\[
f(t) = B^{(3)}_0(t)p_0 + B^{(3)}_1(t)p_1 + B^{(3)}_2(t)p_2 + B^{(3)}_3(t)p_3
\]

\[
\begin{align*}
B^{(3)}_0(t) &= -t^3 + 3t^2 - 3t + 1 & \text{linear combination of } t^3, t^2, t, 1 \\
B^{(3)}_1(t) &= 3t^3 - 6t^2 + 3t \\
B^{(3)}_2(t) &= -3t^3 + 3t^2 \\
B^{(3)}_3(t) &= t^3
\end{align*}
\]
Bézier curve matrix form

For \( n = 3 \) (4 points)

\[
f(t) = B_0^{(3)}(t)p_0 + B_1^{(3)}(t)p_1 + B_2^{(3)}(t)p_2 + B_3^{(3)}(t)p_3
\]

\[
B_0^{(3)}(t) = -t^3 + 3t^2 - 3t + 1
\]

\[
B_1^{(3)}(t) = 3t^3 - 6t^2 + 3t
\]

\[
B_2^{(3)}(t) = -3t^3 + 3t^2
\]

\[
B_3^{(3)}(t) = t^3
\]
Bézier curve matrix form

\[
f(t) = B^{(3)}_0(t)p_0 + B^{(3)}_1(t)p_1 + B^{(3)}_2(t)p_2 + B^{(3)}_3(t)p_3
\]

\[
\begin{bmatrix}
B^{(3)}_0(t) & B^{(3)}_1(t) & B^{(3)}_2(t) & B^{(3)}_3(t)
\end{bmatrix}
= \begin{bmatrix}
t^3 & t^2 & t & 1
\end{bmatrix}
\begin{bmatrix}
-1 & 3 & -3 & 1 \\
3 & -6 & 3 & 0 \\
-3 & 3 & 0 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

Bernstein basis

\[
f(t) = \begin{bmatrix}
B^{(3)}_0(t) & B^{(3)}_1(t) & B^{(3)}_2(t) & B^{(3)}_3(t)
\end{bmatrix}
\begin{bmatrix}
p_0 \\
p_1 \\
p_2 \\
p_3
\end{bmatrix}
\]

monomial basis

\[
= \begin{bmatrix}
t^3 & t^2 & t & 1
\end{bmatrix}
\begin{bmatrix}
-1 & 3 & -3 & 1 \\
3 & -6 & 3 & 0 \\
-3 & 3 & 0 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
p_0 \\
p_1 \\
p_2 \\
p_3
\end{bmatrix}
\]

transformation between bases
### Bézier curve matrix form

- **Writing out coordinates**

\[
f(t) = \begin{bmatrix} x(t) & y(t) \end{bmatrix} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}
\]

**Polynomial basis**

**Spline matrix**

**Geometry matrix**

(contains control points)

- In applications where \( n \) is fixed, we can just hard-code the spline matrix and evaluate Bézier curve directly
Bézier curves overview

- **De Casteljau algorithm**
  - Algorithmic introduction (intuitive, historical)
- **Bernstein polynomial**
  - Explicit formula (analysis)
- **Cubic Blossom**
  - Mathematical abstraction (for generalization)
Bézier Curves
Blossom
De Casteljau algorithm

- Recall that the de Casteljau algorithm is made of several lerps
- “Blossom” is to reverse-engineer a polynomial curve back to its control points
- Blossom re-describes de Casteljau algorithm using simple principles
• Let us focus on Bézier curve with $n=3$ (4 control points)

• Let $f(t)$ be a cubic polynomial in $t$ and take values in some affine space
The **cubic blossom** of cubic polynomial $f$ is a function of 3 variables $F(t_1, t_2, t_3)$ satisfying the properties:

- **Symmetric:** $F(t_1, t_2, t_3) = F(t_2, t_3, t_1) = F(t_3, t_1, t_2) = F(t_2, t_1, t_3) = F(t_3, t_2, t_1) = F(t_1, t_3, t_2)$
- **Tri-affine:** $F(t_1, t_2, t_3)$ is an affine function of $t_1$
  \[ F(\text{lerp}(a, b, \lambda), t_2, t_3) = \text{lerp}(F(a, t_2, t_3), F(b, t_2, t_3), \lambda) \]
- **Diagonal consistency:** $F(t, t, t) = f(t)$
Blossom

- **Symmetric:** $F(t_1, t_2, t_3) = F(t_2, t_3, t_1) = F(t_3, t_1, t_2)$
  
  \[= F(t_2, t_1, t_3) = F(t_3, t_2, t_1) = F(t_1, t_3, t_2)\]

- **Tri-affine:** $F(t_1, t_2, t_3)$ is an affine function of $t_1$

- **Diagonal consistency:** $F(t, t, t) = f(t)$

- Example $f(t) = t^3$

  \[F(t_1, t_2, t_3) = t_1 t_2 t_3\]
Symmetric: \( F(t_1, t_2, t_3) = F(t_2, t_3, t_1) = F(t_3, t_1, t_2) \)

\[ = F(t_2, t_1, t_3) = F(t_3, t_2, t_1) = F(t_1, t_3, t_2) \]

Tri-affine: \( F(t_1, t_2, t_3) \) is an affine function of \( t_1 \)

Diagonal consistency: \( F(t, t, t) = f(t) \)

Example \( f(t) = t^2 \)

\[ F(t_1, t_2, t_3) = \frac{1}{3}(t_1 t_2 + t_2 t_3 + t_1 t_3) \]
Symmetric:  \[ F(t_1, t_2, t_3) = F(t_2, t_3, t_1) = F(t_3, t_1, t_2) \]
\[ = F(t_2, t_1, t_3) = F(t_3, t_2, t_1) = F(t_1, t_3, t_2) \]

Tri-affine:  \[ F(t_1, t_2, t_3) \] is an affine function of \( t_1 \)

Diagonal consistency:  \[ F(t, t, t) = f(t) \]

Example  \( f(t) = t \)

\[ F(t_1, t_2, t_3) = \frac{1}{3}(t_1 + t_2 + t_3) \]
Symmetric:  \[ F(t_1, t_2, t_3) = F(t_2, t_3, t_1) = F(t_3, t_1, t_2) \]
\[ = F(t_2, t_1, t_3) = F(t_3, t_2, t_1) = F(t_1, t_3, t_2) \]

Tri-affine:  \[ F(t_1, t_2, t_3) \] is an affine function of \( t_1 \)

Diagonal consistency:  \[ F(t, t, t) = f(t) \]

Example  \[ f(t) = 1 \]

\[ F(t_1, t_2, t_3) = 1 \]
Blossom

- In general $f(t) = at^3 + bt^2 + ct + d$

$$F(t_1, t_2, t_3) = at_1t_2t_3 + b\frac{t_1t_2 + t_2t_3 + t_1t_3}{3} + c\frac{t_1 + t_2 + t_3}{3} + d$$
De Casteljau revisited

• The control points are

\[ F(0, 0, 0) = p_0 \]
\[ F(0, 0, 1) = p_1 \]
\[ F(0, 1, 1) = p_2 \]
\[ F(1, 1, 1) = p_3 \]

• Goal: Find \( F(t, t, t) = f(t) \)
We write $F_{abc} = F(a, b, c)$ for short.

We draw a dashed line between two points whenever *all but one* indices agree.

Dashed line indicates lerp-ability

\[
F_{abc} \quad F_{dba} = F_{abd}
\]

\[
F_{ab} \text{lerp}(c, d, t) = \text{lerp}(F_{abc}, F_{dba}, t)
\]
De Casteljau revisited

\[ F_{00t} = \text{lerp}(F_{000}, F_{001}, t) \]
\[ F_{0t1} = \text{lerp}(F_{001}, F_{011}, t) \]
\[ F_{t11} = \text{lerp}(F_{011}, F_{111}, t) \]
De Casteljau revisited

\[ F_{0tt} = \text{lerp}(F_{00t}, F_{0t1}, t) \]

\[ F_{tt1} = \text{lerp}(F_{0t1}, F_{t11}, t) \]
De Casteljau revisited

\[ F_{ttt} = \text{lerp}(F_{0tt}, F_{tt1}, t) \]
De Casteljau revisited

- We easily see which points are passed by the curve (here, $F_{000}, F_{111}$).
- We don’t need to start with $F_{000}, F_{001}, F_{011}, F_{111}$.

Starting with $F_{000}, F_{00\frac{1}{2}}, F_{0\frac{1}{2}1\frac{1}{2}}, F_{\frac{1}{2}2\frac{1}{2}}$ (as long as they form a chain of dashed lines) will yield the same result $F_{ttt} = f(t)$. 

We easily see which points are passed by the curve (here, $F_{000}, F_{111}$).

We don’t need to start with $F_{000}, F_{001}, F_{011}, F_{111}$.

Starting with $F_{000}, F_{00\frac{1}{2}}, F_{0\frac{1}{2}1\frac{1}{2}}, F_{\frac{1}{2}2\frac{1}{2}}$ (as long as they form a chain of dashed lines) will yield the same result $F_{ttt} = f(t)$.
Recap

- **De Casteljau algorithm**
  Repeated linear interpolation

- **Bernstein polynomial**
  You can write down explicit formula

- **Blossom**
  De Casteljau revised with symmetric tri-affine forms
Two common methods for controlling splines

- Bézier curve
- Basis spline (B-spline)
B-Spline
The general lerp

- The lerp notation only interpolates between 0 and 1

\[ \text{lerp}(p, q, t) = (1 - t)p + tq \]

- For general range (recall barycentric coord.)

\[ \text{lerp}_{a,b}(p, q, t) = \frac{b - t}{b - a} p + \frac{t - a}{b - a} q \]
Cubic B-spline

- Given \((n+1)\) control points \(p_0, \ldots, p_n\)
- We build a function \(f(t)\) over \(t \in [1, n]\)
  consisting of \((n-1)\) segments

\[
f(t) = \begin{cases} 
  f_1(t) & 1 \leq t < 2 \\
  f_2(t) & 2 \leq t < 3 \\
  \vdots \\
  f_{n-1}(t) & n-1 \leq t \leq n 
\end{cases}
\]
Cubic B-spline

\[ f(t) = \begin{cases} f_1(t) & 1 \leq t < 2 \\ f_2(t) & 2 \leq t < 3 \\ \vdots \\ f_{n-1}(t) & n-1 \leq t \leq n \end{cases} \]

- Label
  - \( F_{-1,0,1} = p_0 \)
  - \( F_{0,1,2} = p_1 \)
  - \( F_{1,2,3} = p_2 \)
  - \( \vdots \)
  - \( F_{n-1,n,n+1} = p_n \)

- For \( k \leq t < k + 1 \) we construct \( f_k(t) = F_{ttt} \) using de Casteljau algorithm from the blossom points

\[
\begin{align*}
F_{k-2,k-1,k} &= F_{k-1,k,k+1} \\
F_{k,k+1,k+2} &= F_{k+1,k+2,k+3}
\end{align*}
\]
Cubic B-spline

- For simplicity of exposition, assume $2 \leq t < 3$
- Compute $f_2(t)$ given $F_{012}, F_{123}, F_{234}, F_{345}$
- You can write a function
  
  ```
  Point BSplineHelper(Point F012, F123, F234, F345; float t);
  ```

- and use it like
  
  ```
  k = FLOOR(t);
  f(t) = BSPLINEHELPER(p[k - 1], p[k], p[k + 1], p[k + 2], t - k + 2);
  ```
Cubic B-spline

- Compute $f_2(t)$ given $F_{012}, F_{123}, F_{234}, F_{345}$

$$
F_{012} \quad \overset{\text{lerp}_{0,3}}{\rightarrow} \quad F_{123} \quad \overset{\text{lerp}_{1,4}}{\rightarrow} \quad F_{234} \quad \overset{\text{lerp}_{2,5}}{\rightarrow} \quad F_{345}
$$

$$
F_{12t} \quad \overset{\text{lerp}_{1,3}}{\rightarrow} \quad F_{23t} \quad \overset{\text{lerp}_{2,4}}{\rightarrow} \quad F_{34t}
$$

$$
F_{2tt} \quad \overset{\text{lerp}_{2,3}}{\rightarrow} \quad F_{3tt}
$$

$$
F_{ttt} = f_2(t)
$$
B-spline matrix form

Shift \( \tau = t - 2 \quad (0 \leq \tau \leq 1) \)

\[
f_2(t) = \begin{bmatrix} \tau^3 & \tau^2 & \tau & 1 \end{bmatrix} \begin{bmatrix} -\frac{1}{6} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{6} \\ \frac{1}{2} & -1 & \frac{1}{2} & 0 \\ -\frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}
\]

\[
= \begin{bmatrix} b_1(\tau) & b_2(\tau) & b_3(\tau) & b_4(\tau) \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}
\]
B-spline basis

\[ f_2(t) = b_1(t-2)p_1 + b_2(t-2)p_2 + b_3(t-2)p_3 + b_4(t-2)p_4 \]
\[ f_3(t) = b_1(t-3)p_2 + b_2(t-3)p_3 + b_3(t-3)p_4 + b_4(t-3)p_5 \]
B-spline basis

\[ f_2(t) = b_1(t - 2)p_1 + b_2(t - 2)p_2 + b_3(t - 2)p_3 + b_4(t - 2)p_4 \]
\[ f_3(t) = b_1(t - 3)p_2 + b_2(t - 3)p_3 + b_3(t - 3)p_4 + b_4(t - 3)p_5 \]
B-spline basis

\[ f_2(t) = b_1(t - 2)p_1 + b_2(t - 2)p_2 + b_3(t - 2)p_3 + b_4(t - 2)p_4 \]

\[ f_3(t) = b_1(t - 3)p_2 + b_2(t - 3)p_3 + b_3(t - 3)p_4 + b_4(t - 3)p_5 \]
B-spline basis

- $N(t)$ is a piecewise cubic polynomial (on each integer interval)
- $N(t)$ is $C^2$
- $N(t)$ is nonnegative and is an even function (left-right symmetric)
- $N(t)$ is zero outside $[-2,2]$
- $\sum_{i=-\infty}^{\infty} N(t - i) = 1$ for all $t$
B-spline basis

\[ f(t) = \sum_{i} N(t - i) p_i \]
B-spline recap

- B-spline are generated by de Casteljau algorithm when the control points are labeled as (0,1,2), (1,2,3), (2,3,4), etc.
- The spline generally doesn’t pass through the control points.
- The spline can be understood as combining shifted basis functions:

\[ f(t) = \sum_i N(t - i)p_i \]
• A generalization of B-spline is **Non-Uniform Rational B-Spline**

• Non-uniform: control points are labeled with irregular indices such as (1,2,5), (2,5,7), (5,7,8), ...

• Rational: Values of the curve are in homogeneous coordinate (rather than affine points). So after dehomogenization we get rational functions.

• Rational is important to generate circles

\[
\begin{bmatrix}
\frac{1-t^2}{1+t^2} \\
\frac{2t}{1+t^2} \\
\frac{1-t^2}{1+t^2}
\end{bmatrix}
\sim
\begin{bmatrix}
1-t^2 \\
2t \\
1+t^2
\end{bmatrix}
= \begin{bmatrix}
-1 \\
0 \\
1
\end{bmatrix} t^2 + \begin{bmatrix}
0 \\
1 \\
0
\end{bmatrix} t + \begin{bmatrix}
1 \\
0 \\
1
\end{bmatrix}
\]

• NURBS is a common tool in computer-aided design
Subdivision
Change of control points

- Consider de Casteljau algorithm with $t=0.5$
Change of control points

- Consider de Casteljau algorithm with $t=0.5$
- $F_{000}$, $F_{011}$ are control points for left half
- $F_{0\frac{1}{2}\frac{1}{2}}$, $F_{0\frac{1}{2}1}$, $F_{01\frac{1}{2}}$, $F_{111}$ are control points for right half
Change of control points

• Consider de Casteljau algorithm with $t=0.5$
• $F_{000} \ F_{00\frac{1}{2}} \ F_{0\frac{1}{2}\frac{1}{2}} \ F_{\frac{1}{2}\frac{1}{2}\frac{1}{2}}$ are control points for left half
• $F_{\frac{1}{2}\frac{1}{2}\frac{1}{2}} \ F_{\frac{1}{2}\frac{1}{2}\frac{1}{2}} \ F_{\frac{1}{2}1 \frac{1}{2}} \ F_{1\frac{1}{2}} \ F_{111}$ are control points for right half

• Subdivide control points from $F_{000} F_{001} F_{011} F_{111}$ to the finer control points (and still represent the same spline curve).
Change of control points

Original control points

New control points
Spline v.s. Subdivision

- **Spline**
  
  Directly evaluate $f(t)$ for each $t$

- **Subdivision**
  
  Instead of drawing $f(t)$, we just draw the control polygon.

  A subdivision is a refinement of the control polygon.

  Under recursive subdivision, the refined control polygons converge to the spline $f(t)$
Subdivision for B-Splines

1. Insert the edge midpoints

2. Update the vertices by the average between the old vertex position and the average of the neighboring midpoints.