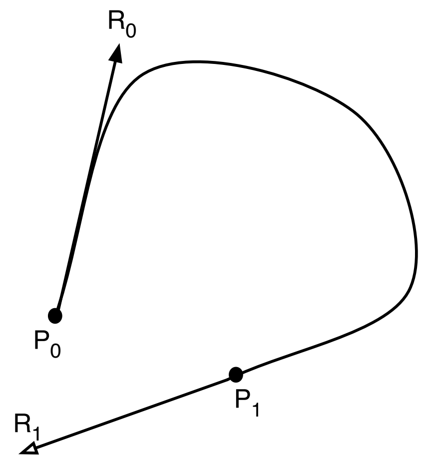
Hermite and Bézier Curves and Surfaces

CS/CptS 442/542

November 26, 2007

Hermite Curve



- Interpolates endpoints: P_0, P_1
- Matches tangent vectors at endpoints: R_0, R_1
- Parametric curve using cubic polynomials:

$$\mathbf{C}(t) \;=\; \begin{bmatrix} a_0 + a_1 t + a_2 t^2 + a_3 t^3 \\ b_0 + b_1 t + b_2 t^2 + b_3 t^3 \\ c_0 + c_1 t + c_2 t^2 + c_3 t^3 \end{bmatrix} \qquad t \in [0, 1]$$

Hermite Curve Constraints

$$\mathbf{C}(t) = \begin{bmatrix} a_0 + a_1 t + a_2 t^2 + a_3 t^3 \\ b_0 + b_1 t + b_2 t^2 + b_3 t^3 \\ c_0 + c_1 t + c_2 t^2 + c_3 t^3 \end{bmatrix} \quad t \in [0,1]$$

$$\mathbf{C}'(t) = \begin{bmatrix} a_1 + 2a_2 t + 3a_3 t^2 \\ b_1 + 2b_2 t + 3b_3 t^2 \\ c_1 + 2c_2 t + 3c_3 t^2 \end{bmatrix}.$$

- Interpolates first endpoint: $\mathbf{C}(0) = \mathbf{P}_0$,
- Interpolates second endpoint:$\mathbf{C}(1) = \mathbf{P}_1$,
- Tangent match at first endpoint: $\mathbf{C}'(0) = \mathbf{R}_0$,
- Tangent match at second endpoint: $\mathbf{C}'(1) = \mathbf{R}_1$.

Solving for the Hermite Coefficients

System of equations from constraints:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} a_0 & b_0 & c_0 \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} = \begin{bmatrix} P_{0x} & P_{0y} & P_{0z} \\ P_{1x} & P_{1y} & P_{1z} \\ R_{0x} & R_{0y} & R_{0z} \\ R_{1x} & R_{1y} & R_{1z} \end{bmatrix}$$

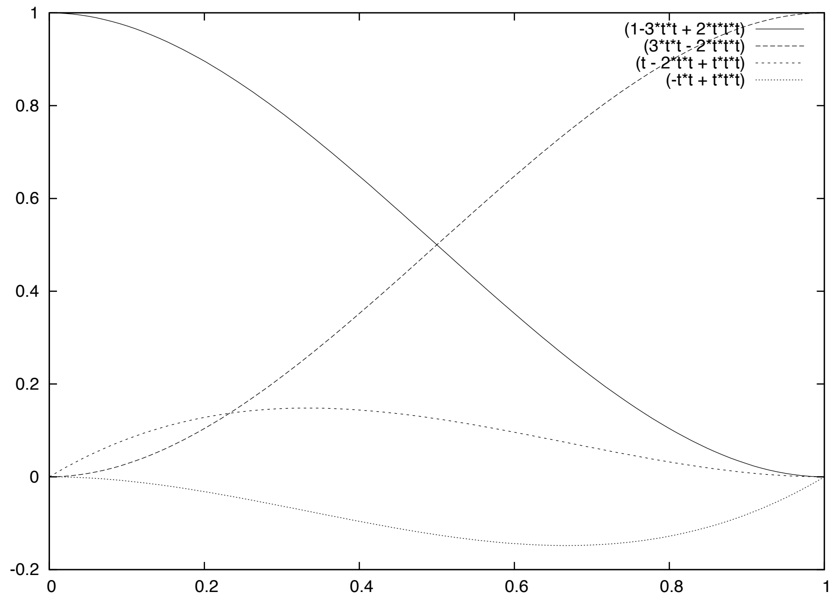
Solve via matrix inversion:

$$\begin{bmatrix} a_0 & b_0 & c_0 \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \begin{bmatrix} P_{0x} & P_{0y} & P_{0z} \\ P_{1x} & P_{1y} & P_{1z} \\ R_{0x} & R_{0y} & R_{0z} \\ R_{1x} & R_{1y} & R_{1z} \end{bmatrix}$$

Resulting curve:

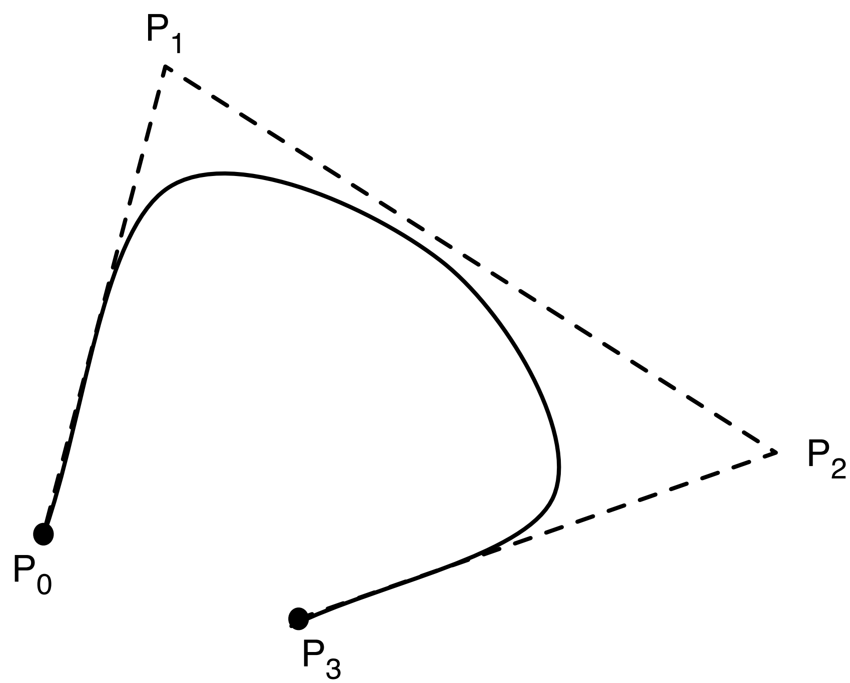
$$\begin{aligned} \mathbf{C}(t) &= (1 - 3t^2 + 2t^3)\mathbf{P}_0 + (3t^2 - 2t^3)\mathbf{P}_1 + \\ &\quad (t - 2t^2 + t^3)\mathbf{R}_0 + (-t^2 + t^3)\mathbf{R}_1. \end{aligned}$$

Hermite Blending Functions



$$\begin{aligned}\mathbf{C}(t) \;&=\; (1-3t^2+2t^3)\mathbf{P}_0 + (3t^2-2t^3)\mathbf{P}_1 + \\ &\quad\; (t-2t^2+t^3)\mathbf{R}_0 + (-t^2+t^3)\mathbf{R}_1.\end{aligned}$$

Cubic Bézier Curve



- Four **control points**: $\{P_0, P_1, P_2, P_3\}$
- Interpolates endpoints: P_0, P_3
- Tangent vectors at endpoints:
  $R_0 = 3(P_1 - P_0)$, $R_1 = 3(P_3 - P_2)$

Cubic Bézier Curve

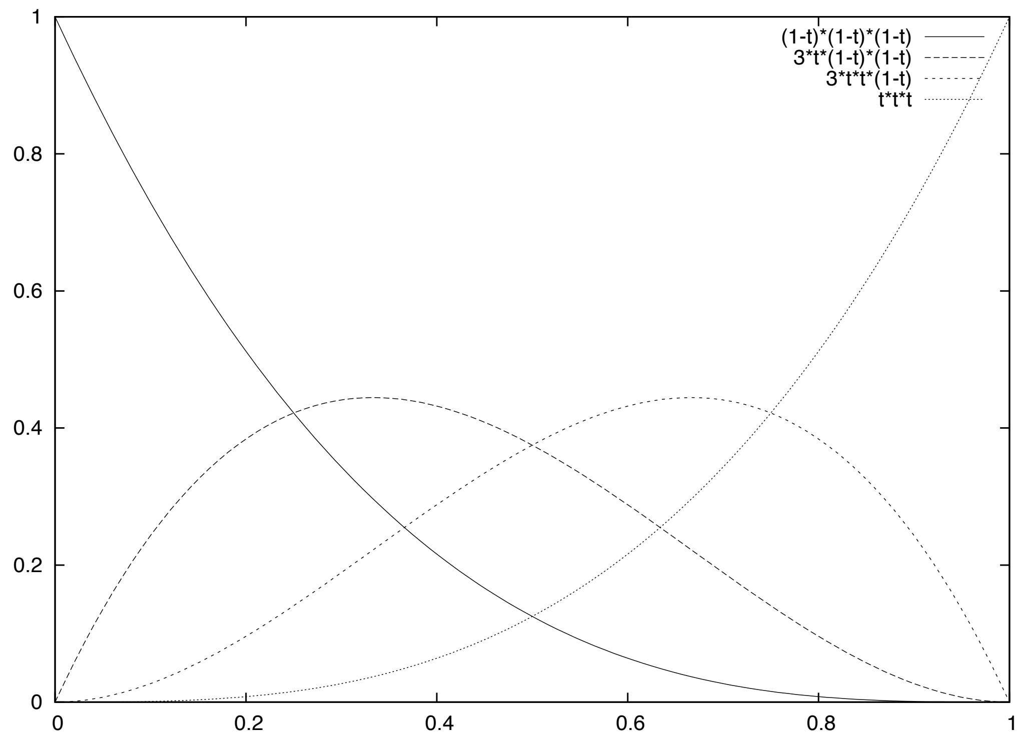
The resulting curve:

$$\mathbf{C}(t) = B_0(t)\mathbf{P}_0 + B_1(t)\mathbf{P}_1 + B_2(t)\mathbf{P}_2 + B_3(t)\mathbf{P}_3$$

The blending functions are the cubic **Bernstein Polynomials:**

$$\begin{aligned} B_0(t) &= (1-t)^3 \\ B_1(t) &= 3t(1-t)^2 \\ B_2(t) &= 3t^2(1-t) \\ B_3(t) &= t^3. \end{aligned}$$

Cubic Bézier Blending Functions



Bézier Curve Properties

- **Interpolates endpoints** $\mathbf{P}_0$ and $\mathbf{P}_3$ since

$$\begin{aligned} B_0(0) &= 1, \quad B_1(0) = B_2(0) = B_3(0) = 0 \\ B_3(1) &= 1, \quad B_0(1) = B_1(1) = B_2(1) = 0 \end{aligned}$$

- The curve is within the **convex-hull** of its control points, and is **affine transform invariant** since

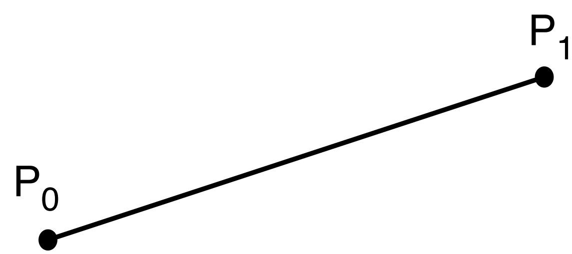
$$\begin{aligned} B_i(t) &\geq 0 \quad \text{(non-negative)} \\ \sum B_i(t) &= 1 \quad \text{(partition of unity)} \end{aligned}$$

- **Derivative** curve is quadratic Bézier curve with control points $\{\mathbf{P}_1 - \mathbf{P}_0,\ \mathbf{P}_2 - \mathbf{P}_1,\ \mathbf{P}_3 - \mathbf{P}_2\}$.
- **Variation diminishing property**: no straight line intersects the curve more that it intersects the control polygon (*i.e.*, the curve has no more "wiggles" than the control polygon).

1st degree Bézier curve

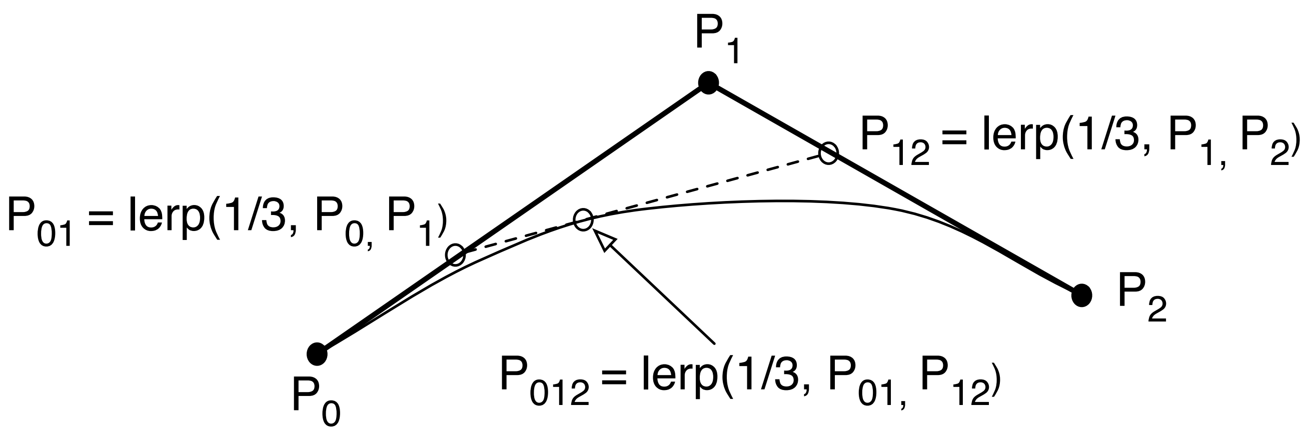
a line segment

$$\begin{aligned}\mathbf{C}(u) &= (1-u)\mathbf{P_0} + u\mathbf{P_1} \\ &= \mathrm{lerp}(u, \mathbf{P_0}, \mathbf{P_1})\end{aligned}$$



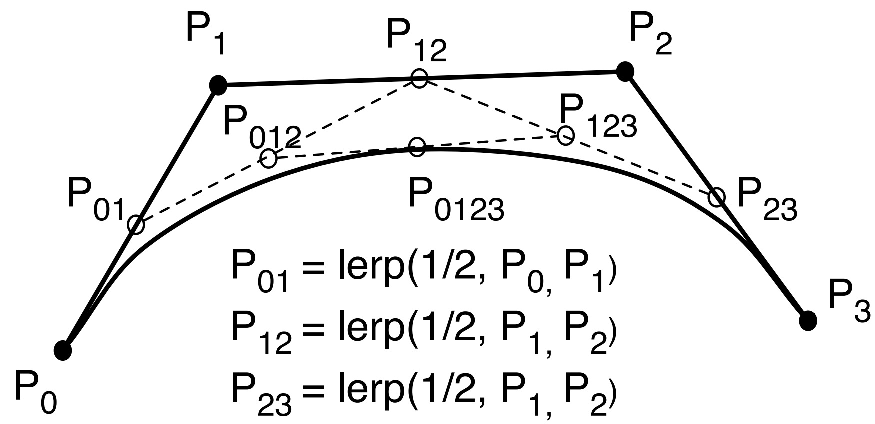
Quadratic (2nd degree) Bézier curve

$$
\begin{aligned}
\mathbf{C}(u) &= (1-u)(\underbrace{(1-u)\mathbf{P_0} + u\mathbf{P_1}}_{\text{linear}}) + u(\underbrace{(1-u)\mathbf{P_1} + u\mathbf{P_2}}_{\text{linear}}) \\
&= (1-u)^2\mathbf{P_0} + 2u(1-u)\mathbf{P_1} + u^2\mathbf{P_2} \\
&= \mathrm{lerp}(u,\ \mathrm{lerp}(u,\ \mathbf{P_0},\mathbf{P_1}),\ \mathrm{lerp}(u,\ \mathbf{P_1},\mathbf{P_2}))
\end{aligned}
$$



Cubic (3rd degree) Bézier curve

$$
\begin{aligned}
\mathbf{C}(u) &= (1-u)(\underbrace{(1-u)^2\mathbf{P_0} + 2u(1-u)\mathbf{P_1} + u^2\mathbf{P_2}}_{\text{quadratic}}) + \\
&\quad u(\underbrace{(1-u)^2\mathbf{P_1} + 2u(1-u)\mathbf{P_2} + u^2\mathbf{P_3}}_{\text{quadratic}}) \\
&= (1-u)^3\mathbf{P_0} + 3u(1-u)^2\mathbf{P_1} + 3u^2(1-u)\mathbf{P_2} + u^3\mathbf{P_3}
\end{aligned}
$$



$P_{01} = \text{lerp}(1/2, P_0, P_1)$

$P_{12} = \text{lerp}(1/2, P_1, P_2)$

$P_{23} = \text{lerp}(1/2, P_1, P_2)$

$P_{012} = \text{lerp}(1/2, P_{01}, P_{12})$

$P_{123} = \text{lerp}(1/2, P_{12}, P_{23})$

$P_{0123} = \text{lerp}(1/2, P_{012}, P_{123})$

nth degree Bézier curve

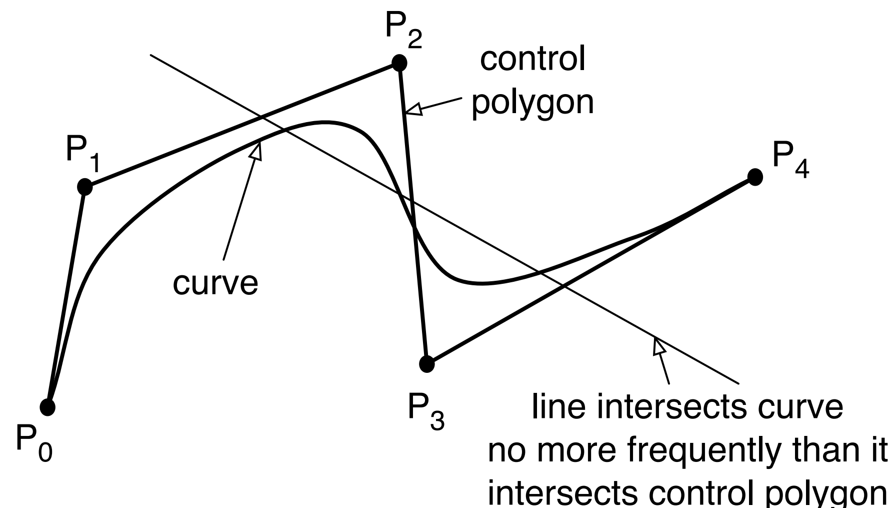
$$\mathbf{C}(u) = \sum_{i=0}^{n} B_{i,n}(u)\mathbf{P}_i, \qquad 0 \leq u \leq 1.$$

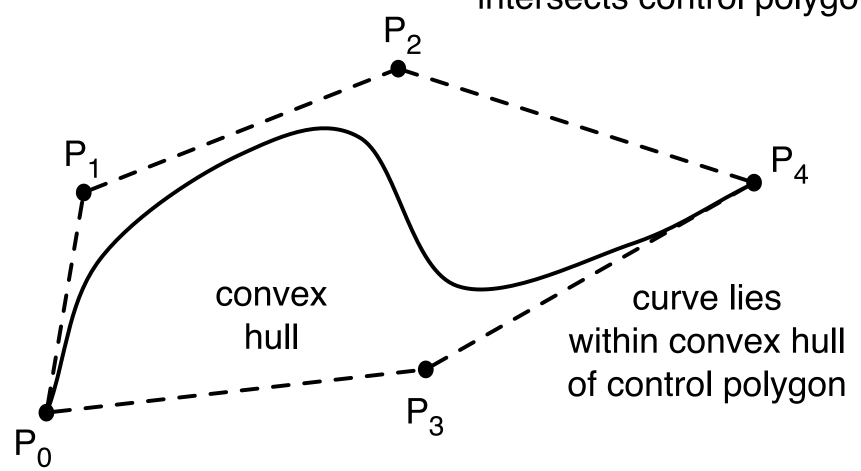
- The $n+1$ blending functions $\{B_{i,n}\}_{i=0}^{n}$ are the **Bernstein polynomials**

$$B_{i,n}(u) = \frac{n!}{i!(n-i)!}u^i(1-u)^{n-i}.$$

- The geometric coefficients $\{\mathbf{P}_i\}$ are called **control points** and, when connected by line segments, define the **control polygon**.

Convex-Hull and Variation Diminishing Property





deCasteljau Algorithm

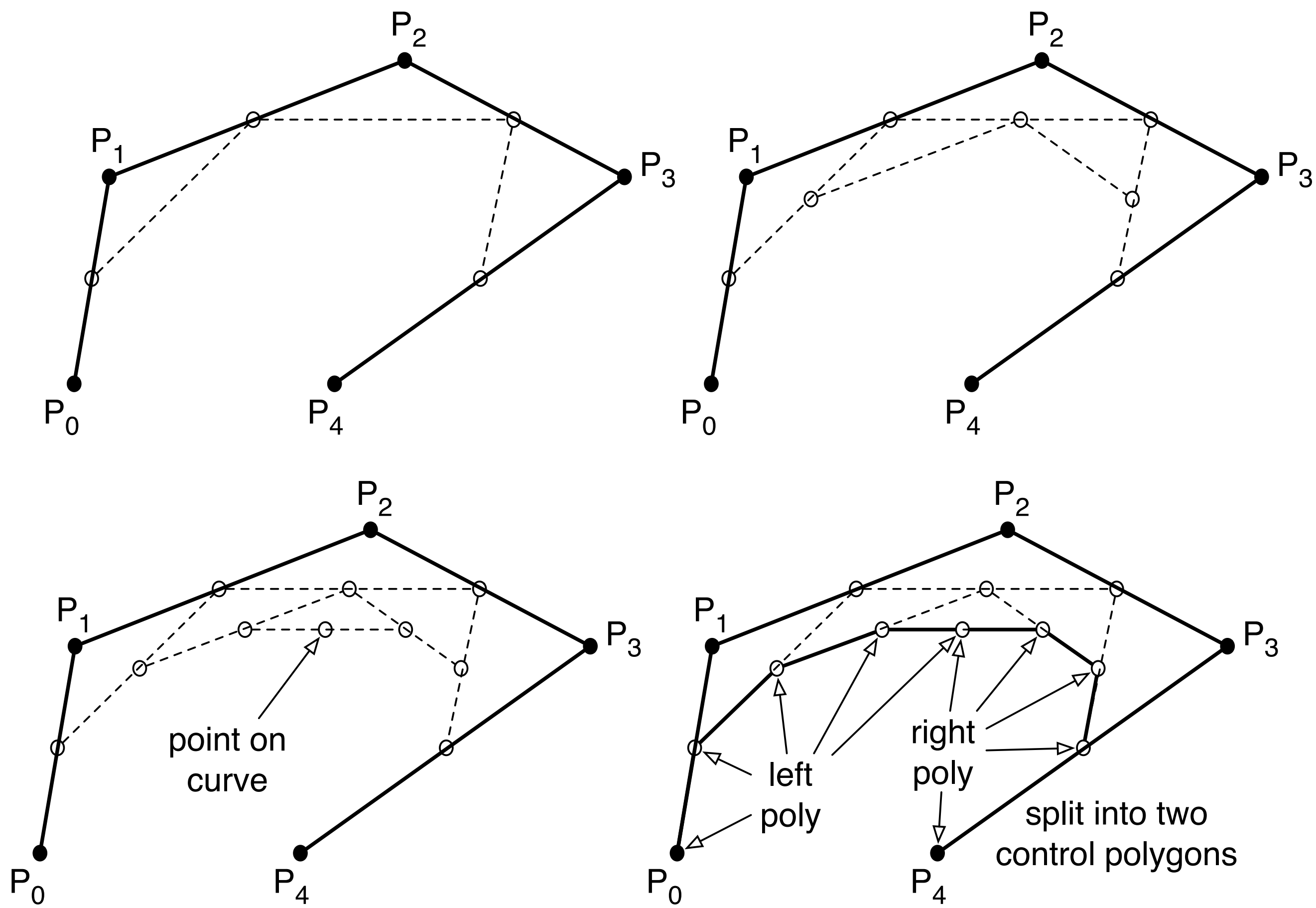
We can always represent an nth degree Bézier curve as a linear combination of two $n-1$ degree curves:

$$\mathbf{C}_n(\mathbf{P}_0,\ldots,\mathbf{P}_n) = (1-u)\mathbf{C}_{n-1}(\mathbf{P}_0,\ldots,\mathbf{P}_{n-1}) + u\mathbf{C}_{n-1}(\mathbf{P}_1,\ldots,\mathbf{P}_n)$$

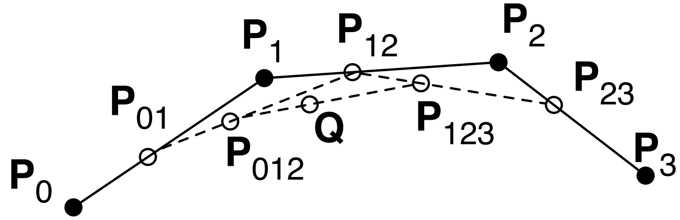
So if we fix $u = u_0$ and denote $\mathbf{P}_i$ by $\mathbf{P}_{0,i}$, we can describe the *deCasteljau Algorithm* for computing the point $\mathbf{C}(u_0) = \mathbf{P}_{n,0}(u_0)$ on an nth degree Bézier curve as

$$\begin{aligned} \mathbf{P}_{k,i}(u_0) &= (1-u_0)\mathbf{P}_{k-1,i}(u_0) + u_0\mathbf{P}_{k-1,i+1}(u_0), \\ & \quad \text{for} \begin{cases} k = 1,\ldots,n \\ i = 0,\ldots,n-k \end{cases} \end{aligned}$$

deCasteljau Algorithm Example $(u = 0.5)$



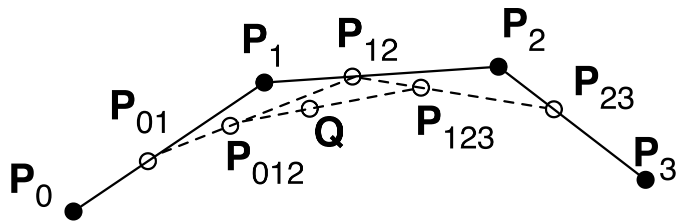
Point Q on Cubic Bézier curve via deCasteljau




```
Point lerp(float t, Point A, Point B) {
  return A*(1-t) + B*t; // + and * overloaded for Points
}

Point deCasteljau(float t, Point P[4]) {
  Point P01 = lerp(t, P[0], P[1]);
  Point P12 = lerp(t, P[1], P[2]);
  Point P23 = lerp(t, P[2], P[3]);
  Point P012 = lerp(t, P01, P12);
  Point P123 = lerp(t, P12, P13);
  return lerp(t, P012, P123);
}
```

Splitting a Cubic Bézier control polygon into two




```
void split(Point P[4], Point L[4], Point R[4]) {
  Point P01 = lerp(0.5, P[0], P[1]);
  Point P12 = lerp(0.5, P[1], P[2]);
  Point P23 = lerp(0.5, P[2], P[3]);
  Point P012 = lerp(0.5, P01, P12);
  Point P123 = lerp(0.5, P12, P13);
  Point Q = lerp(0.5, P012, P123);
  L[0] = P[0]; L[1] = P01;  L[2] = P012; L[3] = Q;
  R[0] = Q;    R[1] = P123; R[2] = P23;  R[3] = P[3];
}
```

Rational Parametric Curves

- **Bad News:** We can *not* represent circles or ellipses with parametric curves defined by polynomials.
- **Good News:** We can represent circles and ellipses using **rational polynomials.** For example, the unit circle in the first quadrant can be represented by

$$x(u) = \frac{1-u^2}{1+u^2}, \;\; y(u) = \frac{2u}{1+u^2}, \;\; u \in [0,1].$$

- We can use **homogenous coordinates** to represent rational polynomials:

$$x(u) = 1 - u^2, \;\; y(u) = 2u, \;\; w(u) = 1 + u^2, \;\; u \in [0,1].$$

$$(x(u),\; y(u),\; w(u)) \mapsto \left(\frac{x(u)}{w(u)},\; \frac{y(u)}{w(u)}\right)$$

Rational Bézier Curves

An nth degree Rational Bézier Curve

$$
\begin{aligned}
\mathbf{C}^w(u) &= (X(u),\ Y(u),\ Z(u),\ W(u)) \\
\mathbf{C}(u) &= \left(\frac{X(u)}{W(u)},\ \frac{Y(u)}{W(u)},\ \frac{Z(u)}{W(u)}\right)
\end{aligned}
$$

is defined by $n+1$ homogeneous control points

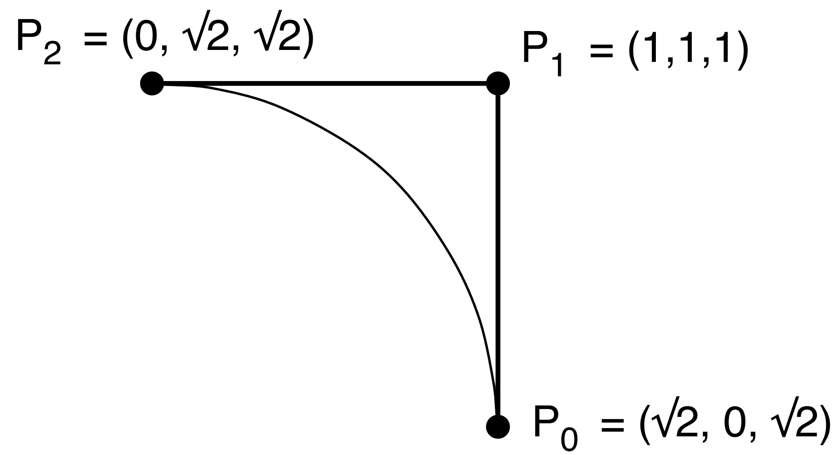
$$\{\mathbf{P}_i^w = (x_i w_i, y_i w_i, z_i w_i, w_i)\}_{i=0}^n$$

which are blended in 4-D using the Bernstein Polynomials:

$$
\begin{array}{ll}
X(u) = \sum_{i=0}^n B_{i,n}(u) w_i x_i & Y(u) = \sum_{i=0}^n B_{i,n}(u) w_i y_i \\
Z(u) = \sum_{i=0}^n B_{i,n}(u) w_i z_i & W(u) = \sum_{i=0}^n B_{i,n}(u) w_i.
\end{array}
$$

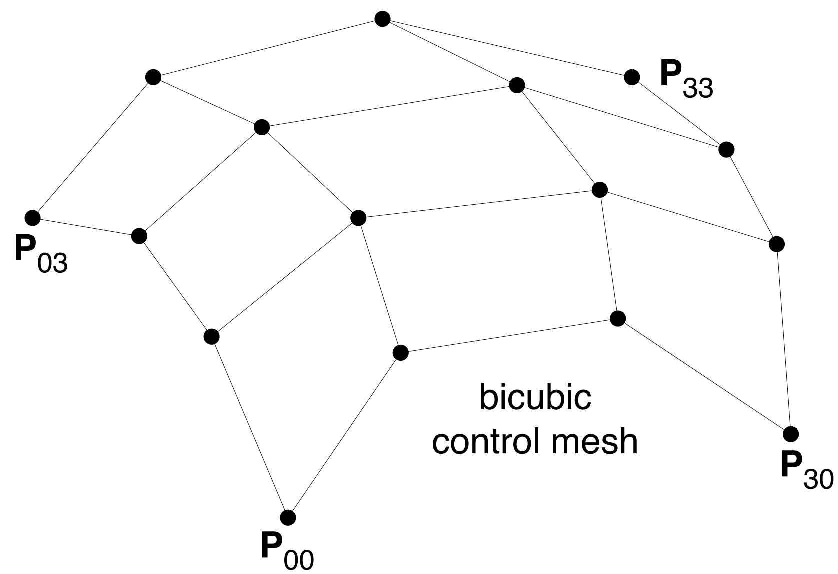
The values w_i are called weights ($w_i > 0$ for points, and $w_i = 0$ for direction vectors).

Circular Arc via a Quadratic Rational Bézier



$$
\begin{aligned}
X(t) &= \sqrt{2}(1-t)^2 + 2(1-t)t \\
Y(t) &= 2(1-t)t + \sqrt{2}t^2 \\
W(t) &= \sqrt{2}(1-t)^2 + 2(1-t)t + \sqrt{2}t^2
\end{aligned}
$$

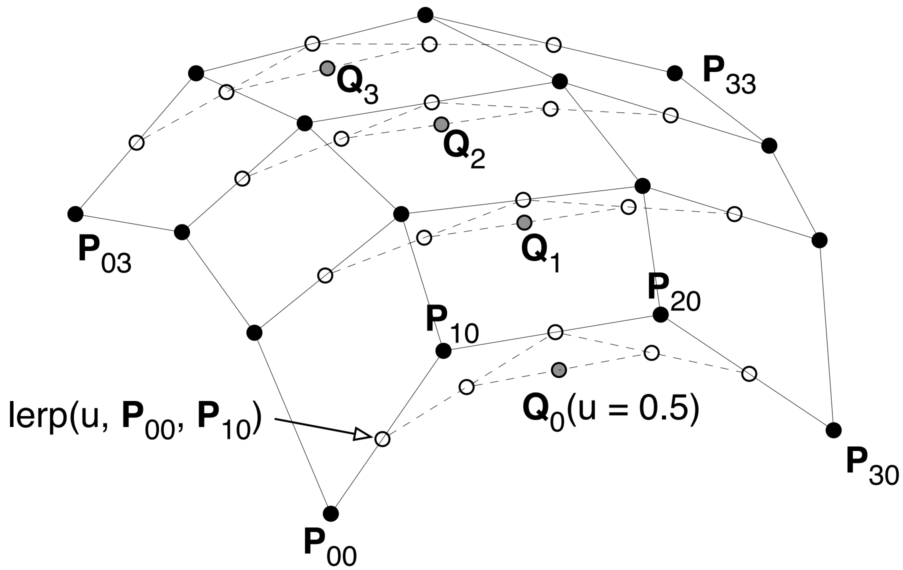
Bézier Surfaces



- $(n+1) \times (m+1)$ points in **control mesh** $\{\mathbf{P}_{ij}\}$.
- bicubic surface : $n = m = 3$
- tensor product surface

$$\mathbf{S}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_{i,n}(u) B_{j,m}(v) \mathbf{P}_{ij}$$

Fixing $u = u_0$ we get a Bézier curve on the surface



$$\mathbf{S}(u_0, v) = \sum_{j=0}^{m} B_{j,m}(v) \underbrace{\left(\sum_{i=0}^{n} B_{i,n}(u) \mathbf{P}_{ij} \right)}_{\mathbf{Q}_j(u_0)} = \sum_{j=0}^{m} B_{j,m}(v) \mathbf{Q}_j(u_0)$$

The points $\{\mathbf{Q}_j\}_{j=0}^{3}$ (computed via deCasteljau) define a cubic Bézier curve.

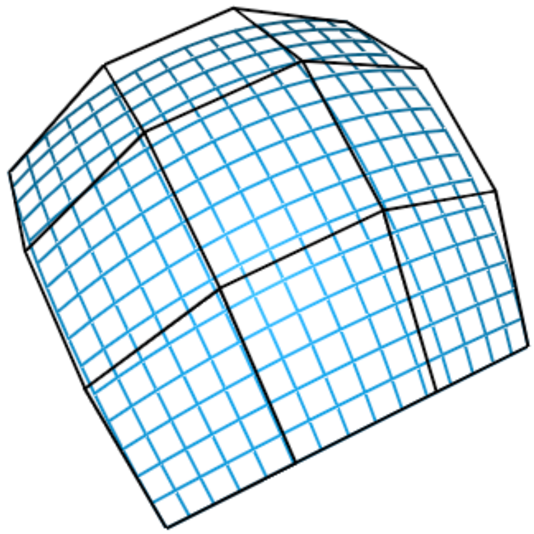
Computing Point on Cubic Bézier surface

```
//
// coefficient P_{ij} stored P[j][i]
//
Point eval(float u, float v, Point P[4][4]) {
  Point Q[4];
  for (int j = 0; j <= 3; j++)
    Q[j] = deCasteljau(u, P[j]);
  return deCasteljau(v, Q);
}
```

Bézier Surface Properties

- Surface **interpolates four corners** of control mesh.
- **Affine invariant:** merely transform control points when transforming surface.
- Surface within **convex hull** of control mesh.
- **Variation diminishing property:** line pierces surface no more frequently than it pierces the control mesh.
- **Continuity** of adjacent surfaces governed by continuity of adjacent control meshes.

Rendering a Bicubic Bézier Surface using OpenGL Evaluators



4×4 (non-rational) control mesh:

```
GLfloat controlMesh[4][4][3] = {
  {{0.0, 0.0, 0.0}, {1.0, 0.0, 1.0}, {2.0, 0.0, 1.0}, {3.0, 0.0, 0.0}},
  {{0.0, 1.0, 1.0}, {1.0, 1.0, 2.0}, {2.0, 1.0, 2.0}, {3.0, 1.0, 1.0}},
  {{0.0, 2.0, 1.0}, {1.0, 2.0, 2.0}, {2.0, 2.0, 2.0}, {3.0, 2.0, 1.0}},
  {{0.0, 3.0, 0.0}, {1.0, 3.0, 1.0}, {2.0, 3.0, 1.0}, {3.0, 3.0, 0.0}}
};
```

OpenGL 2D-Evaluator
based on bivariate Bernstein polynomials

```
GLfloat controlMesh[4][4][3] = {...};

glEnable(GL_MAP2_VERTEX_3);// enable 2D evaluator map
glMap2f(GL_MAP2_VERTEX_3,  // 3:non-rational, 4:homogeneous
        0.0, 1.0,          // 0 <= u <= 1
        3,                 // u stride: 3 floats per coord
        4,                 // u is 4th order (degree = 3)
        0.0, 1.0,          // 0 <= v <= 1
        4*3,               // v stride: 4*3 floats per row
        4,                 // 4th order surface (degree = 3)
        controlMesh);      // ptr to buffer holding points
```

Rendering the Surface using the Evaluator

- Define 2D grid of sample points

```
glMapGrid2f(15, 0.0, 1.0, //15 u grid samples, 0<=u<=1
            15, 0.0, 1.0);//15 v grid samples, 0<=v<=1
```

- Let OpenGL generate the surface normals:

```
glEnable(GL_AUTO_NORMAL);  // auto-generate normals
```

- Draw the mesh:

```
glEvalMesh2(GL_LINE        // or  GL_FILL
            0, 15,
            0, 15);
```

Evaluating Coordinates

`glEvalMesh2(GL_LINE, 0,15, 0,15)` is equivalent to

```
du = dv = 1.0/15;
for (j = 0, v = 0.0; j <= 15; j++, v += dv) {
  glBegin(GL_LINE_STRIP);
  for (i = 0, u = 0.0; i <= 15; i++, u += du)
    glEvalCoord2f(u, v); // invokes glNormal(), glVertex()
  glEnd();
}
for (i = 0, u = 0.0; i <= 15; i++, u += du) {
  glBegin(GL_LINE_STRIP);
  for (j = 0, v = 0.0; j <= 15; j++, v += dv)
    glEvalCoord2f(u, v);
  glEnd();
}
```

Rendering a Solid Mesh

glEvalMesh2(GL_FILL, 0,15, 0,15) is equivalent to

```
du = dv = 1.0/15;
for (j = 0, v = 0.0; j < 15; j++, v += dv) {
  glBegin(GL_QUAD_STRIP);
  for (i = 0, u = 0.0; i <= 15; i++, u += du) {
    glEvalCoord2f(u, v); // invokes glNormal(), glVertex()
    glEvalCoord2f(u, v+dv);
  }
  glEnd();
}
```