10. Spline and Surfaces

Hongxin Zhang
State Key Lab of CAD&CG, Zhejiang University

2013-10-16
Splines
Bézier curve

\[
C(t) = \sum_{i=0}^{n} P_i B_{i,n}(t), \quad t \in [0, 1]
\]

where, \( P_i \) (i=0,1,…,n) are control points.

\[
B_{i,n}(t) = C_n^i t^i (1-t)^{n-i}, \quad t \in [0, 1]
\]  Bernstein basis

\[
\begin{align*}
X(t) &= \sum_{i=0}^{n} x_i B_{i,t}(t) \\
Y(t) &= \sum_{i=0}^{n} y_i B_{i,t}(t)
\end{align*}
\]

\[
C(t) = \begin{pmatrix}
X(t) \\
Y(t)
\end{pmatrix}, \quad P_i = \begin{pmatrix}
x_i \\
y_i
\end{pmatrix}
\]
Bézier curve
General spline curves

parametric curve

$P(t) = \sum_{i} P_i B_i(t)$

basis functions

$t \in [t_0, t_1)$

control points
Bézier curve

Properties of Bernstein basis

\[ B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad t \in [0,1] \]

1. \( B_{i,n}(t) \geq 0, \quad i = 0, 1, \ldots, n, \quad t \in [0,1]. \)

2. \( \sum_{i=0}^{n} B_{i,n}(t) = 1, \quad t \in [0,1]. \)

3. \( B_{i,n}(t) = B_{n-i,n}(1-t), \quad i = 0, 1, \ldots, n, \quad t \in [0,1]. \)

4. \[
B_{i,n}(0) = \begin{cases} 
1, & i = 0, \\
0, & \text{else}; 
\end{cases} \quad B_{i,n}(1) = \begin{cases} 
1, & i = n, \\
0, & \text{else}. 
\end{cases}
\]
Bézier curve

Properties of Bernstein basis

5. \[ B_{i,n}(t) = (1-t)B_{i,n-1}(t) + tB_{i-1,n-1}(t), \quad i = 0,1,...,n. \]

6. \[ B'_{i,n}(t) = n[B_{i-1,n-1}(t) - B_{i,n-1}(t)], \quad i = 0,1,...,n. \]

7. \[
(1-t)B_{i,n}(t) = (1 - \frac{i}{n+1})B_{i,n+1}(t); \\
tB_{i,n}(t) = \frac{i+1}{n+1}B_{i+1,n+1}(t); \\
B_{i,n}(t) = (1 - \frac{i}{n+1})B_{i,n+1}(t) + \frac{i+1}{n+1}B_{i+1,n+1}(t).\]
properties of Bézier curves

\[ C(t) = \sum_{i=0}^{n} P_i B_{i,n} (t), \quad t \in [0,1] \]

1. **Endpoint Interpolation**: interpolating two end points

   \[ C(0) = P_0, \quad C(1) = P_n. \]

2. **tangent direction** of \( P_0 \): \( P_0P_1 \), tangent direction of \( P_n : P_{n-1}P_n \).

   \[ C'(t) = n \sum_{i=0}^{n-1} (P_{i+1} - P_i) B_{i,n-1} (t), \quad t \in [0,1]; \quad C'(0) = n(P_1 - P_0), \quad C'(1) = n(P_n - P_{n-1}). \]

3. **Symmetry**: Let two Bézier curves be generated by ordered Bezier (control) points labelled by \{p0,p1,...,pn\} and \{pn, pn-1,..., p0\} respectively, then the curves corresponding to the two different orderings of control points look the same; they differ only in the direction in which they are traversed.
properties of Bézier curves

\[ C(t) = \sum_{i=0}^{n} P_i B_{i,n}(t), \quad t \in [0,1] \]

4. Affine Invariance –
the following two procedures yield the same result:
(1) first, from starting control points \{p0, p1, ..., pn\} compute the curve and then apply an affine map to it;
(2) first apply an affine map to the control points \{p0, p1, ..., pn\} to obtain new control points \{F(p0), ..., F(pn)\} and then find the curve with these new control points.
5. **Convex hull property**: Bézier curve $\mathbf{C}(t)$ lies in the convex hull of the control points $\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_n$.

6. **Variation diminishing property**. Informally this means that the Bezier curve will not "wiggle" any more than the control polygon does.
Bézier curves

1. linear: \( C(t) = (1-t)P_0 + tP_1, \quad t \in [0,1], \)

\[
C(t) = [t, 1] \begin{bmatrix}
-1 & 1 \\
1 & 0
\end{bmatrix} \begin{bmatrix}
P_0 \\
P_1
\end{bmatrix}
\]

2. quadratic

\[ C(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2 \]

\[
C(t) = [t^2 \quad t \quad 1] \begin{bmatrix}
1 & -2 & 1 \\
-2 & 2 & 0 \\
1 & 0 & 0
\end{bmatrix} \begin{bmatrix}
P_0 \\
P_1 \\
P_2
\end{bmatrix}
\]
Bézier curve

3. cubic:

\[ C(t) = (1 - t)^3 P_0 + 3t(1 - t)^2 P_1 + 3t^2 (1 - t)P_2 + t^3 P_3 \]

\[
C(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}
\]

Degree 3
Bezier Curve

\[ T_0 = 3(P_1 - P_0) \]

\[ T_1 = 3(P_3 - P_2) \]
Beziers Curve in OpenGL

- `glMap1*(GL_MAP1_VERTEX_3, uMin, uMax, stride, nPts, *ctrlPts);`
- `glEnable/glDisable(GL_MAP1_VERTEX_3);`
- `glBegin(GL_LINE_STRIP);
  for (...) {
    glEvalCoord1*(uValue);
  }
  glEnd();`
Beziers Curve in OpenGL

GLfloat ctrlpoints[4][3] = {
    { -4.0, -4.0, 0.0}, { -2.0, 4.0, 0.0},
    {2.0, -4.0, 0.0}, {4.0, 4.0, 0.0}};

void display(void)
{
    int i;

    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_LINE_STRIP);
    for (i = 0; i <= 30; i++)
        glEvalCoord1f((GLfloat) i/30.0);
    glEnd();
    /* The following code displays the control points as dots. */
    glPointSize(5.0);
    glColor3f(1.0, 1.0, 0.0);
    glBegin(GL_POINTS);
    for (i = 0; i < 4; i++)
        glVertex3fv(&ctrlpoints[i][0]);
    glEnd();
    glFlush();
}

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_FLAT);
    glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, &ctrlpoints[0][0]);
    glEnable(GL_MAP1_VERTEX_3);
    / * The following code displays the control points as dots. */
    glPointSize(5.0);
    glColor3f(1.0, 1.0, 0.0);
    glBegin(GL_POINTS);
    for (i = 0; i < 4; i++)
        glVertex3fv(&ctrlpoints[i][0]);
    glEnd();
    glFlush();
}
Bézier curve

de Casteljau algorithm

given the control points $P_0, P_1, \ldots, P_n$, and $t$ of Bézier curve, let:

$$P_i^r(t) = (1-t)P_i^{r-1}(t) + tP_{i+1}^{r-1}(t), \quad r = 1, \ldots, n; \quad i = 0, \ldots, n-r$$

then $P_0^n(t) = C(t)$. 
Consider Three Points
Insert Point Using Linear Interpolation

\[ P_0^1 = (1 - t)P_0 + tP_1 \]
Insert Points on Both Edges

\[ P_0^1 = (1 - t)P_0 + tP_1 \]

\[ P_1^1 = (1 - t)P_1 + tP_2 \]
Repeat Recursively

$$P_0^1 = (1-t)P_0 + tP_1$$
$$P_1^1 = (1-t)P_1 + tP_2$$
$$P_0^2 = (1-t)P_0^1 + tP_1^1$$
Algorithm Defines Curve

\[ \begin{align*}
    P_0^1 &= (1 - t)P_0 + tP_1 \\
    P_1^1 &= (1 - t)P_1 + tP_2 \\
    P_0^2 &= (1 - t)P_0^1 + tP_1^1
\end{align*} \]

Resulting point \[ P(t) = P_0^2 \]
Consider Four Points

\[ P_0 \quad P_1 \quad P_2 \quad P_3 \]
Linear Interpolation

\[ P_0, P_1, P_2, P_3 \]

\[ P_0^t, P_1^1 - t \]

CS148 Lecture 8

Pat Hanrahan, Fall 2010
On All Edge Segments
Repeat Recursively
Algorithm Defines Curve

P(t) = P_0^3
Properties

Property 1: Interpolate end points
\[ P(0) = P_0 \]
\[ P(1) = P_3 \]

Property 2: Tangents
\[ P'(0) = 3(P_1 - P_0) \]
\[ P'(1) = 3(P_3 - P_2) \]

Property 3: Convex hull property
\[ P(t) \text{ inside } \text{chull}(P_0, P_1, P_2, P_2) \]
Pyramid Algorithm

\[
\begin{array}{c}
P_0 \\
1-t \\
P_0^1 \\
P_0^3 \\
P_0^1 \\
P_0^2 \\
P_0 \\
\end{array}
\begin{array}{c}
P_1 \\
t \\
P_1^1 \\
P_1^2 \\
P_1^1 \\
P_1^2 \\
P_1 \\
\end{array}
\begin{array}{c}
P_2 \\
P_2^1 \\
P_2^2 \\
P_2 \\
\end{array}
\begin{array}{c}
P_3 \\
\end{array}
\end{array}
\]
Pyramid Algorithm

\[ P(t) = \sum_{i=0}^{3} P_i B_i(t) \]
Pyramid Algorithm

Path

\( (1 - t)^3 P_0 \)
Pyramid Algorithm

Path
\[ t(1 - t)^2 P_1 \]
Pyramid Algorithm

\[ P_0 \quad P_1 \quad P_2 \quad P_3 \]

\[ 1-t \quad 1-t \quad 1-t \quad t \]

Path
\[ t(1-t)^2P_1 \]
Pyramid Algorithm

Three paths total

\[3t(1 - t)^2 P_0^1\]
Leads to a Cubic Polynomial Curve

\[ P(t) = \sum_{i}^{n} P_i B_i^n(t) \]

\[ B_0^3(t) = (1 - t)^3 \]
\[ B_1^3(t) = 3t(1 - t)^2 \]
\[ B_2^3(t) = 3t^2(1 - t) \]
\[ B_3^3(t) = t^3 \]

Bernstein polynomials

\[ B_i^n(t) = \binom{n}{i} t^i (1 - t)^{n-i} \]
Rational Bézier Curve

\[ R(t) = \frac{\sum_{i=0}^{n} B_{i,n}(t) \omega_i P_i}{\sum_{i=0}^{n} B_{i,n}(t) \omega_i} = \sum_{i=0}^{n} R_{i,n}(t) P_i \]

where \( B_{i,n}(t) \) is Bernstein basis, \( \omega_i \) is the weight at \( p_i \).

It’s a generalization of Bézier curve, which can express more curves, such as circle.
Properties of rational Bézier curve:

1. endpoints: \( R(0) = P_0; \ R(1) = P_n \)

2. tangent of endpoints:
   \[
   R'(0) = n \frac{\omega_1}{\omega_0} (P_1 - P_0); \quad R'(1) = n \frac{\omega_{n-1}}{\omega_0} (P_n - P_{n-1})
   \]

3. Convex Hull Property
   ......

5.

6. Influence of the weights

Figure 2.16: Rational Bézier curve.
Bézier surface

Bézier surface:

\[ S(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} P_{ij} B_{i,n}(u) B_{j,m}(v), \quad 0 \leq u, v \leq 1 \]

where \( B_{i,n}(u) \) and \( B_{j,m}(v) \) Bernstein basis with degree \( n \) and \( m \) degree, respectively, \((n + 1) \times (m + 1) P_{ij}(i=0,1,...,n; j=0,1,...,m)\) construct the control meshes.
Bezier Surface in OpenGL

- `glMap2*(GL_MAP2_VERTEX_3, uMin, uMax, uStride, nuPts, vMin, vMax, vStride, nvPts,*ctrlPts);`
- `glEnable/glDisable(GL_MAP2_VERTEX_3);`
- `glBegin(GL_LINE_STRIP); / GL_QUAD_STRIP
  for (...) {
    glVertex2*(uValue, vValue);
  }
  glEnd();`
Bezier Surface in OpenGL

- `glBegin(GL_LINE_STRIP);` / `GL_QUAD_STRIP`
  
  ```
  for (...) {
    `glEvalCoord2*(uValue, vValue);
  }
  `glEnd();
  ```

```c
GLfloat controlpoints[4][4][3] = {
    {{-1.5, -1.5, 4.0}, {-0.5, -1.5, 2.0},
     {0.5, -1.5, -1.0}, {1.5, -1.5, 2.0}},
    {{-1.5, -0.5, 1.0}, {-0.5, -0.5, 3.0},
     {0.5, -0.5, 0.0}, {1.5, -0.5, -1.0}},
    {{-1.5, 0.5, 4.0}, {-0.5, 0.5, 0.0},
     {0.5, 0.5, 3.0}, {1.5, 0.5, 4.0}},
    {{-1.5, 1.5, -2.0}, {-0.5, 1.5, -2.0},
     {0.5, 1.5, 0.0}, {1.5, 1.5, -1.0}}
};
```
normal vector of Bézier surface

partial derivation of Bézier surface $S(u,v)$:

$$\frac{\partial}{\partial u} S(u,v) = \frac{\partial}{\partial u} \sum_{i=0}^{n} \sum_{j=0}^{m} P_{ij} B_{i,n}(u) B_{j,m}(v) = \sum_{i=0}^{n-1} \sum_{j=0}^{m} (P_{i+1,j} - P_{ij}) B_{i,n-1}(u) B_{j,m}(v)$$

$$\frac{\partial}{\partial v} S(u,v) = \frac{\partial}{\partial v} \sum_{i=0}^{n} \sum_{j=0}^{m} P_{ij} B_{i,n}(u) B_{j,m}(v) = \sum_{i=0}^{n-1} \sum_{j=0}^{m} (P_{i,j+1} - P_{ij}) B_{i,n}(u) B_{j,m-1}(v)$$

normal $\mathbf{N}(u,v)$:

$$\mathbf{N}(u,v) = \frac{\partial S(u,v)}{\partial u} \times \frac{\partial S(u,v)}{\partial v}$$
Disadvantages of Bézier curve:

1. control points determine the degree of the curve; many control points means high degree.

2. It’s **global**. A control point influences the whole curve.
Corner Cutting Algorithm

Chaiken (1974)

CS148 Lecture 8

Pat Hanrahan, Fall 2010
Procedural Curve

Repeatedly cutting corners generates a limit curve
1. Interpolates midpoints
2. Tangent preserved at midpoints
B-spline curve

• disadvantages of Bézier curve:
  1. control points determine the degree of the curve. many control points means high degree.
  2. It’s global. A control point influences the whole curve.

de Boor et al. replaced Bernstein basis with B-spline basis to generate B-spline curve.
B-spline curve:

\[ C(u) = \sum_{i=0}^{n} P_i N_{i,p}(u) \quad a \leq u \leq b \]

Where \( P_0, P_1, \ldots, P_n \) are control points, \( u=[u_0=a, u_1, \ldots, u_i, \ldots, u_{n+k+1}=b] \).

\[ N_{i,0}(u) = \begin{cases} 1 & u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases} \]

\[ N_{i,p}(u) = \frac{u-u_i}{u_{i+p}-u_i} N_{i,p-1}(u) + \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}} N_{i+1,p-1}(u), \]

\[ \frac{0}{0} = 0 \]
B-spline basis

\[ U = \left\{ u_i \right\}_{i=-\infty}^{\infty} \]

\[ N_{i,0}(u) = \begin{cases} 1 & u_i \leq u < u_{i+1} \\ 0 & \text{else} \end{cases} \]

\[ N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u), \]

\[ \frac{0}{0} = 0 \]
B-spline basis v.s. Bernstein ~

\[ B_{i,n}(t) = (1 - t)B_{i,n-1}(t) + tB_{i-1,n-1}(t), \quad i = 0, 1, \ldots, n. \]

**Fig. 2**

\[
N_{i,0}(u) = \begin{cases} 
1 & u_i \leq u < u_{i+1} \\
0 & \text{else}
\end{cases}
\]

\[
N_{i,p}(u) = \frac{u-u_i}{u_{i+p}-u_i} N_{i,p-1}(u) + \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}} N_{i+1,p-1}(u),
\]

\[
0 = 0
\]

Computer Graphics @ ZJU

Hongxin Zhang, 2013
**properties of B-spline basis**

1. localization: \( N_{i,p}(u) > 0 \) only when \( u \in [u_i, u_{i+p+1}] \).

\[
N_{i,p}(u) = \begin{cases} 
> 0, & u_i \leq u < u_{i+p+1} \\
= 0, & u < u_i \text{ or } u > u_{i+p+1}
\end{cases}
\]

2. normalization:

\[
\sum_{j=-\infty}^{\infty} N_{j,p}(u) = \sum_{j=i-p}^{i} N_{j,p}(u) = 1, \quad u \in [u_i, u_{i+1})
\]

3. piecewise polynomial: \( N_{i,p}(u) \) is a polynomial with degree \( < p \), in every \([u_j, u_{j+1})\).

4. differential:

\[
N_{i,p}'(u) = \frac{p}{u_{i+p} - u_i} N_{i,p-1}(u) - \frac{p}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u)
\]
Properties of B-spline curve:

1. Convex Hull Property
2. variation diminishing property.
3. Affine Invariance
4. local
5. piecewise polynomial
B-spline---de Boor algorithm

to calculate the point of B-spline curve \( C(u) \) at \( u \):

1. find the interval where \( u \) lies in: \( u \in [u_j, u_{j+1}) \);

2. curve in \( u \in [u_j, u_{j+1}) \) is only determined by \( P_{j-p}, P_{j-p+1}, \ldots, P_j \);

3. calculate

\[
P_i^r(u) = \begin{cases} 
  P_i & r = 0, i = j - p; \ j - p + 1, L \ j; \\
  \frac{u - u_i}{u_{i+k-r} - u_i} P_i^{r-1}(u) + \frac{u_{i+k-r} - u}{u_{i+k-r} - u_i} P_{i-1}^{r-1}(u), & r = 1, 2, L \ k - 1; \ i = j - p + r, j - p + r + 1, L \ j.
\end{cases}
\]

4. \( P_j^{k-1}(u) = C(u) \)
Catmull-Clark and Doo-Sabin subdivision

Start from

\[ P^i = (L, p^-_1, p_0^i, p_1^i, p_2^i, L) \]

Catmull-Clark rules

\[ p_{2j}^{i+1} = \frac{1}{8} p_{j-1}^i + \frac{6}{8} p_j^i + \frac{1}{8} p_{j+1}^i \]

\[ p_{2j+1}^{i+1} = \frac{4}{8} p_j^i + \frac{4}{8} p_{j+1}^i \]

Doo-Sabin rules:

\[ p_{2j}^{i+1} = \frac{3}{4} p_j^i + \frac{1}{4} p_{j+1}^i \]

\[ p_{2j+1}^{i+1} = \frac{1}{4} p_j^i + \frac{3}{4} p_{j+1}^i \]

Figure 3: Subdividing an initial set of control points (upper, red) results in additional control points (lower, black), that more closely approximate a curve.
B-spline surface

\((n+1) \times (m+1)\) control points: \(P_{ij}\) (Degrees of \(u, v\): \(p, q\));

nodes: \(U = [u_0, u_1, \ldots, u_{n+p+1}], V = [v_0, v_1, \ldots, v_{m+q+1}]\),

Then a tensor B-spline surface with degree \(p \times q\):

\[
S(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) P_{i,j}
\]
NURBS (Non-uniform Rational B-spline)

NURBS curves:

$$C(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u) \omega_i P_i}{\sum_{i=0}^{n} N_{i,p}(u) \omega_i}, \quad a \leq u \leq b$$

$$U = \{a, \ldots, a, u_{p+1}, \ldots, u_{m-p-1}, b, \ldots, b\}$$
NURBS surface

\[ S(u, v) = \frac{\sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) \omega_{i,j} P_{i,j}}{\sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) \omega_{i,j}} \quad 0 \leq u, v \leq 1 \]

\( \omega_{ij} \): weights

\[ U = \{ 0, \ldots, 0, u_{p+1}, \ldots, u_{r-p-1}, 1, \ldots, 1 \} \]

\[ \begin{array}{c}
p+1 \\
p+1
\end{array} \]

\[ V = \{ 0, \ldots, 0, v_{q+1}, \ldots, v_{s-q-1}, 1, \ldots, 1 \} \]

\[ \begin{array}{c}
q+1 \\
q+1
\end{array} \]
NURBS in OpenGL

- curveName = gluNewNurbsRenderer();
  gluBeginCurve (curveName);

  gluNurbsCurbe(curveName, nknots, *knotVector,
  stride,*ctrlPts, order, GL_MAP1_VERTEX_3);

  gluEndCurve(curveName);
Surface trimming

Trimming Loops

Computer Graphics @ ZJU

Hongxin Zhang, 2013
Sweeping
subdivision curves:

- starting from a set of points, generate new points in every step under some rules, when such step goes on infinitely, the points will be convergent to a smooth curve.
subdivision surface