Graphics & Visualization

Chapter 7

Parametric Curves and Surfaces

Introduction

- We have already presented algorithms for the rasterization of basic geometric primitives, lines and circles.
- Realistic graphics scenes need more flexible, free-form curves & surfaces.
- CAGD (Computer-Aided Geometric Design): area of computer graphics that deals with free-form shapes
- 1960's:
  - The need for mathematical representations of free-form shapes became apparent in the automotive and aeronautic industries
  - Paul de Casteljau & Pierre Bézier developed independently the theory of polynomial curves & surfaces
  - This theory constitutes the basic tool for describing and rendering free-form shapes.

Introduction (2)

- All of the curve and surface descriptions are in parametric form:
- Curves in parametric form:
  - Given as 2 (for plane curves) or 3 (for space curves) independent *coordinate functions* in terms of a parameter *t*.

$$\mathbf{X}(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} \quad \text{or} \quad \mathbf{X}(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}$$

  - The description of plane and space curves is essentially the same, $z(t) = 0$ for a plane curve
- Surfaces in parametric form:
  - are given as 3 independent coordinate functions in terms of 2 parameters *u* and *v*:

$$\mathbf{X}(u,v) = \begin{bmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{bmatrix}$$

Introduction (3)

- Line segment in parametric form:
  - Between two points $\mathbf{p}_0$ and $\mathbf{p}_1$
  - $\mathbf{P}(t) = (1-t)\,\mathbf{p}_0 + t\,\mathbf{p}_1\ ,\ \ t \in [0,1]$
  - Expresses the *linear interpolation* between these points ($\mathbf{p}_0$ and $\mathbf{p}_1$)

Bézier Curves

I. Quadratic Bézier Curves
II. *n*th-Degree Bézier Curves
III. The de Casteljau Algorithm
IV. Bernstein Polynomials
V. Properties of Bézier Curves
VI. Bézier Curve Subdivision
VII. Smoothly Joining Bézier Curves

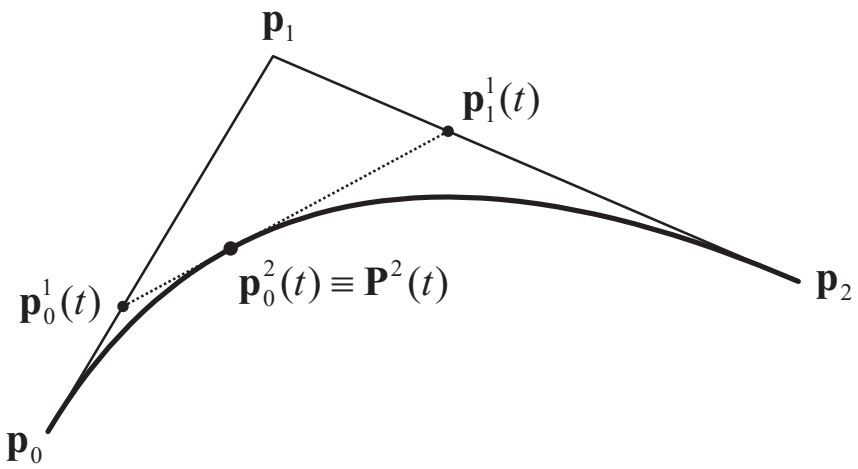
I. Quadratic Bézier Curves

- Consider 3 points $\mathbf{p}_0$, $\mathbf{p}_1$ and $\mathbf{p}_2$
- First step: interpolate them in pairs ($\mathbf{p}_0$, $\mathbf{p}_1$) and ($\mathbf{p}_1$, $\mathbf{p}_2$):

$$\mathbf{p}_0^1(t) = (1-t)\mathbf{p}_0 + t\mathbf{p}_1 , \quad t \in [0,1]$$
$$\mathbf{p}_1^1(t) = (1-t)\mathbf{p}_1 + t\mathbf{p}_2 ,$$

- For each value $t \in [0,1]$, $\mathbf{p}_0^1(t)$ and $\mathbf{p}_1^1(t)$ represent points on the line segments $\mathbf{p}_0\mathbf{p}_1$ and $\mathbf{p}_1\mathbf{p}_2$ respectively.
- Second step: interpolate points $\mathbf{p}_0^1(t)$ and $\mathbf{p}_1^1(t)$ for the same t:

$$\begin{aligned}\mathbf{p}_0^2(t) &= (1-t)\mathbf{p}_0^1(t) + t\mathbf{p}_1^1(t) \\ &= (1-t)^2\mathbf{p}_0 + 2t(1-t)\mathbf{p}_1 + t^2\mathbf{p}_2\end{aligned}$$



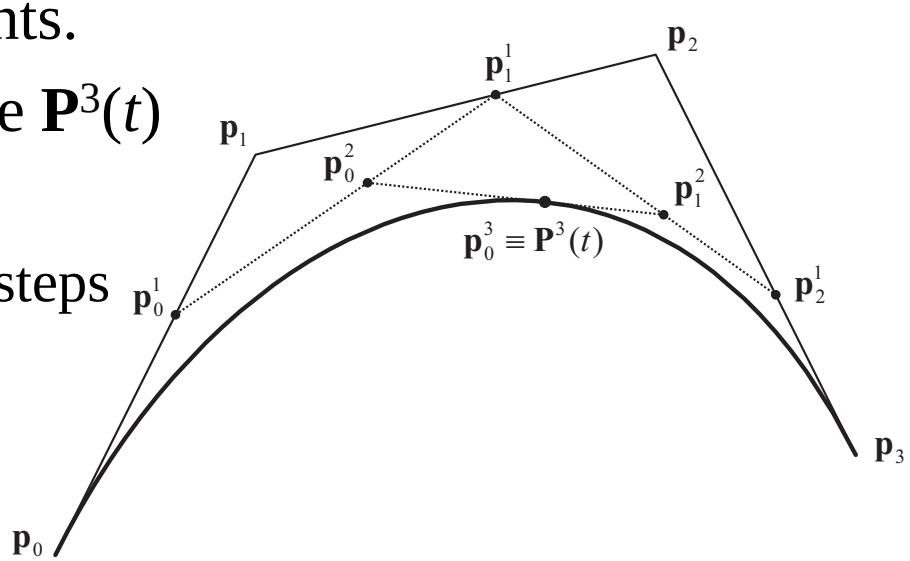
I. Quadratic Bézier Curves (2)

- As t increases from 0 to 1, the three points $\mathbf{p}_0^1(t)$, $\mathbf{p}_1^1(t)$, $\mathbf{p}_0^2(t)$ move concurrently on the respective line segments
- Last equation shows that the point $\mathbf{p}_0^2(t)$ traces a quadratic (second-degree) curve with respect to the parameter t.
- This curve
  - is a quadratic Bézier curve (or a second-degree Bézier curve)
  - Will be denoted by $\mathbf{P}^2(t)$
- **Control points** of the Bézier curve: the initial points $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$
- $\mathbf{p}_i^r(t)$: r : the interpolation step

  i : the index of the first point being interpolated

II. *n*th-Degree Bézier Curves

- The process for the generation of a quadratic Bézier curve can be generalized for more control points.
- Cubic (or three-degree) Bézier curve $\mathbf{P}^3(t)$
  - Needs four control points
  - Performs three linear interpolation steps



- General case: *n*th-degree Bézier curve $\mathbf{P}^n(t)$
  - Need for $(n+1)$ control points $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_n$
  - Perform n linear interpolation steps
  - Control polygon of the curve: the polygon formed by $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_n$
  - The curve is given by:

$$\mathbf{P}^n(t) = \sum_{i=0}^{n} \binom{n}{i} t^i \, (1-t)^{n-i} \mathbf{p}_i, \quad t \in [0,1] \qquad \textbf{(1)}$$

III. The de Casteljau Algorithm

- Formula (1) is numerically complex & unstable (requires computations of binomial coefficients & powers of t and (1-t))
- The interpolation steps performed for the generation of the Bézier curve are simple linear relations of t.
- The de Casteljau algorithm:
  - Computes the Bézier curve points
  - Summarizes the linear interpolation steps in an iterative scheme
- Steps:
  1. For the required value of t, set $\mathbf{p}_i^0(t) = \mathbf{p}_i, \quad i = 0,1,...,n$
  2. Perform the linear interpolation steps:

$$\mathbf{p}_i^r(t) = (1-t)\ \mathbf{p}_i^{r-1}(t) + t\ \mathbf{p}_{i+1}^{r-1}(t), \quad r = 1,2,...,n$$

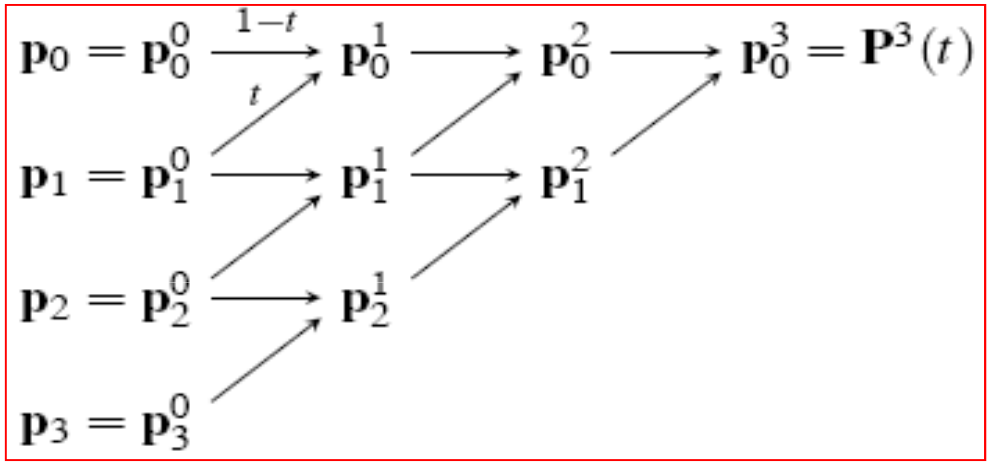
$$i = 0,1,...,n-r$$

  3. The point on the curve corresponding to parametric value t is

$$\mathbf{P}^n(t) = \mathbf{p}_0^n(t)$$

III. The de Casteljau Algorithm (2)

- The de Casteljau triangle: All the intermediate points involved in the de Casteljau algorithm can be written in a triangular arrangement
- For the case of a cubic Bézier curve, the triangle is



- Implementation:
  - No need to store all intermediate points
  - Use a one-dimensional array
  - Initialize it with the control points of the curve
  - Overwrite its elements from top to bottom as the algorithm progresses
  - At the end, the point on the curve will be the first element of the array

III. The de Casteljau Algorithm (3)

The de Casteljau Algorithm:

```
point bezierPoint ( int n, point[] controlPt, float t )
{
    point deCasPt[n+1];
    for (i=0; i <= n; i++)
        deCasPt[i] = controlPt[i];
    for (r=1; r <= n; r++)  {
        for (i=0; i <= n-r; i++)  {
            deCasPt[i] = (1-t)*deCasPt[i] + t*deCasPt[i+1];
        }
    }
    return deCasPt[0];
}
```

III. The de Casteljau Algorithm (4)

EXAMPLE 1: Bézier Curve Point Evaluation

Given a Bézier curve with control points $\mathbf{p}_0=[0\ \ 0]^T$, $\mathbf{p}_1=[2\ \ 2]^T$, $\mathbf{p}_2=[6\ \ 4]^T$ and $\mathbf{p}_3=[8\ \ 2]^T$ compute the point corresponding to the parametric value t = ¼.

SOLUTION

- The de Casteljau triangle is:
- Each point is an affine combination of two points from the column at its left
  - On the same line with coefficients ¾
  - On the next line with coefficient ¼

$$\begin{bmatrix}0\\0\end{bmatrix} \quad \begin{bmatrix}\frac{1}{2}\\\frac{1}{2}\end{bmatrix} \quad \begin{bmatrix}\frac{9}{8}\\1\end{bmatrix} \quad \begin{bmatrix}\frac{29}{16}\\\frac{23}{16}\end{bmatrix}$$

$$\begin{bmatrix}2\\2\end{bmatrix} \quad \begin{bmatrix}3\\\frac{5}{2}\end{bmatrix} \quad \begin{bmatrix}\frac{31}{8}\\\frac{11}{4}\end{bmatrix}$$

$$\begin{bmatrix}6\\4\end{bmatrix} \quad \begin{bmatrix}\frac{13}{2}\\\frac{7}{2}\end{bmatrix}$$

$$\begin{bmatrix}8\\2\end{bmatrix}$$

The required point is $\mathbf{P}^3\left(\frac{1}{4}\right)=\begin{bmatrix}\frac{29}{16}\\\frac{23}{16}\end{bmatrix}$

IV. Bernstein Polynomials

- The Bézier curve definition:

$$\mathbf{P}^n(t) = \sum_{i=0}^{n} \binom{n}{i} t^i \ (1-t)^{n-i} \mathbf{p}_i, \quad t \in [0,1]$$

- The coefficients of $\mathbf{p}_i$ are special polynomials called nth-degree Bernstein polynomials:

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad i = 0,1,2,...,n$$

- So the Bézier curve can be written:

$$\mathbf{P}^n(t) = \sum_{i=0}^{n} B_i^n(t)\mathbf{p}_i, \quad t \in [0,1]$$

- For the most common cases n=2 and n=3, the Bernstein polynomials are:

n=2

$$B_0^2(t) = (1-t)^2$$
$$B_1^2(t) = 2t(1-t)$$
$$B_2^2(t) = t^2$$

n=3

$$B_0^3(t) = (1-t)^3$$
$$B_1^3(t) = 3t(1-t)^2$$
$$B_2^3(t) = 3t^2(1-t)$$
$$B_3^3(t) = t^3$$

IV. Bernstein Polynomials: properties

- The *n*th-degree Bernstein polynomials constitute a basis of the vector space of the *n*th-degree polynomials:

  any *n*th-degree polynomial $f(t)$ can be written in the form:

$$f(t) = \sum_{i=0}^{n} B_i^n(t) c_i$$

  where c_i , $i=0,1,2,\ldots,n$ are suitable (scalar) coefficients

- Bernstein polynomials satisfy $\sum_{i=0}^{n} B_i^n(t) = 1 \quad , \forall\ t$ and

$$0 \le B_i^n(t) \le 1 \ , t \in [0,1]$$

- Bernstein polynomials are symmetric with respect to t and $1 - t$:

$$B_j^n(t) = B_{n-j}^n(1-t)$$

V. Properties of Bézier Curves

- Every *n*th-degree polynomial curve may be written in the form of a Bézier curve.
- Convex-hull property
  - The Bézier curve always lies inside the convex hull of its control points
- Invariance under affine transformations
  - The Bézier curve is an affine combination of its control points
  - Therefore, to apply an affine transformation to a Bézier curve, just transform its control points
- Invariance under affine transformations of its parameter
  - The curve remains unaltered if $t \in [0, 1]$ is changed to $u \in [a, b]$

    where $u = a + (b\text{-}a)\, t$
  - The interpolation steps become

$$\mathbf{p}_i^r(t) = \frac{b-u}{b-a}\mathbf{p}_i^{r-1}(t) + \frac{u-a}{b-a}\mathbf{p}_{i+1}^{r-1}(t)$$

V. Properties of Bézier Curves (2)

- Symmetry with respect to its control points
  - If the control points are used in reverse order $p_n,p_{n-1},\ldots,p_1,p_0$ the shape of the curve does not change
  - The curve is traversed in the opposite direction
- Linear precision
  - If all control points lie on a straight line, then the curve also has the shape of a straight line
- Variation-diminishing property
  - A planar Bézier curve is intersected by an arbitrary straight line no more than the number of times that the line intersects the control polygon of the curve.
  - A non-planar Bézier curve is intersected by an arbitrary line or plane no more than the number of times that the line or plane intersects the control polygon of the curve.
  - A curve with a convex control polygon is also convex but the inverse is not always true (a convex Bézier curve may have a non-convex control polygon)

V. Properties of Bézier Curves (3)

- Endpoint interpolation
  - The curve starts at its first control point and ends at its last one

$$\boldsymbol{P}^n(0) = \mathbf{p}_0 \quad \textit{and}\ \boldsymbol{P}^n(1) = \mathbf{p}_n$$

- Derivative
  - The tangent (first derivative) of a Bézier curve is: $\frac{d}{dt}\mathbf{P}^n(t) = n\sum_{i=0}^{n-1} B_i^{n-1}(t)(\mathbf{p}_{i+1} - \mathbf{p}_i)$
- Tangents at the endpoints
  - The tangent vector at the endpoints are parallel to the first and last edge of its control polygon: $\frac{d}{dt}\mathbf{P}^n(0) = n(\mathbf{p}_1 - \mathbf{p}_0)$ and $\frac{d}{dt}\mathbf{P}^n(1) = n(\mathbf{p}_n - \mathbf{p}_{n-1})$
  - If the curve is defined over an arbitrary parametric interval $u \in [a, b]$, then:

$$\frac{d}{du}\mathbf{P}^n(a) = \frac{d}{dt}\frac{dt}{du}\mathbf{P}^n(0) = \frac{1}{b-a}n(\mathbf{p}_1 - \mathbf{p}_0)$$

$$\frac{d}{du}\mathbf{P}^n(b) = \frac{d}{dt}\frac{dt}{du}\mathbf{P}^n(1) = \frac{1}{b-a}n(\mathbf{p}_n - \mathbf{p}_{n-1})$$

V. Properties of Bézier Curves (4)

- Second derivatives at the endpoints
  - The second derivatives at the endpoints of an *n*th-degree Bézier curve are:

$$\frac{d^2}{dt^2}\mathbf{P}^n(0) = n(n-1)(\mathbf{p}_2 - 2\mathbf{p}_1 + \mathbf{p}_0)$$

$$\frac{d^2}{dt^2}\mathbf{P}^n(1) = n(n-1)(\mathbf{p}_n - 2\mathbf{p}_{n-1} + \mathbf{p}_{n-2})$$

  - If the curve is defined over an arbitrary parametric interval $u \in [a, b]$, then:

$$\frac{d^2}{du^2}\mathbf{P}^n(a) = \frac{1}{(b-a)^2} n(n-1)(\mathbf{p}_2 - 2\mathbf{p}_1 + \mathbf{p}_0)$$

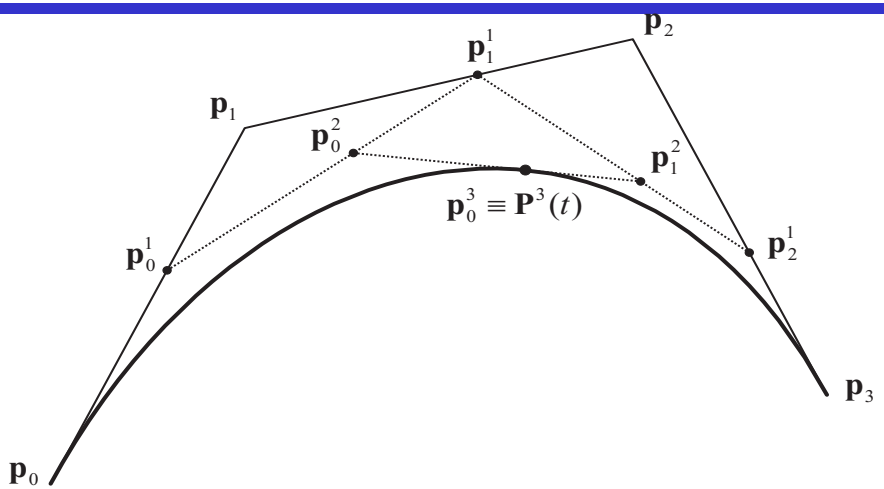
$$\frac{d^2}{du^2}\mathbf{P}^n(b) = \frac{1}{(b-a)^2} n(n-1)(\mathbf{p}_n - 2\mathbf{p}_{n-1} + \mathbf{p}_{n-2})$$

V. Properties of Bézier Curves (5)

- Pseudo-local control
  - *Local control*: moving a control point has a localized effect on the curve
  - Bézier curves do not possess local control since the Bernstein polynomials are non-zero over the whole parametric interval [0,1] of the curve
  - Moving any control point affects the shape of the whole curve
  - Bézier curves posses *pseudo*-local control:

    the effect of moving $\mathbf{p}_i$ is more pronounced around the parametric value *i/n*, where $B_i^n(t)$ has its only maximum
  - Makes it easier to predict the change of the shape of a Bézier curve when moving its control points

VI. Bézier Curve Subdivision



- Consider cubic Bézier curve $\mathbf{P}^3(t)$
- Any specific parametric value $t_0 \in$ [0,1] divides the curve into 2 segments:
  - The "left" one with endpoints $\mathbf{P}^3(0) = \mathbf{p}_0$ and $\mathbf{P}^3(t_0)$
  - The "right" one with end points $\mathbf{P}^3(t_0)$ and $\mathbf{P}^3(1) = \mathbf{p}_3$
- These segments are both cubic curves
- Can be written in Bézier curve form
- Aim: determine the Bézier control points of these 2 segments

VI. Bézier Curve Subdivision (2)

- First work on the left segment, **L**
  - Call its control points $\mathbf{l}_i$, i = 0, 1, 2, 3 so that $\mathbf{L}(t') = \sum_{i=0}^{3} B_i^3(t')\mathbf{l}_i, \quad t' \in [0,1]$
- Use a local parameter t' for **L**, $t \neq t'$, so that $\mathbf{L}(t')$ traces the whole left segment when $t' \in [0,1]$
- The curve interpolates its endpoints, so the first and last control point can be determined immediately:

$$\mathbf{l}_0 = \mathbf{p}_0 = \mathbf{P}^3(0) = \mathbf{p}_0^0(t_0) \quad , \; \mathbf{l}_3 = \mathbf{P}^3(t_0) = \mathbf{p}_0^3(t_0)$$

- For $\mathbf{l}_1$: it is involved in the tangent of $\mathbf{L}(t')$ for $t' = 0$

  $\mathbf{P}^3$ and **L** have the same tangent at this point, since they coincide when $t \in [0, t_0]$, so

$$\frac{d}{dt}\mathbf{P}^3(0) = \frac{d}{dt}\mathbf{L}(0) = \frac{d}{dt'}\frac{dt'}{dt}\mathbf{L}(0) \Leftrightarrow 3(\mathbf{p}_1 - \mathbf{p}_0) = \frac{1}{t_0}3(\mathbf{l}_1 - \mathbf{l}_0)$$

$$\Leftrightarrow \mathbf{l}_1 = (1-t_0)\mathbf{p}_0 + t_0\mathbf{p}_1$$

VI. Bézier Curve Subdivision (3)

- Using the notation of the de Casteljau algorithm

$$\mathbf{l}_1 = \mathbf{p}_0^1(t_0)$$

- Similarly, $\mathbf{l}_2$ is involved in the second derivative of $\mathbf{L}(t')$, for t=0:

$$6(\mathbf{p}_2 - 2\mathbf{p}_1 + \mathbf{p}_0) = \frac{1}{t_0^2} 6(\mathbf{l}_2 - 2\mathbf{l}_1 + \mathbf{l}_0)$$

$$\Leftrightarrow \mathbf{l}_2 = (1-t_0)^2\mathbf{p}_0 + 2t_0(1-t_0)\mathbf{p}_1 + t_0^2\mathbf{p}_2$$

- Using the notation of the de Casteljau algorithm

$$\mathbf{l}_2 = \mathbf{p}_0^2(t_0)$$

- The above relations can be written as: $\mathbf{l}_i = \mathbf{p}_0^i(t_0)$

- The control points $\mathbf{l}_i$ of the left segment are exactly the points of the first line of the de Casteljau triangle.
- The above property holds for Bézier curves of any degree *n*.

VI. Bézier Curve Subdivision (4)

- The right segment:
  - Is the part that corresponds to the interval $[t_0,1]$ of the curve
  - Its control points $\mathbf{r}_i$ can be computed by working similarly , so:

$$\mathbf{r}_i = \mathbf{p}_i^{n-i}(t_0)$$

  - They are the points of the “hypotenuse” of the de Casteljau triangle
  - This result holds for Bézier curves of any degree.

- The de Casteljau algorithm:
  - Not only computes the points to the curve that corresponds to t_0
  - But also provides the control points of the 2 segments into which the curve is subdivided by this point
  - The implementation readily provides points $\mathbf{r}_i$ (they are the deCasPt)
  - To provide points $\mathbf{l}_i$: after each column of the triangle is completed, its 0-index element is a control point of the left segment

VI. Bézier Curve Subdivision- Applications

- The subdivision of a Bézier curve can be repeated recursively for each of the two segments of the curve
- The control points generated during this recursion converge to initial curve
- This convergence is rather fast
- The result has interesting practical applications:

❖ It is a way to draw Bézier curves

- Subdivide recursively the curve into two segments
- After a number of steps the control points will be nearly collinear
- Then the subdivision may stop
- Each segment can be drawn as a line segment between its two endpoints

VI. Bézier Curve Subdivision- Applications (2)

- This process is *adaptive*
  - draw its flat regions quickly
  - perform more recursive steps wherever the curve has high curvature
- Subdivision may be performed at any value t_0
  - Prefer t_0=1/2 because interpolation steps involve only divisions by 2
- Test the collinearity of the control points $\mathbf{p}_0$, $\mathbf{p}_n$ by
  - computing the distance of every other control points $\mathbf{p}_i$, i=1,2,…,n-1
  - ensuring that every such distance is less than the required tolerance
- The distance of a control point $\mathbf{p}_i$ from the line through $\mathbf{p}_0$ and $\mathbf{p}_n$:

$$d = \frac{\left|(\mathbf{p}_n - \mathbf{p}_0) \times (\mathbf{p}_n - \mathbf{p}_i)\right|}{\left|\mathbf{p}_n - \mathbf{p}_0\right|}$$

VI. Bézier Curve Subdivision- Applications (3)

❖ Finding the intersection of a Bézier curve with a line

- Recursive algorithm:

```
Construct the axis-aligned bounding box (AABB) of the initial curve
Check the (AABB) for intersections with the line
if exists an intersection
     subdivide the curve into its left & right segment
     continue recursively with the AABB of each segment
```

- Subdivision stops when the AABB is too small (for given tolerance)
- The algorithm works correctly:
  - If the line does not intersect the AABB, it will not intersect the curve

VII.Smoothly Joining Bézier Curves

- The complexity of the shape of a Bézier curve is restricted by the number of its control points.
- To draw complex shapes we need Bézier curves of high degree
- The use of high-degree Bézier curves is not advisable:
  - The higher the degree, the less efficient is to compute it

    (more linear interpolation steps, numerical accuracy problems)
  - The lack of local control makes it difficult to create a desirable shape with a single Bézier curve
- Bézier curves of degree > 5 are not used in practice
- Solution: use low-degree curves joined in such a way that the resulting shape is smooth
- Most common form is B-spline curve (presented later)
- Here, we discuss the joining of 2 Bézier curves smoothly

VII. Smoothly Joining Bézier Curves (2)

- Two polynomial curves $\mathbf{F}(t)$, $t \in [t_0,t_1]$ and $\mathbf{G}(t)$, $t \in [t_1,t_2]$ join with parametric continuity C^r at t_1 if their rth-order derivatives are equal at t_1:

$$\mathbf{F}^{(r)}(t_1) = \mathbf{G}^{(r)}(t_1)$$

- C^r continuity at a point → C^m continuity for all $0 \le m < r$
- E.g. if $\mathbf{F}(t)$ and $\mathbf{G}(t)$ join at t_1 with C^2 continuity, then their
  - values (C^0),
  - tangents (C^1) and
  - second derivatives (C^2)

  are equal at t_1
- This join is "smooth": the slope of the curve at the join does not change abruptly
- For kth-degree polynomial curves look for continuity up to C^{k-1}, because their kth-order derivatives are constant & higher order ones are 0.

VII. Smoothly Joining Bézier Curves (3)

- Consider two Bézier curves:
  - $\mathbf{P}^n(t)$, $t \in [0,1]$ of degree n with control points $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_n$
  - $\mathbf{Q}^m(t)$, $t \in [1,2]$ of degree m with control points $\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_m$
- Which are the conditions for the curves to join at $t=1$ with C^2 continuity?

1. C^0 continuity: $\mathbf{p}_n = \mathbf{q}_0$
2. C^1 continuity: $n(\mathbf{p}_n - \mathbf{p}_{n-1}) = m(\mathbf{q}_1 - \mathbf{q}_0) \Leftrightarrow \mathbf{q}_1 - \mathbf{p}_n = \frac{n}{m}(\mathbf{p}_n - \mathbf{p}_{n-1})$ (1)

   $\mathbf{q}_1$ must be placed at the line ($\mathbf{p}_{n-1}\mathbf{p}_n$)and its distance from $\mathbf{p}_n$ is (1)
3. C^2 continuity: $n(n-1)(\mathbf{p}_n - 2\mathbf{p}_{n-1} + \mathbf{p}_{n-2}) = m(m-1)(\mathbf{q}_2 - 2\mathbf{q}_1 + \mathbf{q}_0)$ (2)

   to define the position of $\mathbf{q}_2$

- If $n = m$ then (1) becomes $\mathbf{q}_1 - \mathbf{p}_n = \mathbf{p}_n - \mathbf{p}_{n-1}$ so $|\mathbf{q}_1\mathbf{p}_n| = |\mathbf{p}_n\mathbf{p}_{n-1}|$

VII. Smoothly Joining Bézier Curves (4)

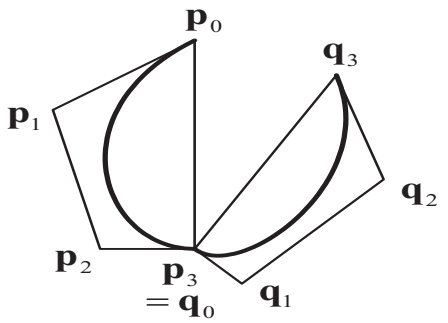
- If $n = m = 3$ then (2) becomes: $\mathbf{q}_2 - 2\mathbf{q}_1 = \mathbf{p}_1 - 2\mathbf{p}_2$

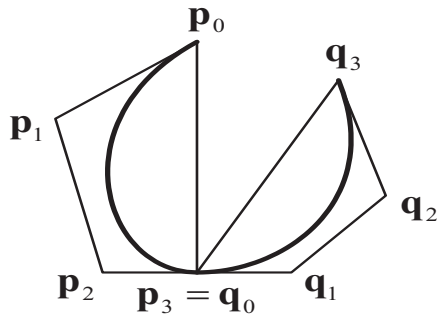
  If $\mathbf{d} = \mathbf{q}_2 - 2\mathbf{q}_1 = \mathbf{p}_1 - 2\mathbf{p}_2$ then

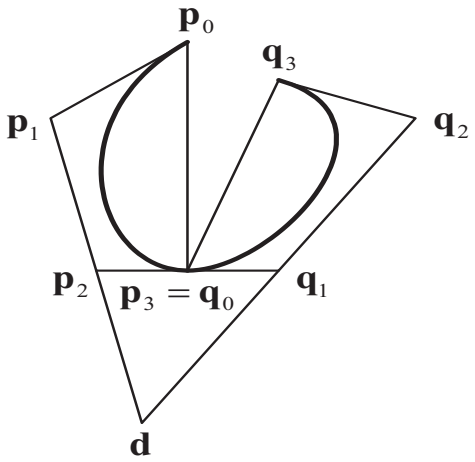
  $\mathbf{d}$ must be placed so that $|\mathbf{dp}_2| = |\mathbf{p}_2\mathbf{p}_1|$

  $\mathbf{q}_2$ must be placed so that $|\mathbf{dq}_1| = |\mathbf{q}_1\mathbf{q}_2|$

- Each additional degree of continuity restricts the position of one more control point.
- For $n=m=3$ with C^2 continuity, only the position of $\mathbf{p}_0$ and $\mathbf{q}_3$ is free.
- For greater flexibility:
  - Use higher-degree curves (not efficient)
  - Require lower continuity (not satisfying results)
  - Modify the kind of continuity required : instead of parametric continuity (C^r) require the weaker geometric continuity (G^r)







B-Spline Curves

- Spline curves: curves generated by joining parametric curves with continuity constraints (cf. last section)
- B-spline curves:
  - Spline curves comprised of polynomial segments of degree k joined with C^{k-1} continuity
  - The degree k of the segments is also the degree of the B-spline curve
  - Are defined by control points $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_n$
  - The number $(n+1)$ of control points is independent of the curve degree and related to the number of the polynomial segments
  - The polynomial segments are defined over consecutive parametric intervals $[t_i, t_{i+1}]$
  - Domain of the B-spline curve: the union $[t_{min}, t_{max}]$ of the intervals

B-Spline Curves (2)

- Knots of the curve: the values t_i at the boundaries of the intervals
- A B-spline curve has a knot sequence

$$t_{first} \leq \ldots \leq t_{min} \leq \ldots \leq t_{max} \leq \ldots \leq t_{last}$$

  because its definition requires some additional knots outside its domain.
- The number of knots depends on:
  - The degree of the curve
  - The number of its control points

B-Spline Curves

I. Quadratic B-Spline Curves
II. *k*th-Degree B-Spline Curves
III. B-Spline Functions
IV. The de Boor Algorithm
V. Knots and Parameterizations
VI. Properties of B-Spline Curves
VII. B-Spline Curves in Bézier Form

I. Quadratic B-Spline Curves

- A quadratic B-spline curve $\mathbf{Q}(t)$ is comprised of quadratic segments $\mathbf{Q}_i(t)$, defined in parametric intervals $[t_i, t_{i+1}]$
- Use 3 control points for each segment
- The 2 linear interpolation steps differ from those for Bézier curves
- For segment $\mathbf{Q}_i(t)$, use control points $\mathbf{p}_{i-2}$, $\mathbf{p}_{i-1}$ and $\mathbf{p}_i$

Step 1:

- Interpolate separately
  - $(\mathbf{p}_{i-2}, \mathbf{p}_{i-1})$ in the interval $[t_{i-1}, t_{i+1}]$ and
  - $(\mathbf{p}_{i-1}, \mathbf{p}_i)$ in the interval $[t_i, t_{i+2}]$

  to get the intermediate points:

$$\mathbf{q}_{i-1}^1(t) = \frac{t_{i+1} - t}{t_{i+1} - t_{i-1}} \mathbf{p}_{i-2} + \frac{t - t_{i-1}}{t_{i+1} - t_{i-1}} \mathbf{p}_{i-1}, \qquad t \in [t_{i-1}, t_{i+1}]$$

$$\mathbf{q}_i^1(t) = \frac{t_{i+2} - t}{t_{i+2} - t_i} \mathbf{p}_{i-1} + \frac{t - t_i}{t_{i+2} - t_i} \mathbf{p}_i, \qquad t \in [t_i, t_{i+2}]$$

I. Quadratic B-Spline Curves (2)

Step 2:

- Interpolate $\mathbf{q}_{i-1}^1(t)$ and $\mathbf{q}_i^1(t)$ in the interval $[t_i, t_{i+1}]$:

$$\mathbf{Q}_i(t) = \mathbf{q}_i^2(t) = \frac{t_{i+1} - t}{t_{i+1} - t_i}\mathbf{q}_{i-1}^1(t) + \frac{t - t_i}{t_{i+1} - t_i}\mathbf{q}_i^1(t), \quad t \in [t_i, t_{i+1}] \qquad (1)$$

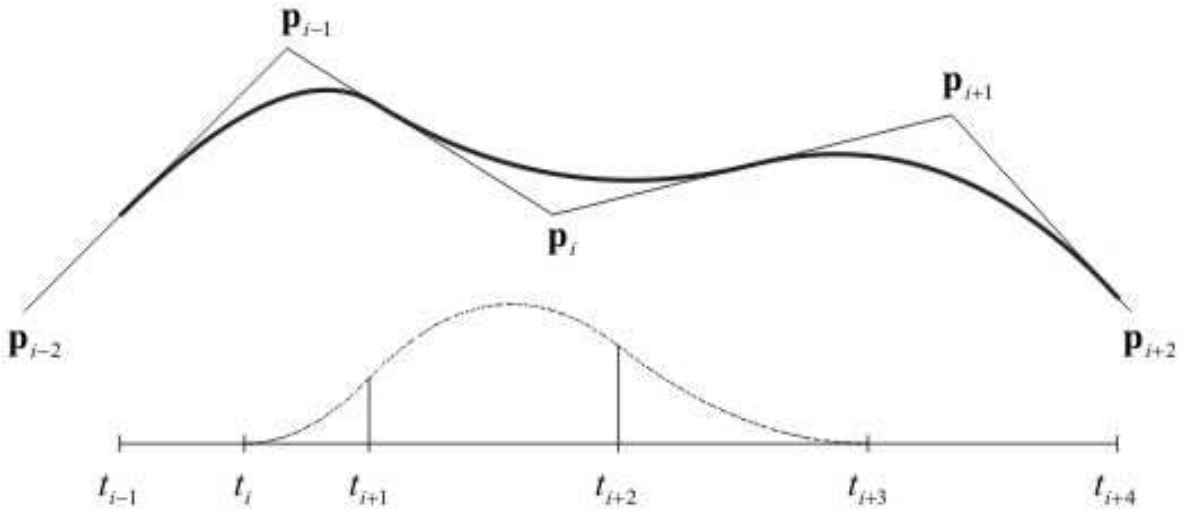
- Substitute $\mathbf{q}_{i-1}^1(t)$ and $\mathbf{q}_i^1(t)$ to get:

$$\mathbf{Q}_i(t) = \frac{t_{i+1} - t}{t_{i+1} - t_i}\frac{t_{i+1} - t}{t_{i+1} - t_{i-1}}\mathbf{p}_{i-2} + \left(\frac{t_{i+1} - t}{t_{i+1} - t_i}\frac{t - t_{i-1}}{t_{i+1} - t_{i-1}} + \frac{t - t_i}{t_{i+1} - t_i}\frac{t_{i+2} - t}{t_{i+2} - t_i}\right)\mathbf{p}_{i-1} + \frac{t - t_i}{t_{i+1} - t_i}\frac{t - t_i}{t_{i+2} - t_i}\mathbf{p}_i$$

- C^1 continuity:
  - They satisfy $\mathbf{Q}_i(t_{i+1}) = \mathbf{Q}_{i+1}(t_{i+1})$ and $\mathbf{Q}_i'(t_{i+1}) = \mathbf{Q}_{i+1}'(t_{i+1})$
  - They join at t_{i+1} (C^0)
  - Their tangents coincide (C^1)

I. Quadratic B-Spline Curves (3)

- (1) is not useful, provides a piecewise expression
- Need for a uniform expression for the whole curve in terms of:
  - Its control points
  - Its knots
- Consider a single control point $\mathbf{p}_i$
- $\mathbf{p}_i$ affects three consecutive segments of the curve $\mathbf{Q}_i(t)$, $\mathbf{Q}_{i+1}(t)$ and $\mathbf{Q}_{i+2}(t)$



I. Quadratic B-Spline Curves (4)

- Rewrite (1) for segments $\mathbf{Q}_i(t)$, $\mathbf{Q}_{i+1}(t)$ and $\mathbf{Q}_{i+2}(t)$

$$\mathbf{Q}_i(t) = a_i \mathbf{p}_{i-2} + b_i \mathbf{p}_{i-1} + \frac{t-t_i}{t_{i+1}-t_i}\frac{t-t_i}{t_{i+2}-t_i}\mathbf{p}_i, \qquad t \in [t_i, t_{i+1}],$$

$$\mathbf{Q}_{i+1}(t) = a_{i+1}\mathbf{p}_{i-1} + \left(\frac{t_{i+2}-t}{t_{i+2}-t_{i+1}}\frac{t-t_i}{t_{i+2}-t_i} + \frac{t-t_{i+1}}{t_{i+2}-t_{i+1}}\frac{t_{i+3}-t}{t_{i+3}-t_{i+1}}\right)\mathbf{p}_i + c_{i+1}\mathbf{p}_{i+1}, \qquad t \in [t_{i+1}, t_{i+2}],$$

$$\mathbf{Q}_{i+2}(t) = \frac{t_{i+3}-t}{t_{i+3}-t_{i+2}}\frac{t_{i+3}-t}{t_{i+3}-t_{i+1}}\mathbf{p}_i + b_{i+2}\mathbf{p}_{i+1} + c_{i+2}\mathbf{p}_{i+2}, \qquad t \in [t_{i+2}, t_{i+3}]$$

- If we set

$$N_i^2(t) = \begin{cases} \frac{t-t_i}{t_{i+1}-t_i}\frac{t-t_i}{t_{i+2}-t_i}, & t \in [t_i, t_{i+1}), \\ \frac{t_{i+2}-t}{t_{i+2}-t_{i+1}}\frac{t-t_i}{t_{i+2}-t_i} + \frac{t-t_{i+1}}{t_{i+2}-t_{i+1}}\frac{t_{i+3}-t}{t_{i+3}-t_{i+1}}, & t \in [t_{i+1}, t_{i+2}), \\ \frac{t_{i+3}-t}{t_{i+3}-t_{i+2}}\frac{t_{i+3}-t}{t_{i+3}-t_{i+1}}, & t \in [t_{i+2}, t_{i+3}), \\ 0, & \text{everywhere else,} \end{cases}$$

then the effect of $\mathbf{p}_i$ on the whole curve is $N_i^2(t)\mathbf{p}_i$

I. Quadratic B-Spline Curves (5)

- The quadratic B-spline curve can be written by summing the effects of all control points: $\mathbf{Q}(t) = \sum_{i=0}^{n} N_i^2(t)\mathbf{p}_i$ (3)

- Determine necessary number of knots for quadratic B-spline curve:
  - Control points $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_n$
  - First segment $\mathbf{Q}_2(t)$ uses $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2$ and requires knots t_1, t_2, t_3, t_4
  - Last segment $\mathbf{Q}_n(t)$ uses $\mathbf{p}_{n-2}, \mathbf{p}_{n-1}, \mathbf{p}_n$ and requires knots $t_{n-1}, t_n, t_{n+1}, t_{n+2}$
- A quadratic B-spline curve with $(n+1)$ control points $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_n$ requires $(n+2)$ knots $t_1, t_2, \ldots, t_{n+2}$
- The domain of the curve is $[t_2, t_{n+1}]$
- Two more knots are required (t_0, t_{n+3}) to define $N_0^2(t)$ and $N_n^2(t)$
  - These knots don't contribute to the shape of the curve

II. *k*th-Degree B-Spline Curves

- Generalize the process for the generation of a quadratic B-spline curve for higher-degree curves.
- A segment $\mathbf{Q}_i(t)$, $t \in [t_i, t_{i+1}]$ of a *k*th-degree B-spline requires *k* linear interpolation steps.
- Step 1:
  - interpolate $(k+1)$ control points $\mathbf{p}_{i-k}, \mathbf{p}_{i-k+1}, \mathbf{p}_i$ in pairs
  - result: points $\mathbf{q}_j^1(t)$, $j=i-k+1, i-k+2,\ldots, i$, $t \in [t_j, t_{j+k}]$ on linear segments
- Steps $r = 2,3,\ldots,k$:
  - interpolate the points of previous step
  - result: points $\mathbf{q}_j^r(t)$, $j=i-k+r, \ldots, i$, $t \in [t_j, t_{j+k-r+1}]$ on shrinking t
- After *k* steps: a single *k*th-degree segment
- Consecutive segments $\mathbf{Q}_i(t)$: join with C^{k-1} continuity and form a *k*th-degree B-spline curve

II. *k*th-Degree B-Spline Curves (2)

- Express the whole curve in terms of its control points & knots:
  - Each point $\mathbf{p}_i$ affects $(k+1)$ segments $\mathbf{Q}_j(t)$, $j=i, i+1, \ldots, i+k$
  - Construct a function $N_i^k(t)$ that expresses its contribution to the curve
- $N_i^k(t)$ are called *k*th-degree B-spline functions
- The B-spline is written as: $\mathbf{Q}(t) = \sum_{i=0}^{n} N_i^k(t)\mathbf{p}_i$
- It is comprised of $(n-k+1)$ polynomial segments of degree k
- Each segment is defined over the interval $[t_i, t_{i+1}]$, $i=k, k+1, \ldots, n$
- The domain of the curve is $[t_k, t_{n+1}]$
- There are $(n + k)$ knots $t_1, t_2, \ldots, t_{n+k}$
  - "Dummy" Knots t_0 and t_{n+k+1} needed for the definition of the $N_i^k(t)$ but do not affect the shape of the curve
  - Knots may be repeated
  - At most k consecutive knots may be equal, $t_i < t_{i+k}$

III. B-Spline Functions

- $N_i^k(t)$: kth-degree B-spline functions:

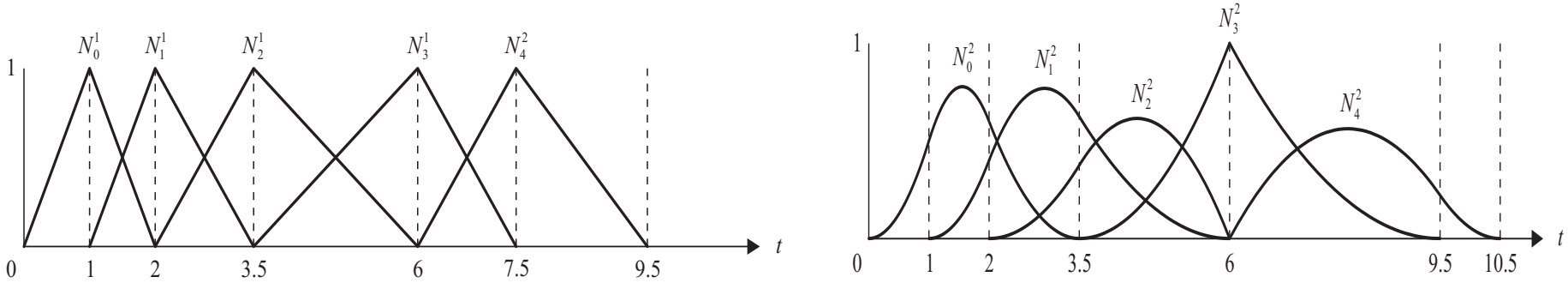
$$N_i^0(t) = \begin{cases} 1, & t \in [t_i, t_{i+1}) \\ 0, & \text{everywhere else} \end{cases}$$

- For $r = 1,2,\ldots,k$ and $i=0,1,\ldots,n+k-r$:

$$N_i^r(t) = \frac{t - t_i}{t_{i+r} - t_i} N_i^{r-1}(t) + \frac{t_{i+r+1} - t}{t_{i+r+1} - t_{i+1}} N_{i+1}^{r-1}(t)$$

- $N_i^r(t)$ are
  - r-degree polynomials of t with local support on $[t_i, t_{i+r+1})$, i.e. 0 everywhere else
  - Spline functions because they consist of $(r+1)$ polynomial segments of degree r joined with C^{r-1} continuity at the knots

III. B-Spline Functions



- Each first-degree $N_i^1(t)$ consists of 2 linear segments joined at t_{i+1} (C^0)
- Each second-degree $N_i^2(t)$ consists of 3 quadratic segments joined with C^1 continuity at t_{i+1} and t_{i+2}
- Given a degree k and knots $t_0, t_1, \ldots, t_{n+k}, t_{n+k+1}$, all the spline functions defined over this knot sequence constitute a vector space
- The $(n+1)$ B-spline functions of degree k defined over this knot form a basis of this vector space
- B-splines → Basis splines

IV. The de Boor Algorithm

- The de Casteljau algorithm computes points on a Bézier curve
- The de Boor algorithm:
  - Computes points on a B-spline curve
  - Summarizes the linear interpolation steps
- Due to the local support, we must know the interval to which the requested t belongs in order to compute $\mathbf{Q}(t)$

Steps :

1. For the required value of t, find the parametric interval $[t_i, t_{i+1})$. The domain is $[t_k, t_{n+1}]$ so i must satisfy $k \le i \le n$
2. Set $\mathbf{q}_j^0(t) = \mathbf{p}_j, \quad j = i-k, i-k+1, \ldots, i$
3. Perform the linear interpolation steps:

$$\mathbf{q}_j^r(t) = \frac{t_{k-r+1+j} - t}{t_{k-r+1+j} - t_j}\mathbf{q}_{j-1}^{r-1}(t) + \frac{t - t_j}{t_{k-r+1+j} - t_j}\mathbf{q}_j^{r-1}(t), \quad \begin{matrix} r = 1, 2, \ldots, k \\ j = i-k+r, i-k+r+1, \ldots, i \end{matrix}$$

4. The point of the curve corresponding to t is: $\mathbf{Q}(t) = \mathbf{q}_i^k(t)$

IV. The de Boor Algorithm (2)

- All the intermediate points can be written in a triangular arrangement
- For the case of a cubic B-spline curve:

$$
\begin{array}{llll}
\mathbf{p}_{i-3} = \mathbf{q}_{i-3}^{0} & & & \\
\mathbf{p}_{i-2} = \mathbf{q}_{i-2}^{0} & \mathbf{q}_{i-2}^{1} & & \\
\mathbf{p}_{i-1} = \mathbf{q}_{i-1}^{0} & \mathbf{q}_{i-1}^{1} & \mathbf{q}_{i-1}^{2} & \\
\mathbf{p}_{i} = \mathbf{q}_{i}^{0} & \mathbf{q}_{i}^{1} & \mathbf{q}_{i}^{2} & \mathbf{q}_{i}^{3} = \mathbf{Q}(t)
\end{array}
$$

- Differences from the de Casteljau triangle:
  - The coefficients involved in the linear interpolation steps are **not constant** (1-*t* and *t*) – they depend on the specific row and column
  - When implemented with a one-dimensional array: the intermediate points must be computed from bottom to top so that the required points are not overwritten

IV. The de Boor Algorithm (3)

The de Boor Algorithm:

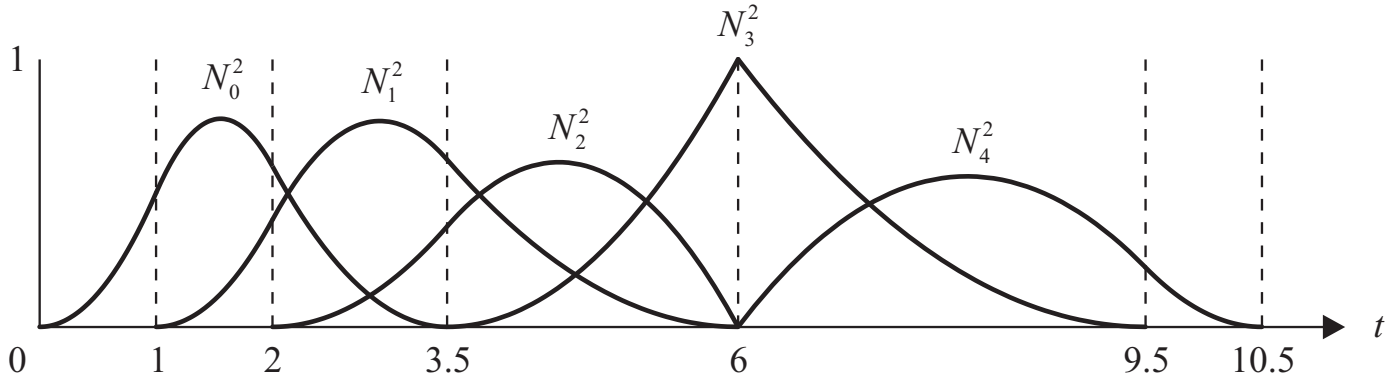
```
for (j = i-k; j <= i; j++)  {
    m = j-i+k;                          //so that m = 0, 1, …, k-r
    deBoorPt[m] = controlPt[j];
}
for (r = 1; r <= k; r++)  {
    for (j = i; j >= i-k+r; j--)  {
        m = j-i+k;
        coeff = (t-knots[j]) / (knots[k-r+1+j] - knots[j]);
        deBoorPt[m] = (1-coeff)*deBoorPt[m-1]
                                        + coeff*deBoorPt[m];
    }
}
```

V. Properties of the Knots

- The knot sequence of a B-spline curve directly affects its shape
  - Using the same control points and different knot sequences, the shape of the curves will vary.
- A knot has multiplicity k : it is repeated k times
- If for a kth-degree B-spline the first knot has multiplicity k ($t_1 = t_2 = \ldots = t_{k-1} = t_k$) then
  - $\mathbf{Q}(t_1) = \mathbf{Q}(t_k) = \mathbf{q}_0$
  - The curve interpolates its first control point
- If the last knot has multiplicity k, the curve interpolates its last control point $\mathbf{p}_n$.
- Open or clamped: is a knot sequence if the first and last k knots are equal
- Unlike Bézier curves, B-splines interpolate their extreme control points only when a clamped knot sequence is used

V. Properties of the Knots (2)

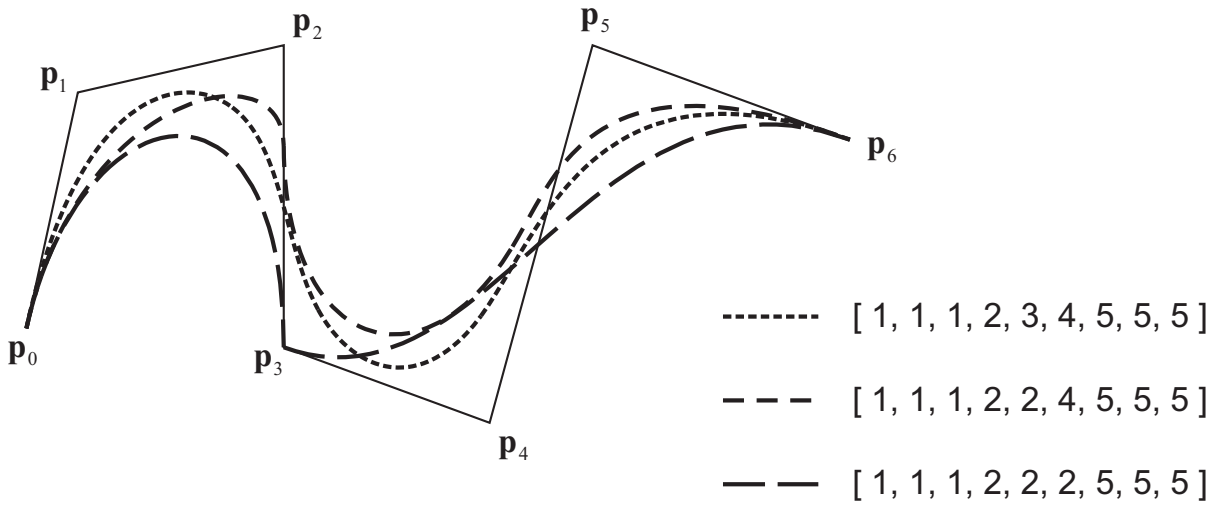
- If a knot is repeated, $t_i = t_{i+1}$, the curve loses 1 degree of continuity at $\mathbf{Q}(t_i)$
- If a knot has multiplicity r, the curve is C^{k-r} at that point



- The B-spline functions above are not C^1 continuous everywhere but C^0 continuous at the double knot.

V. Properties of the Knots (3)

- As the multiplicity of a knot increases, the curve approaches its control polygon in the neighborhood of this knot
- When multiplicity of the knot == degree of the curve :
  - A cusp is formed at that control point since the curve is only C^0 continuous
- The above property helps in practice, to better control the shape of the curve
- No knot can be repeated more than k times
  - lower than C^0 continuity → discontinuity on the curve



V. Parameterizations

- Knot sequence can:
  - Be supplied by the user
  - Be generated automatically
- Parameterizations: specific algorithms that generate the knot sequence
- They try to produce a well-shaped curve

V. Parameterizations (2)

❖ **Uniform parameterization**

- The knots are equidistant
- Set: $t_i = i\text{-}1, \quad i = 1, 2, \ldots, n+k$

❖ **Clamped uniform parameterization**

- Used so that the curve interpolates its first & last endpoints

$$t_i = \begin{cases} 0, & i = 1, \ldots, k \\ i-k, & i = k+1, \ldots, n \\ n-k+1, & i = n+1, \ldots, n+k \end{cases}$$

- Uniform knot sequences:
  - Generate visually acceptable curves in most cases
  - They don't take into account the shape of the curve
  - May not produce well-shaped, "smooth" curves
    - eg. if control points are close to each other at areas where the curvature changes abruptly

V. Parameterizations (3)

❖ **Chord-length parameterization**

- Often used in practice
- The distances between the knots are proportional to the distances between corresponding control points

$$t_i = \begin{cases} 0, & i = 1,\ldots,k \\ t_{i-1} + \left|\mathbf{p}_{i-k} - \mathbf{p}_{i-k-1}\right|, & i = k+1,\ldots,n \\ \sum_{j=0}^{n-k} \left|\mathbf{p}_{j+1} - \mathbf{p}_{j}\right|, & i = n+1,\ldots,n+k \end{cases}$$

- Different chord-length parameterizations can be produced by:
  - Changing the extreme knots

    and / or
  - Not requiring that the curve interpolates its first and last control points

V. Parameterizations (4)

❖ **Centripetal parameterization**

- The distances between the knots are proportional to the square root of the distances between corresponding control points

$$t_i = \begin{cases} 0, & i = 1,\ldots,k \\ t_{i-1} + \sqrt{\left|\mathbf{p}_{i-k} - \mathbf{p}_{i-k-1}\right|}, & i = k+1,\ldots,n \\ \sum_{j=0}^{n-k} \sqrt{\left|\mathbf{p}_{j+1} - \mathbf{p}_{j}\right|}, & i = n+1,\ldots,n+k \end{cases}$$

- Different centripetal parameterizations can be produced by:
  - Changing the end conditions

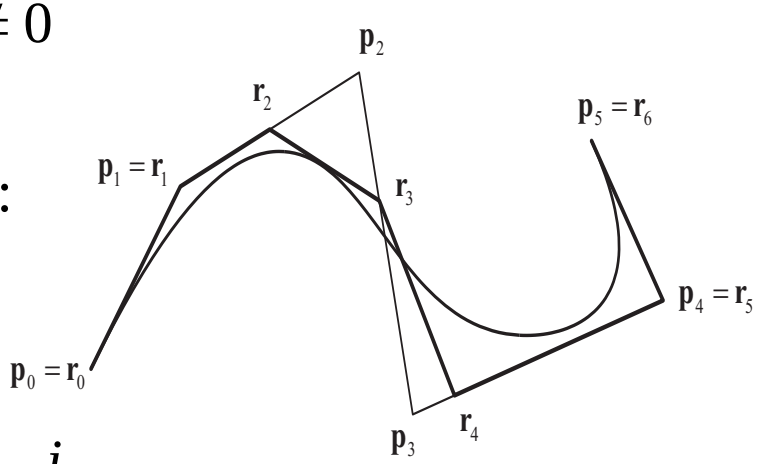
V. Knot insertion

- Add a knot sequence while maintaining its shape
- When inserting a knot one of the following must increase by 1:
  - The degree of the curve **OR**
  - The number of its control points

  so that the correlation between the degree, the number of control points and knots is maintained.
- High-degree curves are difficult to compute & manipulate, so prefer to:
  - Add a new control point and
  - Move some others to maintain the shape of the curve
- The new knot & control point provide grater flexibility to the shape
- As knots are inserted, the control polygon comes closer to the curve

V. Knot insertion (2)

- Insert a new knot s between knots t_i and t_{i+1}:
  - The new knot sequence will be $t_1, \ldots, t_i, s, t_{i+1}, \ldots, t_{n+k}$
- Let $\boldsymbol{r}_j$, $j=0, 1, \ldots, n+1$ be the new control points
- Only k of the new control points will differ because a control point $\mathbf{p}_i$ only affects the curve on $[t_i, t_{i+k+1}]$ where $N_i^k(t) \neq 0$



- The control points that are not affected are:

$$\mathbf{r}_j = \mathbf{p}_j, \quad j = 0,\ldots,i-k$$

$$\mathbf{r}_j = \mathbf{p}_{j-1}, \quad j = i+1,\ldots,n+1$$

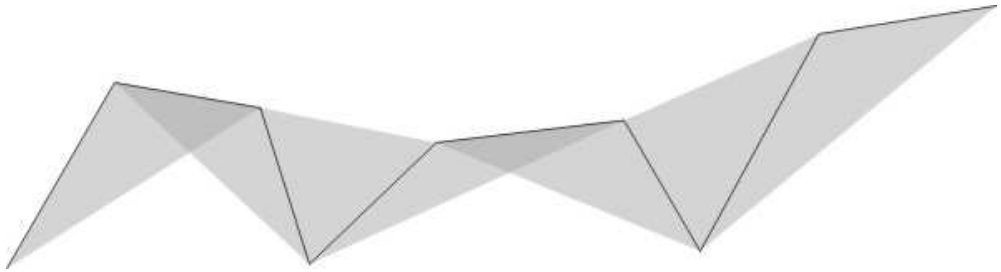
- The new control points are $\boldsymbol{r}_j$, $j = i\text{-}k+1, \ldots, i$

$$\mathbf{r}_j = \frac{t_{j+k} - s}{t_{j+k} - t_j}\mathbf{p}_{j-1} + \frac{s - t_j}{t_{j+k} - t_j}\mathbf{p}_j, \quad j = i-k+1,\ldots,i$$

- This formula is called Boehm's knot-insertion formula

VI. Properties of B-Spline Curves

- Local control
  - B-splines exhibit full local control
  - Changing a control point, affects the shape of the curve only on a restricted segment
  - $\mathbf{p}_i$ affects only the part of the curve corresponding to $[t_i, t_{i+k+1}]$: $N_i^k(t) \neq 0$
- Strong convex-hull property
  - All the interpolation steps performed, are convex combinations of the control points
  - Every point on the curve lies inside the convex hull of the $(k+1)$ control points that contribute to its computation
  - The whole curve lies inside the union of these convex hulls



VI. Properties of B-Spline Curves (2)

- Invariance under affine transformations
  - The B-spline curve is an affine combination of its control points
  - To apply an affine transformation to a B-spline transform only its control points
  - Parameterizations are **not** maintained
  - Only the uniform parameterization is maintained
- Invariance under affine transformations of its parameter
  - The curve remains invariant if $u = a + (b\text{-}a)\, t$
  - Parameterizations are not maintained, except the uniform
- Strong linear precision
  - If the control points lie on a straight line, then the curve degenerates to a straight line
  - If (k+1) control points are collinear, the corresponding segment is a straight line segment

VI. Properties of B-Spline Curves (3)

- Strong variation-diminishing property
  - General form:
    - A planar B-spline curve may not be intersected by an arbitrary straight line more times than its control polygon
    - A non-planar B-spline curve may not be intersected by a straight line or plane more times than its control polygon
  - Stronger form
    - The above hold for the polygon formed by the $(k+1)$ control points that contribute to any point on the curve
- Endpoint interpolation
  - A B-spline curve interpolates its extreme control points only if a clamped knot sequence is used

VI. Properties of B-Spline Curves (4)

- Derivative
  - The tangent (first derivative) of a *k*th-degree B-spline curve is:

$$\frac{d}{dt}\mathbf{Q}(t) = k\sum_{i=0}^{n-1} N_i^{k-1}(t)\frac{1}{t_{i+k+1} - t_{i+1}}(\mathbf{p}_{i+1} - \mathbf{p}_i)$$

- Generation of Bézier curves
  - A B-spline of degree k with $(k+1)$ control points $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_k$ and knot sequence $(0^{<k>}, 1^{<k>})$ is a Bézier curve of degree k and control points $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_k$
  - $x^{<k>}$: a knot with multiplicity k

VII. B-Spline Curves in Bézier Form

- A *k*th-degree B-spline is composed of several *k*th-degree polynomial segments defined over $[t_i, t_{i+1}]$
- Each of these segments can be written as a *k*th-degree Bézier curve
- Obtaining a piecewise Bézier form of a B-spline curve is useful:
  - Bézier curves are simpler and well-studied
  - Draw a B-spline curve by drawing the Bézier curves

VII. Bézier Form of Quadratic B-splines

- Consider :
  - A quadratic (k=2) B-spline $\mathbf{Q}(t)$
  - A specific segment $\mathbf{Q}_i(t)$ defined from (B-spline) control points $\mathbf{p}_{i-2}$, $\mathbf{p}_{i-1}$, $\mathbf{p}_i$
- In Bézier form $\mathbf{Q}_i(t)$ is defined over $[t_i, t_{i+1}]$ by a 3 (Bézier) control points $\mathbf{r}_0$, $\mathbf{r}_1$, $\mathbf{r}_2$
- $\mathbf{r}_0$ and $\mathbf{r}_2$ are the endpoints of this segment $\mathbf{Q}_i(t_i)$ and $\mathbf{Q}_i(t_{i+1})$
- $\mathbf{r}_1$ coincides with the B-spline control point $\mathbf{p}_{i-1}$
- Using the notation of the de Boor algorithm:

$$\mathbf{r}_0 = \mathbf{Q}_i(t_i) = \frac{t_{i+1} - t_i}{t_{i+1} - t_{i-1}}\mathbf{p}_{i-2} + \frac{t_i - t_{i-1}}{t_{i+1} - t_{i-1}}\mathbf{p}_{i-1} = \mathbf{q}_{i-1}^1(t_i)$$

$$\mathbf{r}_1 = \mathbf{p}_{i-1}$$

$$\mathbf{r}_2 = \mathbf{Q}_i(t_{i+1}) = \frac{t_{i+2} - t_{i+1}}{t_{i+2} - t_i}\mathbf{p}_{i-1} + \frac{t_{i+1} - t_i}{t_{i+2} - t_i}\mathbf{p}_i = \mathbf{q}_i^1(t_{i+1})$$

VII. Bézier Form of Cubic B-splines

- Consider :
  - A cubic (k=3) B-spline $\mathbf{Q}(t)$
  - A specific segment $\mathbf{Q}_i(t)$ defined from (B-spline) control points $\mathbf{p}_{i\text{-}3,}$ $\mathbf{p}_{i\text{-}2}$, $\mathbf{p}_{i\text{-}1}$, $\mathbf{p}_i$ over $[t_i, t_{i+1}]$
- In Bézier form $\mathbf{Q}_i(t)$ is defined by a 4 (Bézier) control points $\mathbf{r}_0$, $\mathbf{r}_1$, $\mathbf{r}_{2,}$ $\mathbf{r}_3$
- $\mathbf{r}_0$ and $\mathbf{r}_3$ are the endpoints of this segment $\mathbf{Q}_i(t_i)$ and $\mathbf{Q}_i(t_{i+1})$
- The middle control points are some of the intermediate points generated during the interpolation steps.
- Using the notation of the de Boor algorithm:

$$\mathbf{r}_0 = \frac{t_{i+1} - t_i}{t_{i+1} - t_{i-1}}\left(\frac{t_{i+1} - t_i}{t_{i+1} - t_{i-2}}\mathbf{p}_{i-3} + \frac{t_i - t_{i-2}}{t_{i+1} - t_{i-2}}\mathbf{p}_{i-2}\right) + \frac{t_i - t_{i-1}}{t_{i+1} - t_{i-1}}\mathbf{r}_1$$

$$\mathbf{r}_1 = \mathbf{q}^1_{i-1}(t_i) = \frac{t_{i+2} - t_i}{t_{i+2} - t_{i-1}}\mathbf{p}_{i-2} + \frac{t_i - t_{i-1}}{t_{i+2} - t_{i-1}}\mathbf{p}_{i-1}$$

$$\mathbf{r}_2 = \mathbf{q}^1_{i-1}(t_{i+1}) = \frac{t_{i+2} - t_{i+1}}{t_{i+2} - t_{i-1}}\mathbf{p}_{i-2} + \frac{t_{i+1} - t_{i-1}}{t_{i+2} - t_{i-1}}\mathbf{p}_{i-1}$$

$$\mathbf{r}_3 = \frac{t_{i+2} - t_{i+1}}{t_{i+2} - t_i}\mathbf{r}_2 + \frac{t_{i+1} - t_i}{t_{i+2} - t_i}\left(\frac{t_{i+3} - t_{i+1}}{t_{i+3} - t_i}\mathbf{p}_{i-1} + \frac{t_{i+1} - t_i}{t_{i+3} - t_i}\mathbf{p}_i\right)$$

Rational Bézier & B-Spline Curves

- Bézier and B-Spline curves are the basic parametric curves used
- There are 2 disadvantages:
  - Not invariant to projection transformations → not handled easily in a 3D graphics scene
  - Cannot represent conic sections (circles, ellipses, parabolas, hyperbolas) except for parabolas
- Problems are overcome by *rational Bézier & B-Spline curves*
- Rational curves are polynomial parametric curves that use homogeneous coordinates
- Given a polynomial curve $\mathbf{X}(t)=[x(t), y(t), z(t)]^T$ ($z(t)=0$ for planar curves), we construct: $\mathbf{X}^h(t)=[w(t)\,x(t), w(t)y(t), w(t)z(t), w(t)]^T$
- If $w(t)$ is constant for each t, we get the original $\mathbf{X}(t)$
- Definition of rational Bézier & B-spline curves is straightforward since the coordinates are independent

Rational Bézier Curves

- Consider a sequence of homogeneous control points:

$$\mathbf{p}_i^h = [w_i \mathbf{p}_i, w_i]^T = [w_i x_i, w_i y_i, w_i z_i, w_i]^T$$

i=0,…,n. A homogeneous Bézier curve can be defined as:

$$\mathbf{P}^h(t) = \sum_{i=0}^{n} B_i^n(t)\mathbf{p}_i^h = \begin{bmatrix} \sum_{i=0}^{n} B_i^n(t) w_i \mathbf{p}_i \\ \sum_{i=0}^{n} B_i^n(t) w_i \end{bmatrix}$$

- By dividing all coordinates by the homogeneous one, we get the usual *Cartesian* form:

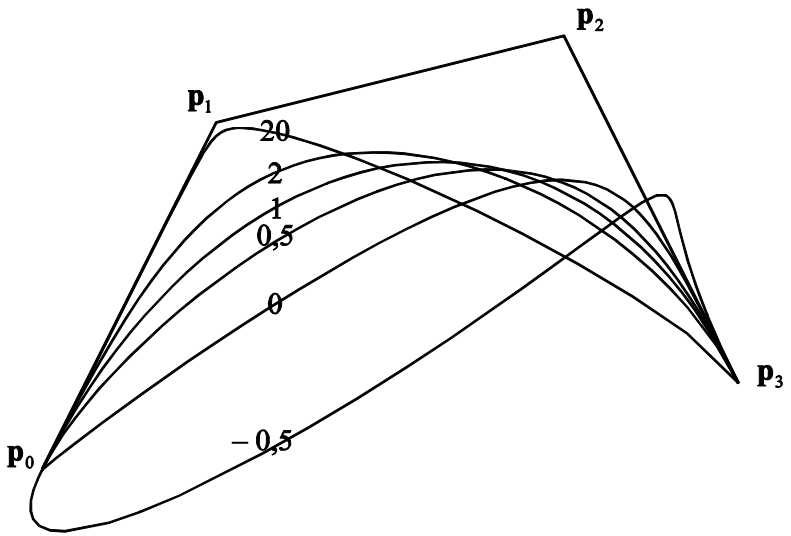
$$\mathbf{P}^r(t) = \frac{1}{\sum_{i=0}^{n} B_i^n(t) w_i} \sum_{i=0}^{n} B_i^n(t) w_i \mathbf{p}_i$$

Rational Bézier Curves (2)

- The above is a *nth-degree rational Bézier curve* , $\mathbf{p}_i$ are the Cartesian control points and w_i are the respective weights
- w_i affects the contribution of the corresponding $\mathbf{p}_i$ to the curve
- As w_i increases the curve is pulled towards $\mathbf{p}_i$
- Ratio of various w_i is of interest and not their absolute values
  - Common factors are ruled out due to the division
- If all weights are equal we have a normal Bézier curve
- Weights are positive → negative weights have unpredictable and undesirable effect on the shape of the curve:
  - Convex-hull property is not maintained
  - Poles, that make the curve go to infinity, may be created
- Weights offer an additional level of local control on the shape of the curve

Rational Bézier Curves (3)

- Effects of different weights to the shape of rational Bézier curve:



- A rational Bézier curve $\mathbf{P}^r(t)$ is the perspective projection of the homogeneous $\mathbf{P}^h(t)$, on the plane $w = 1$
- Rational Bézier curves retain most of the properties of normal Bézier curves:
  - Convex – hull
  - Variation – diminishing

  } hold if the weights are non - negative

Rational Bézier Curves (4)

- Rational Bézier curves are invariant under affine/projective transformations (since they are themselves projection of the homogeneous curves) → to project a rational Bézier curve it suffices to project its control points
- Rational Bézier curves can be evaluated using *de Casteljau* algorithm for the homogeneous control points $\mathbf{p}_i^h$ and performing the homogeneous division at the end
- Alternative formulation: perform division at each step:
  - Numerical accuracy
  - Increased computational complexity

Conic Sections as Rational Bézier Curves

- Conic sections are the various curves resulting from the intersection of a plane at a cone at different angles
- Their algebraic equation have the common form:

$$Ax^2+Bxy+Cy^2+Dx+Ey+F = 0$$

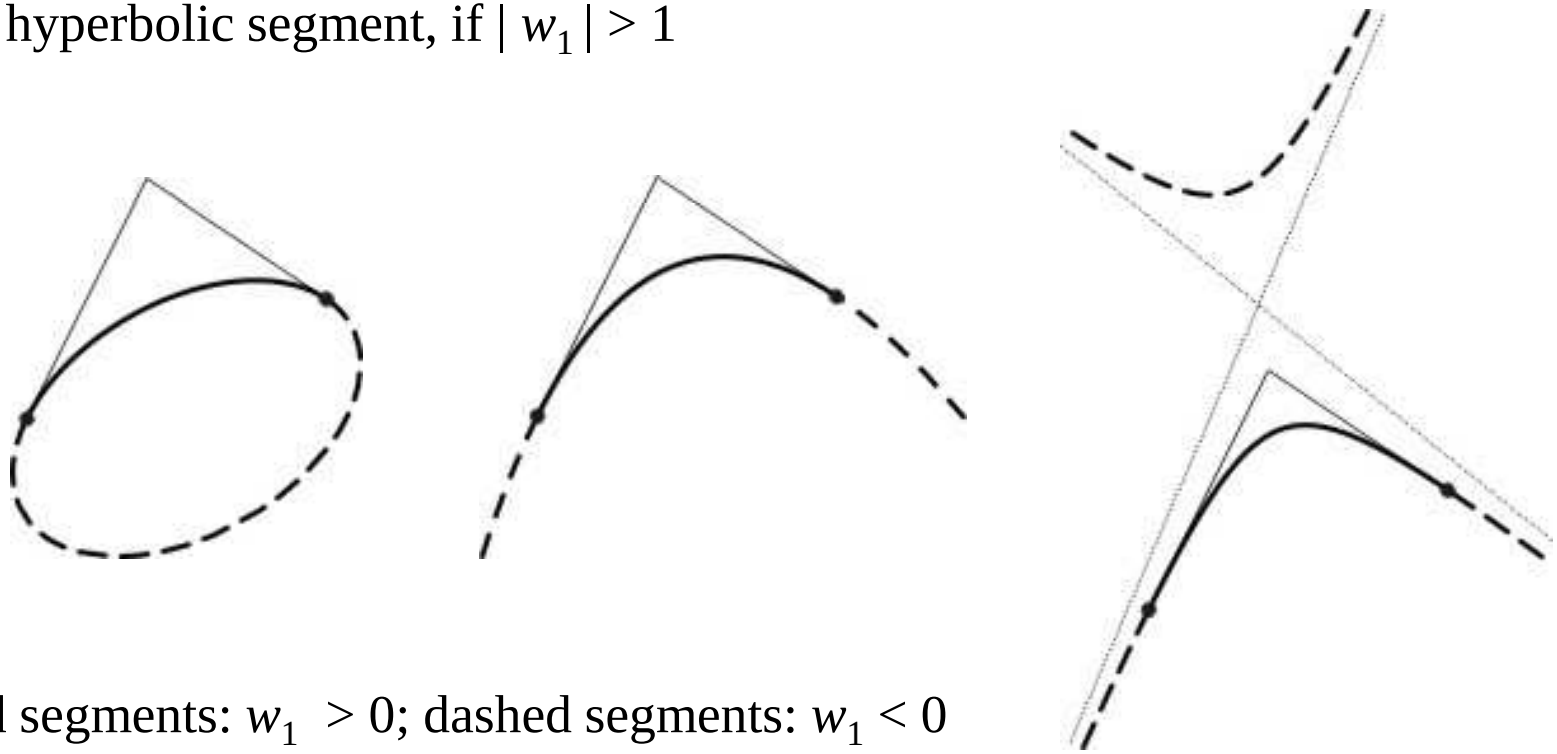
with the following constraints for each kind of curve:

  - If $B^2 - 4AC < 0$, the curve is an ellipse
    - If in addition, $B = 0$ and $A = C$, the curve is a circle
  - If $B^2 - 4AC = 0$, the curve is a parabola
  - If $B^2 - 4AC > 0$, the curve is a hyperbola

- Parabola is the only conic section that can be represented by a non-rational polynomial parametric equation
- Other conic sections can only be represented by rational curves (rational quadratic Bézier curves)

Conic Sections as Rational Bézier Curves (2)

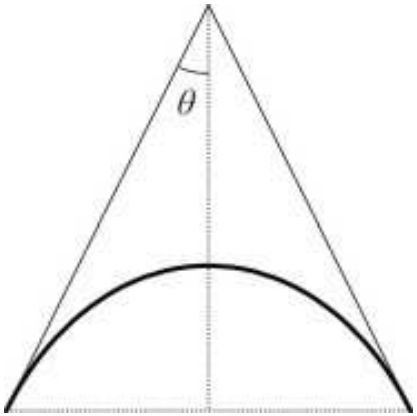
- A rational quadratic Bézier curve with non-collinear control points $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$ and corresponding weights 1, w_1, 1 is:
  - An elliptical segment, if $| w_1 | < 1$
  - A parabolic segment, if $| w_1 | = 1$ (normal Bézier curve)
  - A hyperbolic segment, if $| w_1 | > 1$



Solid segments: $w_1 > 0$; dashed segments: $w_1 < 0$

Conic Sections as Rational Bézier Curves (3)

- Circular arcs deserve special attention:
  - The control polygon must be an isosceles triangle $|\mathbf{p}_0\mathbf{p}_1| = |\mathbf{p}_1\mathbf{p}_2|$
  - $w_1 = \sin\theta$, where θ is the half-angle between $\mathbf{p}_0\mathbf{p}_1$ and $\mathbf{p}_1\mathbf{p}_2$



Rational B-Spline Curves - NURBS

- Given a sequence of control points $\mathbf{p}_i$, i=0,…,n, a sequence of corresponding weights w_i, i=0,…,n and a knot sequence t_i, i=0,…,n+k, a *rational B-Spline curve* of degree k is given by:

$$\mathbf{Q}^r(t) = \frac{1}{\sum_{i=0}^{n} N_i^n(t) w_i} \sum_{i=0}^{n} N_i^n(t) w_i \mathbf{p}_i$$

- Rational B-Spline curves with arbitrary knot sequence are referred to as NURBS (non-uniform rational B-Splines)
- NURBS retain most of the properties of B-Spline curves:
  - Strong convex – hull
  - Strong variation – diminishing

  } hold if the weights are non - negative
- NURBS are invariant under affine/projective transformations

Rational B-Spline Curves – NURBS (2)

- Weights have the same properties as rational Bézier curves → offer flexibility to the designer
- NURBS are the most general of all curve representations up to this point
- They can represent:
  - Simple B-Spline curves (equal weights)
  - Simple and rational Bézier curves
  - Conic sections
- NURBS possess all properties of the other types of curves
- NURBS are the standard tool in CAGD applications

Interpolation Curves

- Bézier and B–Spline curves are *approximation curves*:
  - They do not pass through their control points
  - The control points only provide a good indication of their shape
- Need to construct *interpolation curves*:
  - Curves that pass through given points
- Interpolation problem: Given a set of points $\mathbf{p}_0$, $\mathbf{p}_1$, …, $\mathbf{p}_n$ and corresponding parametric values t_0, t_1, …, t_n, find a parametric curve $\mathbf{P}(t)$ that satisfies:

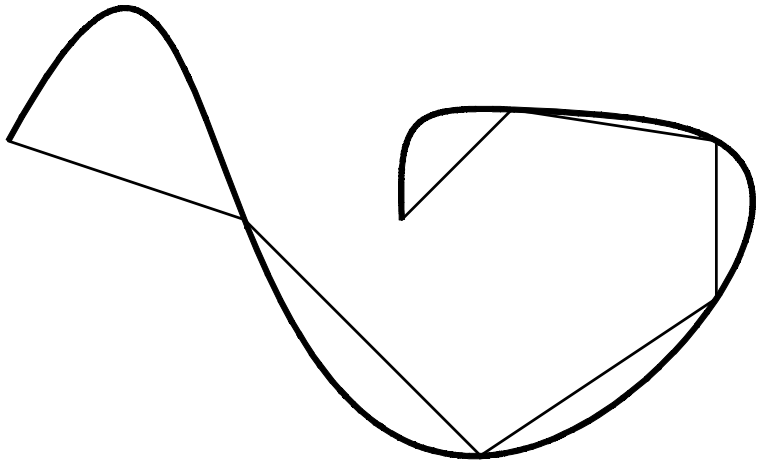
$$\mathbf{P}(t_i) = \mathbf{p}_i,\ i = 0, 1, \ldots, n$$

Interpolation Curves (2)

- Simple interpolation methods construct $\mathbf{P}(t)$ as a single polynomial curve of degree n
- $\mathbf{P}(t)$ is a unique curve
  - Defined by the $(n+1)$ coefficients of the respective polynomials, which can be computed as the single solution of the linear system of $(n+1)$ equations formed as shown above
- Solving a linear system in order to determine the interpolation curve is not practical. Other methods to find the curve:
  - Directly, by using Lagrange polynomials
  - Recursively, by using Aitken's algorithm
- Despite the virtues of these methods there are still drawbacks
  - Interpolation of several points requires a high-degree polynomial → complex and numerically unstable computations
  - The generated curve exhibits *oscillations* and does not follow its control polygon in a predictable way

Interpolation Curves (3)

- Oscillation example:



- To overcome these drawbacks:
  interpolation is performed using curves comprised of several low-degree segments, joined together with continuity constraints
  - Interpolation with B-splines
  - Hermite curves

Simple Polynomial Interpolation

- A way to construct an interpolation curve is using the *n*th-degree *Lagrange polynomials*:

$$L_i^n(t) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{t - t_j}{t_i - t_j}, \qquad i = 0,1,...,n$$

- Then, the interpolation curve is:

$$\mathbf{P}(t) = \sum_{i=0}^{n} L_i^n(t)\mathbf{p_i}(t)$$

- The *i*-th polynomial $L_i^n(t)$ is 0 on every knot t_j except for the *i*-th knot t_i which it is 1 → $\mathbf{P}(t)$ is an interpolation curve
- Further characteristics:
  - *Invariance under affine transformations*: holds since Lagrange polynomials sum to 1 → interpolation curve is a barycentric combination of its control points

Simple Polynomial Interpolation (2)

- *No convex – hull property*: Lagrange polynomials are neither always positive nor less than 1 → the curve is not contained in the convex hull of its control points
- *Linear precision*: if all control points lie on a straight line, the curve has the shape of a straight line
- *No variation–diminishing property*: curve may demonstrate oscillations

- *Aitkens' algorithm* provides a recursive evaluation of the interpolation curve:
  1. For the required value t set: $\mathbf{p}_i^0(t) = \mathbf{p}_i, \qquad i = 0,1,...,n$
  2. Perform the linear interpolation steps:

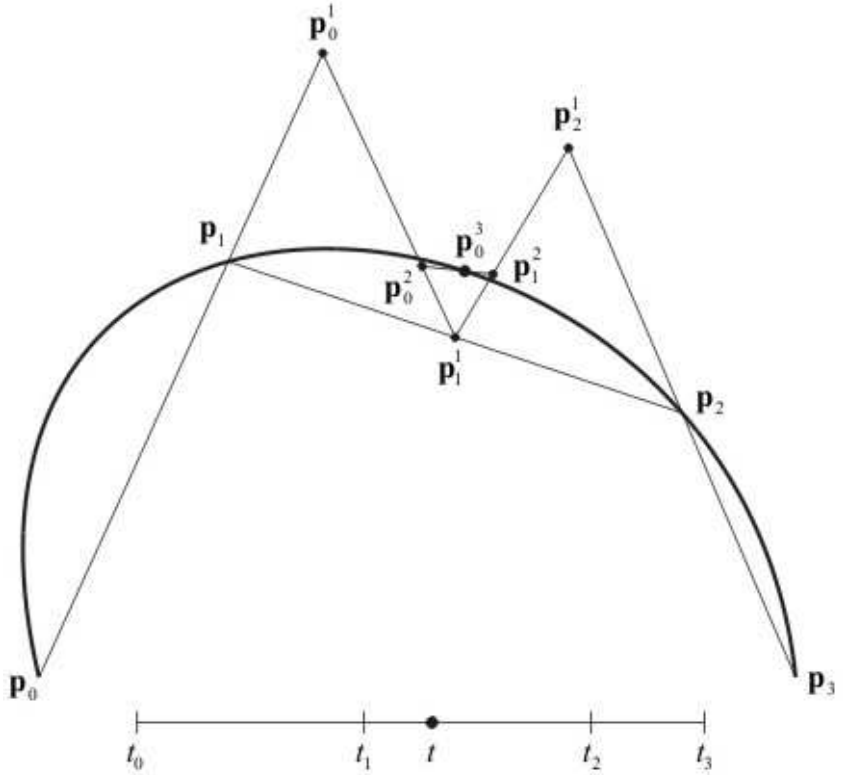
$$\mathbf{p}_i^r(t) = \frac{t_{i+r} - t}{t_{i+r} - t_i}\mathbf{p}_i^{r-1}(t) + \frac{t - t_i}{t_{i+r} - t_i}\mathbf{p}_{i+1}^{r-1}(t), \qquad \begin{matrix} r = 1,2,...,n, \\ i = 0,1,...,n-r \end{matrix}$$

  3. Then, the point on the curve corresponding to parametric value t is:

$$\mathbf{P}(t) = \mathbf{p}_0^n(t)$$

Simple Polynomial Interpolation (3)

- Aitkens' algorithm:



- In the interpolation steps parameter t is not always in $[t_i,t_{i+r}]$
  → the intermediate points generated are not convex combinations of the points in the previous step

Cubic Hermite Interpolation

- To solve the interpolation problem we need to find a curve that passes through given points
- However we may seek a curve that interpolates other elements such as tangents
- *Cubic Hermite curves* are required to interpolate given points and to have given tangents at these points
- Problem: Suppose that we are given 2 points $\mathbf{p}_0$, $\mathbf{p}_1$ and corresponding tangent vectors $\overrightarrow{m_0}, \overrightarrow{m_1}$. We are seeking the cubic Hermite curve $\mathbf{H}(t)$, $t \in [0, 1]$ that satisfies:

$$\mathbf{H}(0) = \mathbf{p}_0, \qquad \mathbf{H}'(0) = \overrightarrow{\mathbf{m}_0},$$

$$\mathbf{H}(1) = \mathbf{p}_1, \qquad \mathbf{H}'(1) = \overrightarrow{\mathbf{m}_1}$$

- We express the Hermite curve in the form of a Bézier curve as:

$$\mathbf{H}(t) = \sum_{i=0}^{n} B_i^n(t)\mathbf{q}_i, \qquad t \in [0,1]$$

for some unknown control points $\mathbf{q}_i$

Cubic Hermite Interpolation (2)

- Using properties of the Bézier curves:

$$\mathbf{p}_0 = \mathbf{H}(0) = \mathbf{q}_0, \qquad \overrightarrow{\mathbf{m}_0} = \mathbf{H}'(0) = 3(\mathbf{q}_1 - \mathbf{q}_0) \Leftrightarrow \mathbf{q}_1 = \mathbf{p}_0 + \frac{1}{3}\overrightarrow{\mathbf{m}_0},$$

$$\mathbf{p}_1 = \mathbf{H}(1) = \mathbf{q}_3, \qquad \overrightarrow{\mathbf{m}_1} = \mathbf{H}'(1) = 3(\mathbf{q}_3 - \mathbf{q}_2) \Leftrightarrow \mathbf{q}_2 = \mathbf{p}_1 - \frac{1}{3}\overrightarrow{\mathbf{m}_1}$$

- Therefore the curve is:

$$\mathbf{H}(t) = (1-t)^3\mathbf{p}_0 + 3t(1-t)^2(\mathbf{p}_0 + \frac{1}{3}\overrightarrow{\mathbf{m}_0}) + 3t^2(1-t)(\mathbf{p}_1 - \frac{1}{3}\overrightarrow{\mathbf{m}_1}) + t^3\mathbf{p}_1$$

- Or expressing it with respect to its defining elements:

$$\mathbf{H}(t) = H_0^3(t)\mathbf{p}_0 + H_1^3(t)\mathbf{p}_1 + H_2^3(t)\overrightarrow{\mathbf{m}_0} + H_3^3(t)\overrightarrow{\mathbf{m}_1}, \quad t \in [0,1]$$

where $H_i^3(t)$ are the cubic Hermite polynomials:

$$H_0^3(t) = 2t^3 - 3t^2 + 1, \quad H_1^3(t) = -2t^3 + 3t^2,$$
$$H_2^3(t) = t^3 - 2t^2 + t, \quad H_0^3(t) = t^3 - t^2$$

- In case the curve is defined over an arbitrary parametric interval [*a*, *b*], these relations must be modified

Cubic Hermite Interpolation (3)

- Hermite curves are *not invariant* to affine transformations of their parameter → their defining elements must be altered for the curve to remain the same, when parameter $t \in [0,1]$ is changed to $u \in [a, b]$:
  - We set $u = (1-t)a + tb$
- The tangents at the endpoints are now:

$$\overrightarrow{\mathbf{m}_0} = \mathbf{H}'(a) = \frac{1}{b-a} 3(\mathbf{q}_1 - \mathbf{q}_0),$$

$$\overrightarrow{\mathbf{m}_1} = \mathbf{H}'(b) = \frac{1}{b-a} 3(\mathbf{q}_3 - \mathbf{q}_2)$$

- Working as above we deduce that:

$$\mathbf{H}(u) = H_0^3(u)\mathbf{p}_0 + H_1^3(u)\mathbf{p}_1 + H_2^3(u)(b-a)\overrightarrow{\mathbf{m}_0} + H_3^3(u)(b-a)\overrightarrow{\mathbf{m}_1}, \quad u \in [a,b]$$

- In order to obtain the curve in the desired form we need to divide by $(b\text{-}a)$
  - Physical explanation: Using $\mathbf{H}(t)$ we traverse the curve in one time unit; if we want to traverse it in $(b\text{-}a)$ time units, our speed, represented by the tangent vectors, must be smaller by a factor of $(b\text{-}a)$

Cubic Hermite Interpolation (4)

Piecewise cubic Hermite interpolation:

- Interpolating only 2 points and respecting tangent vectors has no practical interest
- More interesting is to construct a smooth curve that interpolates a sequence of points $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_n$ and respective tangents vectors $\mathbf{m}_0, \mathbf{m}_1, \ldots, \mathbf{m}_n$ at parametric values $t_0, t_1, \ldots, t_n$
- Possible to construct this curve as a piecewise cubic Hermite curve
- Independent Hermite segments, one for each parametric interval $[t_i, t_{i+1}]$, may be constructed
- The segments will constitute a C^1-continuous curve since they share tangent vectors $\mathbf{m}_i$ at their endpoints
- Each segment will be given:

$$\mathbf{H}_i(u) = H_0^3(u)\mathbf{p}_i + H_1^3(u)\mathbf{p}_{i+1} + H_2^3(u)(t_{i+1} - t_i)\mathbf{m}_i + H_3^3(u)(t_{i+1} - t_i)\mathbf{m}_{i+1},$$

$$u \in [t_i, t_{i+1}]$$

Cubic Hermite Interpolation (5)

- This construction of an interpolating curve provides flexibility:
  - It allows the modification of the shape of the curve by altering the tangent vectors
  - Even more flexibility achieved, by requiring only G^1 geometric continuity at the joins, allowing the tangent vectors at end of a segment and at the beginning of the next one to be multiple of each other instead of equal

Cubic Hermite Interpolation (6)

Automatic generation of tangents:

- In some situations, specifying the tangent vectors explicitly may not be easy or desirable
- An automatic method for computing tangent vectors is needed
- Simplest methods seek a curve C^1 continuous at the joins
- More complicated methods produce C^2 continuous curve

Cubic Hermite Interpolation (7)

An approach to compute tangent vectors is to set $\overrightarrow{\mathbf{m}}_i$ parallel to the line through the 2 neighboring control points $\mathbf{p}_{i-1}$, $\mathbf{p}_{i+1}$:

$$\overrightarrow{\mathbf{m}}_i = \frac{1}{2}(1-c)(\mathbf{p}_{i+1} - \mathbf{p}_{i-1}), \qquad i = 1, 2, ..., n-1$$

where c is a *tension parameter* that affects the norm of the tangent vectors

- The curves generated using these tangent are called *cardinal splines*
- If $c = 0$ then $\overrightarrow{\mathbf{m}}_i = \frac{1}{2}(\mathbf{p}_{i+1} - \mathbf{p}_{i-1})$ and the curves are called *Catmull-Rom splines*
- This procedure cannot determine tangents $\overrightarrow{\mathbf{m}}_0, \overrightarrow{\mathbf{m}}_n$ at the first and last control points

Cubic Hermite Interpolation (8)

A second approach is to use *Bessel tangents*:

- Tangent vector $\overrightarrow{\mathbf{m}_i}$ is set equal to the tangent of the parabola that interpolates the 3 neighboring points $\mathbf{p}_{i-1}$, $\mathbf{p}_i$, $\mathbf{p}_{i+1}$
- If $\mathbf{Q}_i(u)$, $u \in [t_i, t_{i+1}]$ is this parabola (can be computed using Lagrange polynomials or Aitken's algorithm) then:

$$\overrightarrow{\mathbf{m}_i} = \frac{d}{du}\mathbf{Q}_i(t_i), \qquad i = 1, 2, \ldots, n-1$$

- For the first and last tangent vectors we may use the tangents of the first and last parabola respectively:

$$\overrightarrow{\mathbf{m}_0} = \frac{d}{du}\mathbf{Q}_1(t_0) \quad \text{and} \quad \overrightarrow{\mathbf{m}_n} = \frac{d}{du}\mathbf{Q}_{n-1}(t_n)$$

Cubic Hermite Interpolation (9)

- We reach the following formulas for the tangent vectors in terms of the elements of the curve:

$$\overrightarrow{\mathbf{m}_0} = \frac{-t_2 - t_1 + 2t_0}{(t_2 - t_0)(t_1 - t_0)}\mathbf{p}_0 + \frac{t_2 - t_0}{(t_2 - t_1)(t_1 - t_0)}\mathbf{p}_1 + \frac{t_1 - t_0}{(t_2 - t_1)(t_2 - t_0)}\mathbf{p}_2$$

$$\overrightarrow{\mathbf{m}_i} = \frac{t_{i+1} - t_i}{(t_{i+1} - t_{i-1})(t_i - t_{i-1})}\mathbf{p}_{i-1} + \frac{t_{i+1} - 2t_i + t_{i-1}}{(t_{i+1} - t_i)(t_i - t_{i-1})}\mathbf{p}_i + \frac{t_i - t_{i-1}}{(t_{i+1} - t_i)(t_{i+1} - t_{i-1})}\mathbf{p}_{i+1}$$

$$\overrightarrow{\mathbf{m}_n} = \frac{t_n - t_{n-1}}{(t_n - t_{n-2})(t_{n-1} - t_{n-2})}\mathbf{p}_{n-2} - \frac{t_n - t_{n-2}}{(t_n - t_{n-1})(t_{n-1} - t_{n-2})}\mathbf{p}_{n-1} + \frac{2t_n - t_{n-1} - t_{n-2}}{(t_n - t_{n-1})(t_n - t_{n-2})}\mathbf{p}_n$$

- Notice that Bessel tangents $\overrightarrow{\mathbf{m}_0}, \overrightarrow{\mathbf{m}_n}$ can be used independently, in order complement the tangents of cardinal splines mentioned above

Cubic Hermite Interpolation (10)

- The 2 previous methods generate C^1 continuous curves
- To create a C^2 continuous curve, we must require that the second derivatives of each pair of successive segments are equal at the joins
- If $\mathbf{H}_i(u)$, $u \in [t_i, t_{i+1}]$ is the segment that interpolates $\mathbf{p}_i$, $\mathbf{p}_{i+1}$ the following must hold:

$$\frac{d^2}{du^2}\mathbf{H}_{i-1}(t_i) = \frac{d^2}{du^2}\mathbf{H}_i(t_i)$$

- Using :

$$\mathbf{H}(u) = H_0^3(u)\mathbf{p}_0 + H_1^3(u)\mathbf{p}_1 + H_2^3(u)(b-a)\overrightarrow{\mathbf{m}_0} + H_3^3(u)(b-a)\overrightarrow{\mathbf{m}_1}, \quad u \in [a,b]$$

we differentiate the Hermite curve segments twice and get:

$$(t_{i+1}-t_i)\overrightarrow{\mathbf{m}_{i-1}} + 2(t_{i+1}-t_{i-1})\overrightarrow{\mathbf{m}_i} + (t_i-t_{i-1})\overrightarrow{\mathbf{m}_{i+1}} = 3\frac{t_{i+1}-t_i}{t_i-t_{i-1}}(\mathbf{p}_i-\mathbf{p}_{i-1}) + 3\frac{t_i-t_{i-1}}{t_{i+1}-t_i}(\mathbf{p}_{i+1}-\mathbf{p}_i)$$

- This relation holds for i=1,2,…,n-1 providing (n-1) equations for the computation of the (n+1) tangent vectors $\overrightarrow{\mathbf{m}_i}$ i=0,1,…,n

Cubic Hermite Interpolation (11)

- 2 additional conditions must be used for the interpolation curve
- Customary to apply conditions referring to the ends of the curve, from which the values of $\overrightarrow{\mathbf{m}_0}, \overrightarrow{\mathbf{m}_n}$ are computed
- Easiest approach:
  - Allow the user to supply arbitrary values for the 2 tangent vectors
- Alternatively:
  - Geometric conditions, taking into account the shape of the curve near its ends, are applied
- Let us suppose that $\overrightarrow{\mathbf{m}_0}, \overrightarrow{\mathbf{m}_n}$ are known
- Using the previous equation, the linear system to be solved is:

$$\begin{bmatrix} 1 & 0 & \dots & 0 & 0 \\ \alpha_1 & \beta_1 & \gamma_1 & & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & & \alpha_{n-1} & \beta_{n-1} & \gamma_{n-1} \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix} \begin{bmatrix} \overrightarrow{\mathbf{m}_0} \\ \overrightarrow{\mathbf{m}1} \\ \vdots \\ \overrightarrow{\mathbf{m}_{n-1}} \\ \overrightarrow{\mathbf{m}_n} \end{bmatrix} = \begin{bmatrix} \overrightarrow{\mathbf{c}_0} \\ \overrightarrow{\mathbf{c}_1} \\ \vdots \\ \overrightarrow{\mathbf{c}_{n-1}} \\ \overrightarrow{\mathbf{c}_n} \end{bmatrix}$$

Cubic Hermite Interpolation (12)

where:

$$\alpha_i = (t_{i+1} - t_i), \qquad \mathbf{c}_0 = \overrightarrow{\mathbf{m}}_0$$

$$\beta_i = 2(t_{i+1} - t_{i-1}), \quad \mathbf{c}_i = 3\frac{t_{i+1} - t_i}{t_i - t_{i-1}}(\mathbf{p}_i - \mathbf{p}_{i-1}) + 3\frac{t_i - t_{i-1}}{t_{i+1} - t_i}(\mathbf{p}_{i+1} - \mathbf{p}_i)$$

$$\gamma_i = (t_i - t_{i-1}), \qquad \mathbf{c}_n = \overrightarrow{\mathbf{m}}_n$$

- Solving this system we get tangent vectors $\overrightarrow{\mathbf{m}}_i$ so that interpolating curve is C^1 continuous
- The system always has a unique solution
- It is a tridiagonal system → may be solved efficiently using a direct method such as LU decomposition

Cubic Hermite Interpolation (13)

End conditions for C^2 piecewise Hermite interpolation:

- The additional conditions necessary to determine the tangents for C^2 piecewise Hermite curve involve tangent vectors $\mathbf{m}_0, \mathbf{m}_n$

  → called *end conditions*

- *Bessel end condition:*
  - Bessel tangents, computed above for $\overrightarrow{\mathbf{m}_0}, \overrightarrow{\mathbf{m}_n}$, are used so that the tangents at the ends are those of the parabolas that interpolate the first and last 3 control points
  - It suffices to replace $\overrightarrow{\mathbf{c}_0}, \overrightarrow{\mathbf{c}_n}$ in the previous linear system with these expressions

Cubic Hermite Interpolation (14)

- *Quadratic end condition*:
  - Requires that the 2nd derivatives of the interpolation curve at the first (and last) 2 knots are equal:

$$\frac{d^2}{du^2}\mathbf{H}_0(t_0) = \frac{d^2}{du^2}\mathbf{H}_0(t_1) \quad \text{and} \quad \frac{d^2}{du^2}\mathbf{H}_{n-1}(t_{n-1}) = \frac{d^2}{du^2}\mathbf{H}_{n-1}(t_n)$$

  - By differentiating the Hermite curve twice, we can deduce that under this assumption:

$$\overrightarrow{\mathbf{m}_0} + \overrightarrow{\mathbf{m}_1} = 2\frac{\mathbf{p}_1 - \mathbf{p}_0}{t_1 - t_0} \quad \text{and} \quad \overrightarrow{\mathbf{m}_{n-1}} + \overrightarrow{\mathbf{m}_n} = 2\frac{\mathbf{p}_n - \mathbf{p}_{n-1}}{t_n - t_{n-1}}$$

  - These relations must be plugged into the linear system, replacing the first and last lines in full
  - The system remains tridiagonal after the changes → efficiently solved

Cubic Hermite Interpolation (15)

- *Physical end condition*:
  - Requires that the 2nd derivatives are equal to 0 at the ends of the curve
  - Then, the interpolation curve becomes a straight line near its ends (desirable or not, depending on application)
  - The generated curve resembles the mechanical (physical) spline, which is pinned at its ends so that its curvature vanishes
  - Working similarly to the quadratic end condition we get:

$$2\overrightarrow{\mathbf{m}_0} + \overrightarrow{\mathbf{m}_1} = 3\frac{\mathbf{p}_1 - \mathbf{p}_0}{t_1 - t_0} \quad \text{and} \quad \overrightarrow{\mathbf{m}_{n-1}} + 2\overrightarrow{\mathbf{m}_n} = 3\frac{\mathbf{p}_n - \mathbf{p}_{n-1}}{t_n - t_{n-1}}$$

  - We should replace the first and last equation of the linear system, with these equations

Cubic B-Spline Interpolation

- Here it is shown how to construct a cubic B-Spline curve that interpolates a set of given points
- The interpolating B-Spline curve is denoted as $\mathbf{Q}(t)$
- We require that the given parametric values t_i, at which the curve interpolates points $\mathbf{p}_i$:

$$\mathbf{Q}(t_i) = \mathbf{p}_i \,, \quad i = 0, 1, \ldots, n$$

  are also used as the knots of the B-Spline curve
- Aim: Find the control points $\mathbf{q}_i$ of this B–Spline curve
- Suppose that $\mathbf{p}_i$ are all different from each other → t_i should also be different from each other
- The first and last points are easy to interpolate, if a clamped knot sequence is used
- Therefore, we add knots t_{-2}, t_{-1}, t_{n+1}, t_{n+2} such that:

$$t_{-2} = t_{-1} = t_0 \quad \text{and} \quad t_n = t_{n+1} = t_{n+2}$$

Cubic B-Spline Interpolation (2)

- Given this knot sequence, the control points are $\mathbf{q}_i$, i=-3,…,n-1 (indices are ranged according to the knots indices)
- The first and last control points are already known:

$$\mathbf{q}_{-3} = \mathbf{p}_0 \text{ and } \mathbf{q}_{n-1} = \mathbf{p}_n$$

- For the remaining control points, we use the definition of the cubic B-Spline curve:

$$\mathbf{p}_j = \mathbf{Q}(t_j) = \sum_{i=-3}^{n-1} N_i^3(t_j)\mathbf{q}_i, \qquad j = 1,2,...,n-1 \qquad \textbf{(E)}$$

- The value of the cubic B-Spline basis function $N_i^3(t)$ at the knots t_j can be computed as follows:
  - First, evaluate the quadratic B-Spline basis function at the knots to get:

$$N_i^2(t_j) = \begin{cases} 0, & j = 1, \\ \dfrac{t_{i+1} - t_i}{t_{i+2} - t_i}, & j = i+1, \\ \dfrac{t_{i+3} - t_{i+2}}{t_{i+3} - t_{i+1}}, & j = i+2, \\ 0, & \text{otherwise} \end{cases}$$

Cubic B-Spline Interpolation (3)

- Then we apply:

$$N_i^r(t) = \frac{t - t_i}{t_{i+r} - t_i} N_i^{r-1}(t) + \frac{t_{i+r+1} - t}{t_{i+r+1} - t_{i+1}} N_{i+1}^{r-1}(t)$$

to get:

$$N_i^2(t_j) = \begin{cases} \dfrac{t_{i+1} - t_i}{t_{i+3} - t_i} \dfrac{t_{i+1} - t_i}{t_{i+2} - t_i}, & j = i+1, \\ \dfrac{t_{i+2} - t_i}{t_{i+3} - t_i} \dfrac{t_{i+3} - t_{i+2}}{t_{i+3} - t_{i+1}} + \dfrac{t_{i+4} - t_{i+2}}{t_{i+4} - t_{i+1}} \dfrac{t_{i+2} - t_{i+1}}{t_{i+3} - t_{i+1}}, & j = i+2, \\ \dfrac{t_{i+4} - t_{i+3}}{t_{i+4} - t_{i+1}} \dfrac{t_{i+4} - t_{i+3}}{t_{i+4} - t_{i+2}}, & j = i+3, \\ 0, & \text{otherwise} \end{cases}$$

Cubic B-Spline Interpolation (4)

- Finally, we change the indices in order to get the $N_i^3(t)$ for a constant j and for a suitable i:

$$N_{j-1}^3(t_j) = \frac{t_j - t_{j-1}}{t_{j+2} - t_{j-1}} \frac{t_j - t_{j-1}}{t_{j+1} - t_{j-1}},$$

$$N_{j-2}^3(t_j) = \frac{t_j - t_{j-2}}{t_{j+1} - t_{j-2}} \frac{t_{j+1} - t_j}{t_{j+1} - t_{j-1}} + \frac{t_{j+2} - t_j}{t_{j+2} - t_{j-1}} \frac{t_j - t_{j-1}}{t_{j+1} - t_{j-1}},$$

$$N_{j-3}^3(t_j) = \frac{t_{j+1} - t_j}{t_{j+1} - t_{j-2}} \frac{t_{j+1} - t_j}{t_{j+1} - t_{j-1}},$$

- Therefore, equation **(E)** becomes:

$$\mathbf{p}_j = N_{j-3}^3(t_j)\mathbf{q}_{j-3} + N_{j-2}^3(t_j)\mathbf{q}_{j-2} + N_{j-1}^3(t_j)\mathbf{q}_{j-1}$$

Cubic B-Spline Interpolation (5)

- Substituting $N_i^3(t_j)$ we get:

$$\frac{(t_{j+1}-t_j)^2}{t_{j+1}-t_{j-1}}\mathbf{q}_{j-3}$$

$$+\left[\frac{(t_j-t_{j-2})(t_{j+1}-t_j)}{t_{j+1}-t_{j-2}}+\frac{(t_{j+2}-t_j)(t_j-t_{j-1})}{t_{j+2}-t_{j-1}}\right]\mathbf{q}_{j-2}$$

$$+\frac{(t_j-t_{j-1})^2}{t_{j+2}-t_{j-1}}\mathbf{q}_{j-1}=(t_{j+1}-t_{j-1})\mathbf{p}_j$$

- The above expressions together with the relations: $\mathbf{q}_{-3} = \mathbf{p}_0$ and $\mathbf{q}_{n-1} = \mathbf{p}_n$ provide a linear system with (n+1) unknowns for the determination of the (n+3) control points → 2 more conditions are needed
- Similar to the situation occurred when we required that a Hermite interpolation curve be C^2 at the joins of its segments

Cubic B-Spline Interpolation (6)

- The above is not accidental, since in both cases we seek a piecewise cubic curve that is C^2 continuous and interpolates $(n+1)$ given points
- Difference:
  - In the former case, the curve was expressed as a piecewise Hermite curve
  - In the later case, it is given as a B-Spline curve
- We assume that $\mathbf{q}_{-2}$, $\mathbf{q}_{-1}$ are known
- The linear system can be written:

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & \alpha_1 & \beta_1 & \gamma_1 & 0 & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ 0 & \dots & 0 & \alpha_{n-1} & \beta_{n-1} & \gamma_{n-1} & 0 \\ 0 & 0 & \dots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{q}_{-3} \\ \mathbf{q}_{-2} \\ \mathbf{q}_{-1} \\ \vdots \\ \mathbf{q}_{n-3} \\ \mathbf{q}_{n-2} \\ \mathbf{q}_{n-1} \end{bmatrix} = \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{r}_1 \\ (t_2 - t_0)\mathbf{p}_1 \\ \vdots \\ (t_n - t_{n-2})\mathbf{p}_{n-1} \\ \mathbf{r}_2 \\ \mathbf{p}_n \end{bmatrix}$$

Cubic B-Spline Interpolation (7)

where we set α_j, β_j, γ_j the coefficients of $\mathbf{q}_{j-3}$, $\mathbf{q}_{j-2}$, $\mathbf{q}_{j-1}$ respectively and $\mathbf{r}_1 = \mathbf{q}_{-2}$ and $\mathbf{r}_2 = \mathbf{q}_{n-2}$

- This is a tridiagonal system → can be efficiently solved using a direct method such as LU decomposition

End conditions:

- For B-Spline interpolation, the end conditions required to complete the linear system refer to the 2nd and penultimate control points $\mathbf{q}_{-2}$, $\mathbf{q}_{n-2}$
- Same end condition procedures as for cubic Hermite interpolation
- Difference:
  - Equations will be expressed in terms of the given points $\mathbf{p}_i$ instead of the tangent vectors

Cubic B-Spline Interpolation (8)

- *Bessel end condition:*
  - At the start of the curve, the tangent is

$$\mathbf{Q}'(t_0) = \frac{3}{t_1 - t_0}(\mathbf{q}_{-2} - \mathbf{q}_{-3}) = \frac{3}{t_1 - t_0}(\mathbf{q}_{-2} - \mathbf{p}_0)$$

  - Equating this with:

$$\overrightarrow{\mathbf{m}_0} = \frac{-t_2 - t_1 + 2t_0}{(t_2 - t_0)(t_1 - t_0)}\mathbf{p}_0 + \frac{t_2 - t_0}{(t_2 - t_1)(t_1 - t_0)}\mathbf{p}_1 + \frac{t_1 - t_0}{(t_2 - t_1)(t_2 - t_0)}\mathbf{p}_2$$

    we get:

$$\mathbf{q}_{-2} = \frac{1}{3}\left(\frac{2t_2 - t_1 - t_0}{t_2 - t_0}\mathbf{p}_0 - \frac{t_2 - t_0}{t_2 - t_1}\mathbf{p}_1 - \frac{(t_1 - t_0)^2}{(t_2 - t_1)(t_2 - t_0)}\mathbf{p}_2\right)$$

  - Working similar for the end of the curve we get:

$$\mathbf{q}_{n-2} = \frac{1}{3}\left(-\frac{(t_{n-1} - t_n)^2}{(t_{n-2} - t_{n-1})(t_{n-2} - t_n)}\mathbf{p}_n + \frac{t_{n-2} - t_n}{t_{n-2} - t_{n-1}}\mathbf{p}_{n-1} + \frac{2t_{n-2} - t_{n-1} - t_n}{t_{n-2} - t_n}\mathbf{p}_n\right)$$

  - These expressions for $\mathbf{q}_{-2}$, $\mathbf{q}_{n-2}$ must replace $\mathbf{r}_1$ and $\mathbf{r}_2$ in the linear system

Cubic B-Spline Interpolation (9)

- *Quadratic end condition*:
  - We may write the first (and similarly the last) segment of B-Spline curve, defined over a parametric interval $[t_0, t_1]$, as a cubic Bézier curve and differentiate this form twice:

$$\mathbf{q}_{-2} - \mathbf{q}_{-1} = \frac{t_2 - t_0}{3(t_1 - t_0)}(\mathbf{p}_0 - \mathbf{p}_1)$$

$$\mathbf{q}_{n-3} - \mathbf{q}_{n-2} = \frac{t_n - t_{n-2}}{3(t_n - t_{n-1})}(\mathbf{p}_{n-1} - \mathbf{p}_n)$$

  - The above must replace the 2nd and the penultimate equation of the system, in full
  - The system remains tridiagonal after the changes
- *Physical end conditions:*
  - Applied similarly and the respective equations are:

$$(t_2 + t_1 - 2t_0)\mathbf{q}_{-2} - (t_1 - t_0)\mathbf{q}_{-1} = (t_2 - t_0)\mathbf{p}_0$$

$$(t_n - t_{n-1})\mathbf{q}_{n-3} - (2t_n - t_{n-1} - t_{n-2})\mathbf{q}_{n-2} = -(t_n - t_{n-2})\mathbf{p}_n$$

Parameterizations of Piecewise Interpolation Curves

- The user is interested only in providing the points that the curve interpolates
- In such cases, required knots may be computed algorithmically
  - We can use the given points to generate better shaped curves
- Parameterization methods presented for B-Spline curves, can be applied here as well
- *Uniform parameterization* is the simplest
  - In *Uniform* the knots are equidistant
- *Uniform* is used in practice, although other methods may produce better curves:
  - Simplifies the linear systems that must be solved to produce C^2 cubic interpolating curves
- More complex parameterizations (to achieve smoother curves), can be constructed by using the given interpolated points $\mathbf{p}_i$

Parameterizations of Piecewise Interpolation Curves (2)

- A chord-length parameterization can be computed from:

$$\frac{t_{i+2} - t_{i+1}}{t_{i+1} - t_i} = \frac{\left|\mathbf{p}_{i+2} - \mathbf{p}_{i+1}\right|}{\left|\mathbf{p}_{i+1} - \mathbf{p}_i\right|}$$

- A centripetal parameterization can be computed from:

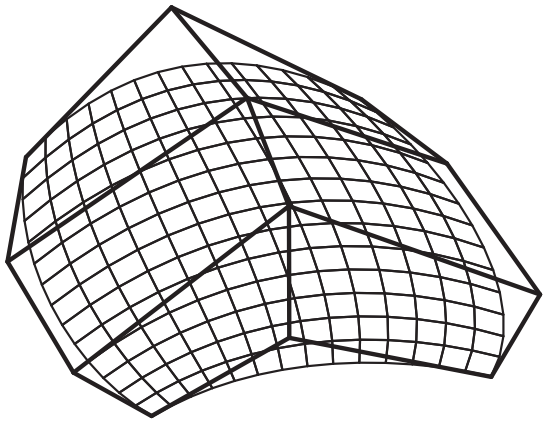
$$\frac{t_{i+2} - t_{i+1}}{t_{i+1} - t_i} = \sqrt{\frac{\left|\mathbf{p}_{i+2} - \mathbf{p}_{i+1}\right|}{\left|\mathbf{p}_{i+1} - \mathbf{p}_i\right|}}$$

- In both above cases, the value of initial knot t_0 can be specified arbitrarily
- Notice:
  - B-Spline parameterizations use the control points of the curve

  But:
  - Parameterizations of interpolating curves use the points being interpolated → the shape of the curve should be adopted to them

Surfaces

- Bézier and B-spline curves can be used to generate parametric surfaces
- Most straightforward types:
  - Tensor product Bézier surfaces



  - Tensor product B-spline surfaces
- Are simple generalizations of the respective curves
- Inherit most of their properties

Tensor Product Bézier Surfaces

- Consider a Bézier curve of degree m with control points $\mathbf{p}_i$, $i=0,1,\ldots,m$ given in terms of parameter u:

$$\mathbf{P}^m(u) = \sum_{i=0}^{m} B_i^m(u)\mathbf{p}_i, \quad u \in [0,1] \qquad (1)$$

- Each control point $\mathbf{p}_i$ traces a Bézier curve of degree n with control points $\mathbf{p}_{i,j}$, $j=0,1,\ldots,n$ in terms of parameter v:

$$\mathbf{P}_i^n(v) = \sum_{j=0}^{n} B_j^n(v)\mathbf{p}_{i,j}, \quad v \in [0,1] \qquad (2)$$

  - Every point of the initial curve will trace a Bézier curve of degree n.
  - All these curves will generate a *tensor product Bézier surface*

- Replace the points $\mathbf{p}_i$ in (1) with the curve $\mathbf{P}_i^n(v)$ that the surface traces
- The equation of a surface $\mathbf{P}^{m,n}(u,v)$ of degree m in u and degree n in v is:

$$\mathbf{P}^{m,n}(u,v) = \sum_{i=0}^{m} B_i^m(u)\left(\sum_{j=0}^{n} B_j^n(v)\mathbf{p}_{i,j}\right) = \sum_{i=0}^{m}\sum_{j=0}^{n} B_i^m(u)B_j^n(v)\mathbf{p}_{i,j}, \quad u \in [0,1], v \in [0,1] \qquad (3)$$

Tensor Product Bézier Surfaces (2)

- Control points (or control net): the $(m+1)\times(n+1)$ control points that are used for the definition of the surface
- They can be shown in a rectangular arrangement:

	$v \rightarrow$			
u	$\mathbf{p}_{0,0}$	$\mathbf{p}_{0,1}$	$\cdots$	$\mathbf{p}_{0,n}$
$\downarrow$	$\mathbf{p}_{1,0}$	$\mathbf{p}_{1,1}$	$\cdots$	$\mathbf{p}_{1,n}$
	$\vdots$	$\vdots$		$\vdots$
	$\mathbf{p}_{m,0}$	$\mathbf{p}_{m,1}$	$\cdots$	$\mathbf{p}_{m,n}$

(T)

- Boundary curves of the Bézier surface: the isoparametric curves that correspond to $u=0$, $u=1$, $v=0$ and $v=1$
- The construction is symmetric: the same surface is constructed by starting with any of the boundary curves

The de Casteljau algorithm

- To compute a point $\mathbf{P}^{m,n}(u,v)$ of a Bézier surface:
  - Apply the de Casteljau algorithm to each of the rows of (T) for v
  - $(m+1)$ points are computed which are the control points of the isoparametric curve corresponding to v
  - Apply again the de Casteljau algorithm to these points for u
  - The resulting point is $\mathbf{P}^{m,n}(u,v)$

The de Casteljau algorithm (2)

The de Casteljau algorithm for tensor product Bézier surfaces:

```
point bezierSurfacePoint ( int m, int n,
                           point[][] controlPt,
                           float u, float v )
{
    point tmpPt[n+1];
    point curvePt[n+1];

    for (i=0; i <= m; i++)  {
        for (j=0; j <= n; j++)  {
            tempPt[j] = controlPt[i][j];
        }
        curvePt[i] = bezierPoint(n, tempPt, v);
    }

    return bezierPoint(m, curvePt, u);
}
```

Tensor Product Bézier Surfaces:Properties

- These properties are generalizations of the properties of Bézier curves:
- Bézier surfaces have:
  - Convex-hull property
  - Invariance under affine transformations
  - Invariance under affine transformations of their parameter
  - Symmetry with respect to their control points
  - Planar precision
  - Boundary control points interpolation
  - Pseudo-local control
- Surfaces don't have the variation-diminishing property
- The partial derivatives are the tangents of isoparametric curves of the surface
- The partial derivatives with respect to *u* are:

$$\frac{\partial}{\partial u}\mathbf{P}^{m,n}(u,v) = \sum_{j=0}^{n} B_j^n(v)\left(\frac{\partial}{\partial u}\sum_{i=0}^{m} B_i^m(u)\mathbf{p}_{i,j}\right)$$

Bézier surface subdivision

- Generalization of Bézier curve subdivision
- A pair of values (u_0, v_0) is chosen and the surface is subdivided into 4 sub-surfaces of the same type
- The control points of the sub-surfaces are produced using the de Casteljau algorithm:
  - Apply the de Casteljau algorithm to every line of (T)
  - Each of the curves is subdivided into a “left” and a “right” segment
  - 2 sets of $(m+1) \times (n+1)$ control points are created
  - Apply de Casteljau to the $2 \times (m+1) \times (n+1)$ columns of control points
  - Each segment is subdivided into a “top” and “bottom” segment
  - 4 sets of $(m+1) \times (n+1)$ control points result, defining the 4 subsurfaces

Bézier surface subdivision (2)

- Applications:
  - Drawing a Bézier surface:

    if the control points are coplanar

        draw the polygon defined by the 4 extreme control points

    else

        subdivide the surface into 4 subsurfaces for an arbitrary pair (u_0, v_0)

        perform the same procedure recursively to each of the subsurfaces
  - Finding intersections between a Bézier surface and a line or plane

Tensor Product B-Spline Surfaces

- Is a smooth surface constructed by joining together low-degree surfaces with suitable continuity constraints
- Is constructed by parametric surfaces of:
  - Degree k with respect to u
  - Degree l with respect to v

  joined with continuity:
  - C^{k-1} with respect to u
  - C^{l-1} with respect to v
- Has a set of $(m+1)\times(n+1)$ control points $\mathbf{p}_{i,j}$
- m and n are independent of the degrees k and l of the surface
- Has 2 sets of knots:
  - $u_1, u_2, \ldots, u_{m+k}$ for u
  - $v_1, v_2, \ldots, v_{n+l}$ for v

Tensor Product B-Spline Surfaces (2)

- Is given by :

$$\mathbf{Q}(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} N_i^k(u) N_j^\ell(v) \mathbf{p}_{i,j}$$

- The domain of the surface is $[u_k, u_{m+1}]\times[v_l, v_{n+1}]$
- Arrangement that depicts the control elements of a B-spline surface:

		v_1	$\cdots$	v_n	$\cdots$	$v_{n+\ell}$
	$\mathbf{p}_{0,0}$	$\mathbf{p}_{0,1}$	$\cdots$	$\mathbf{p}_{0,n}$		
u_1	$\mathbf{p}_{1,0}$	$\mathbf{p}_{1,1}$	$\cdots$	$\mathbf{p}_{1,n}$		
$\vdots$	$\vdots$	$\vdots$		$\vdots$		
u_m	$\mathbf{p}_{m,0}$	$\mathbf{p}_{m,1}$	$\cdots$	$\mathbf{p}_{m,n}$		
$\vdots$						
u_{m+k}						

The de Boor Algorithm

- Compute a point $\mathbf{Q}(u, v)$ on a tensor product B-spline surface:
  - Find the interval $[u_i, u_{i+1}] \times [v_j, v_{j+1}]$ into which (u, v) belongs
  - Apply the de Boor algorithm to each of the rows of the control net for the value $v \in [v_j, v_{j+1}]$
  - This produces $(m+1)$ points
  - Apply the de Boor algorithm to each of the $(m+1)$ points for $u \in [u_i, u_{i+1}]$
- The procedure can be carried out first for u (columns of the control net) and then for v without changing the result.

Knots and Parameterizations

- The 2 knot sequences, for u and v maintain all the properties of the knot sequence of a B-spline curve
- It is difficult to find automatically knot sequences
- Knot insertion is not difficult & uses the Boehm algorithm
- A knot may be inserted independently in any of the 2 knot sequences
  - If a knot is inserted in $u_1, u_2, \ldots, u_{m+k}$
    - apply the Boehm algorithm to the columns of the control net
    - a full row of control points is added to the control net
  - Similarly if a knot is inserted in $v_1, v_2, \ldots, v_{n+l}$
    - apply the Boehm algorithm to the rows of the control net
    - a full column of control points is added to the control net

Properties

- Are generalizations of properties of B-spline curves
- Tensor Product B-spline Surfaces have:
  - Local control
  - Strong convex-hull property
  - Invariance under affine transformations
  - Invariance under affine transformations of their parameters
  - Planar precision
  - Boundary control points interpolation if the extreme knots have suitable multiplicity
- They don’t have the variation-diminishing property

Interpolation

- Bi-cubic tensor product B-spline surfaces $\mathbf{Q}(u, v)$: surfaces of degree 3 for both u and v
- Given
  - $(m+1)\times(n+1)$ points $\mathbf{p}_{i,j}$, $i=0,1,\ldots,m$ and $j=0,1,\ldots,n$
  - Two knot sequences u_i, $i=0,1,\ldots,m$ and v_j, $j=0,1,\ldots,n$

  determine the $(m+3)\times(n+3)$ control points $\mathbf{q}_{i,j}$ of $\mathbf{Q}(u, v)$ that satisfy:

$$\mathbf{Q}(u_i, v_j) = \mathbf{p}_{i,j}, \quad i = 0,1,\ldots,m \text{ and } j = 0,1,...,n$$

- Computation :
  - Interpolate each column of the array of points $\mathbf{p}_{i,j}$ in terms of u
    - Compute the control points $\mathbf{s}_{i,j}$ of the respective interpolation curves
    - Each of the curves has $(m+3)$ control points and requires also $\mathbf{r}_{0,j}$ and $\mathbf{r}_{1,j}$, $j=0,1,\ldots,n$
  - Interpolate each row of $\mathbf{s}_{i,j}$ in terms of v
    - Compute the control points $\mathbf{q}_{i,j}$
    - Each of the $(m+3)$ curves has $(n+3)$ control points and requires $c_{i,0}$ and $c_{i,1}$