

Curves and Surfaces

Lecturer:

Carol O'Sullivan

Professor of Visual Computing

Carol.OSullivan@cs.tcd.ie

Course www:

<http://isg.cs.tcd.ie/cosulliv/>

Overview

- Curves and surfaces
- Parametric form
- Representing curves
- Interpolation vs. approximation
- Splines
- Curve continuity
- Cubic parametric curves
- Geometry matrix
- Blending functions
- Hermite curves and Bézier curves

Introduction

- An understanding of curves is a key to understanding surfaces since most mathematical methods for curves can be extended to surfaces
- An important term in the field is interpolation which comes from Latin inter (between) and polare (to polish)
 - It means to compute new values that lie between (or that are an average of) certain given values
- Another important term is approximation
 - Compute a curve or a surface that passes close to the control points but not necessarily through them

Curves and Surfaces

- Often we are required to represent surfaces that are not planar in nature.
- To do so the parametric representation of 2-dimensional curves and 3-dimensional surfaces may be employed.
- In general for any surface or curve, there is both a parametric and an implicit representation.
- In computer graphics it is often more convenient to adopt the parametric form.

Sphere definition

- For example, the implicit form of a sphere is:

$$x^2 + y^2 + z^2 - r^2 = 0$$

- In general all implicit representations of a 3-d surface are of the form:

$$f(x, y, z) = 0$$

- The parametric form of a sphere is:

$$f: (\theta, \phi) \mapsto (\cos \theta \cos \phi, \sin \theta, \cos \theta \sin \phi)$$

Parametric form

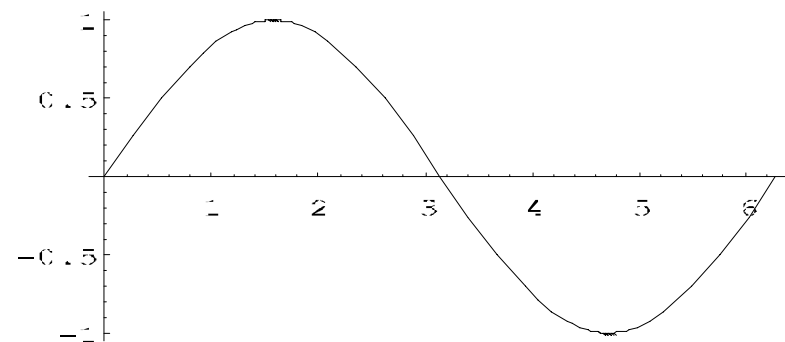
- The general form of a 3-d surface in its parametric (or *explicit*) representation is:

$$f(u, v) = (f_x(u, v), f_y(u, v), f_z(u, v))$$

- When rendering such curves and surfaces using polygons, the parametric representation is more convenient as the surface is defined in terms of a parametric variable or variables.
- This allows the exact determination of the value of the surface at regular intervals, thus allowing an approximation to the surface by taking a number of such "samples".

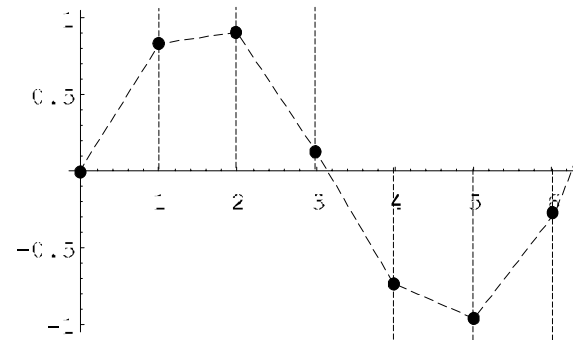
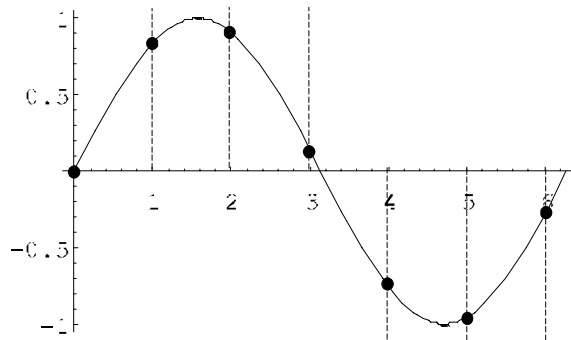
Example - 1

- Determine the representation of the function $f(\theta) = \sin(\theta)$.
- This is a parametric description of a curve in 2 dimensions with parameter θ .
- This is an example of an unbounded curve (in that we can take values of θ from $-\infty \dots +\infty$). We'll limit our curve to the domain $(0 \dots 2\pi)$. This gives the following curve:



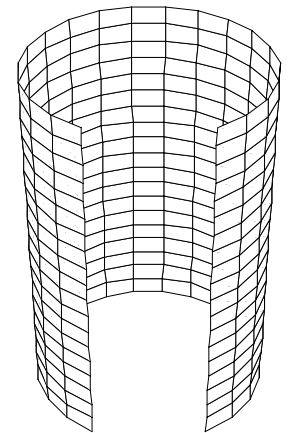
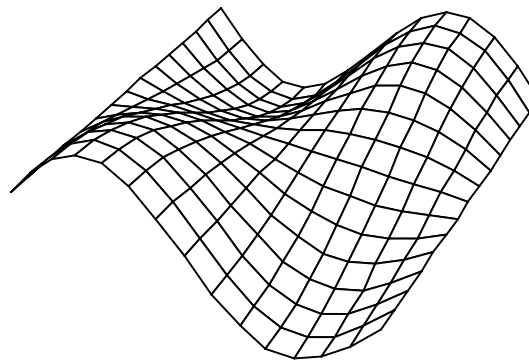
Example - 2

- Now we must determine how fine or coarse a representation we need to use in order to faithfully capture this curve.
- We will sample the curve at regular intervals of θ along the length of the curve. In this example, the curve will be sampled at regular points a unit distance apart (i.e. at $\theta = 0, 1, 2\dots$).
- This yields the following sample points which we will join by straight lines which is the way the curve will be finally displayed on the raster:



Surfaces

- Note that the final representation is not very smooth. If the intervals are chosen carefully, however (for example, by relating the interval distance to the size of a pixel of the raster), then the curve representation will appear continuous and smooth.
- This technique may be extended to surfaces in the same manner (surfaces require 2 parameters):



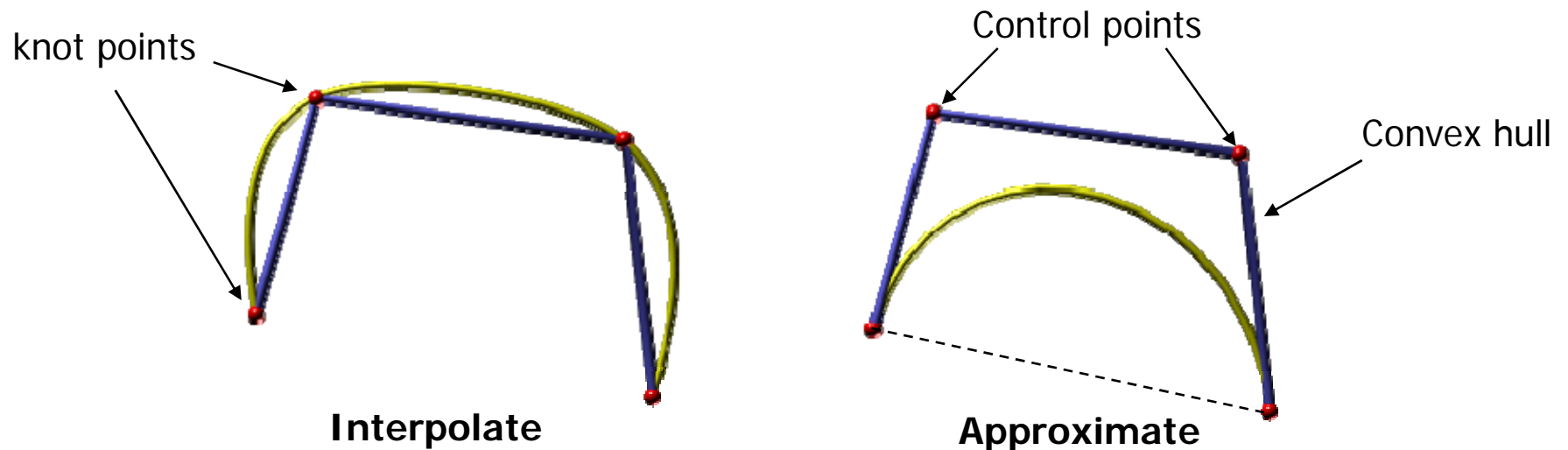
Representing Curves

There are many different methods of representing general curves (rather than attempt to model all surfaces as some existing function, say a Sine or Cosine). The most common are:

- **Cubic Splines**
- **Bezier Curves**
- **B-splines**
- **NURBS and β -splines**

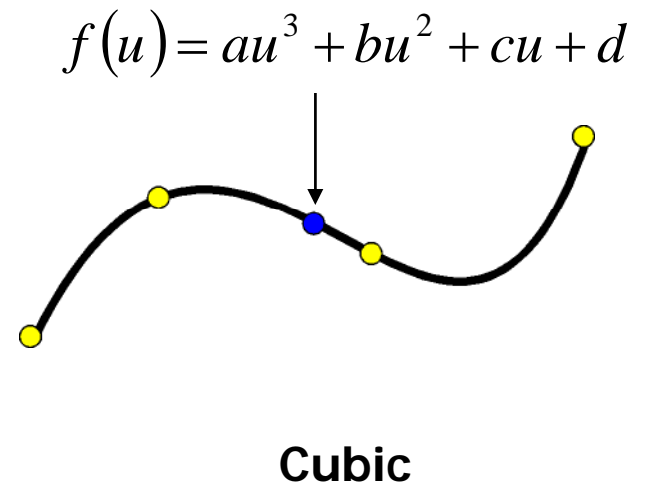
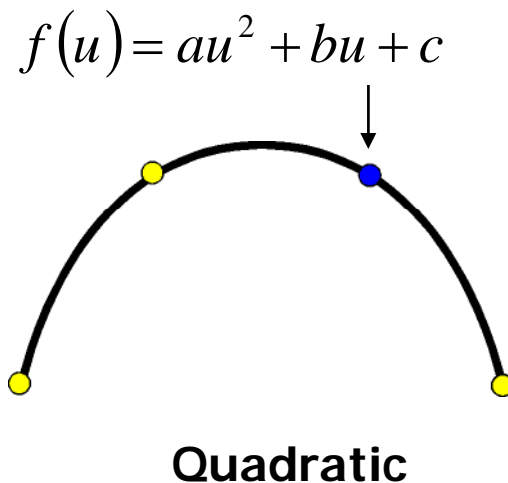
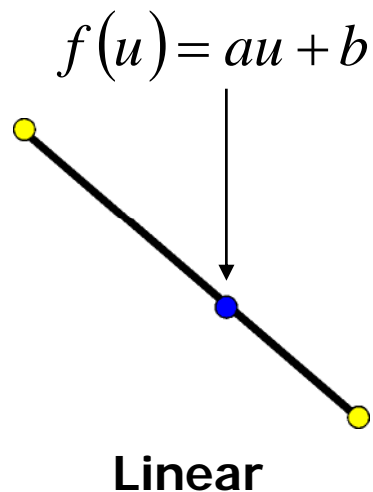
Interpolation vs. Approximation

- Given a set of n points, to create a curve we either
 - *interpolate* the points (curve passes through all points)
 - *approximate* the points (points describe *convex hull* of curve)
- Points on curve = *knot points*
- Points on convex hull (off curve) = *control points*



Splines

- To interpolate we can use a simple polynomial spline. With n points we require a polynomial of degree $n-1$ (order n polynomial).
- Let $f(u)$ be the parameterised polynomial where $0 \leq u \leq 1$



Splines

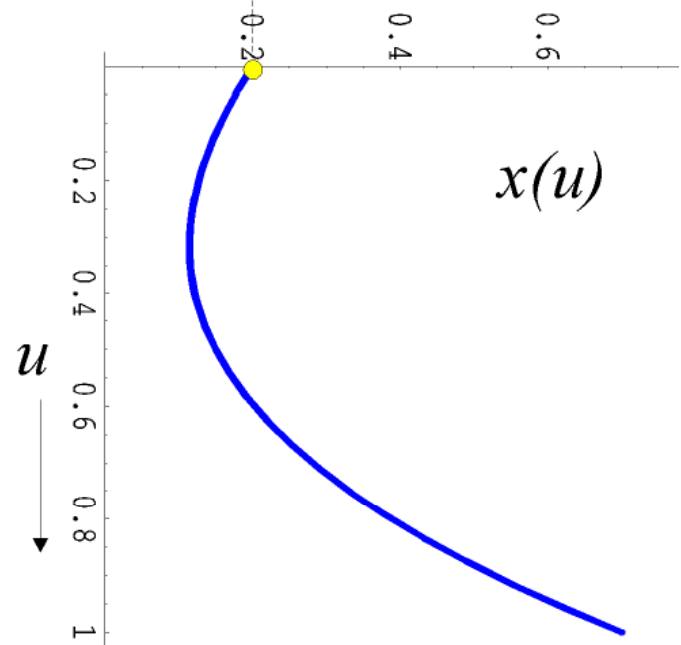
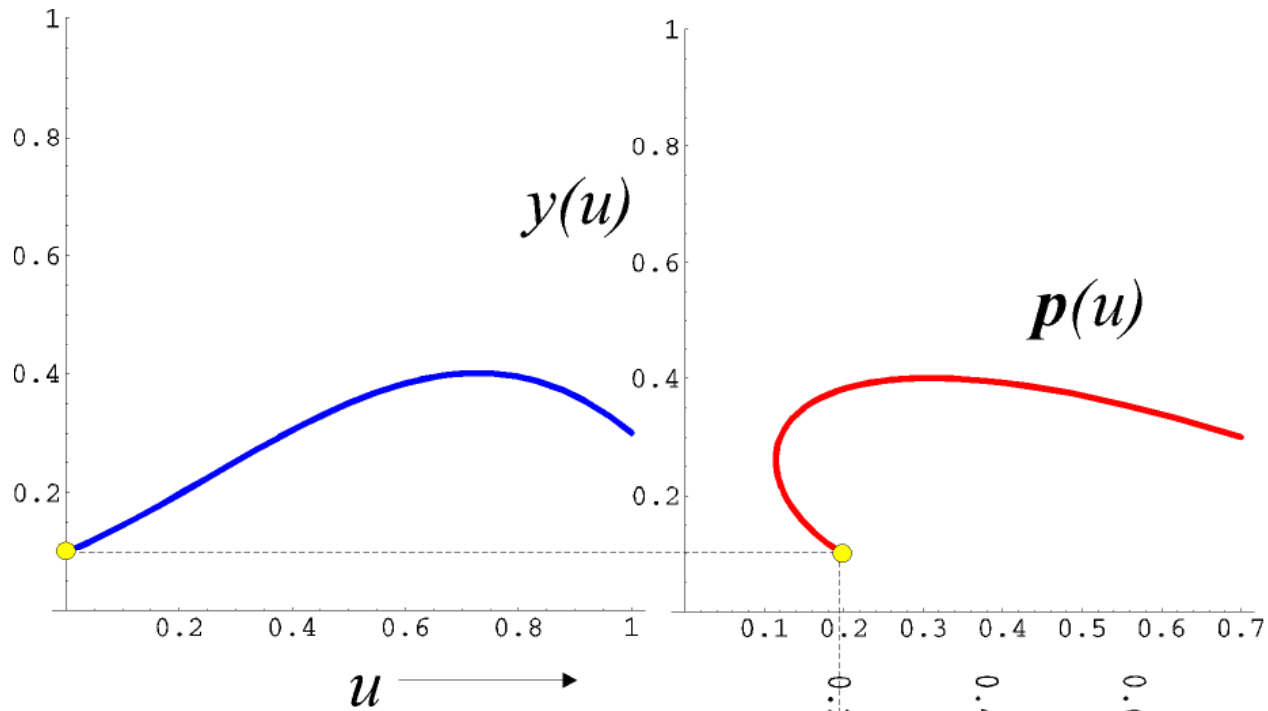
- These polynomials are plots of $f(u)$ with respect to u
 - for each u , there is one and only one $t(u)$
⇒ curve cannot turn back on itself

- Use polynomials for each axis (in 2D we have 2 polys):

$$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$$

$$y(u) = a_y u^3 + b_y u^2 + c_y u + d_y$$

- As before we limit u to $[0,1]$, although the polynomial is defined for all values of u .



$$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$$

$$y(u) = a_y u^3 + b_y u^2 + c_y u + d_y$$

Splines

- For a 3D spline, we have 3 polynomials:

$$\left. \begin{aligned} x(u) &= a_x u^3 + b_x u^2 + c_x u + d_x \\ y(u) &= a_y u^3 + b_y u^2 + c_y u + d_y \\ z(u) &= a_z u^3 + b_z u^2 + c_z u + d_z \end{aligned} \right\}$$

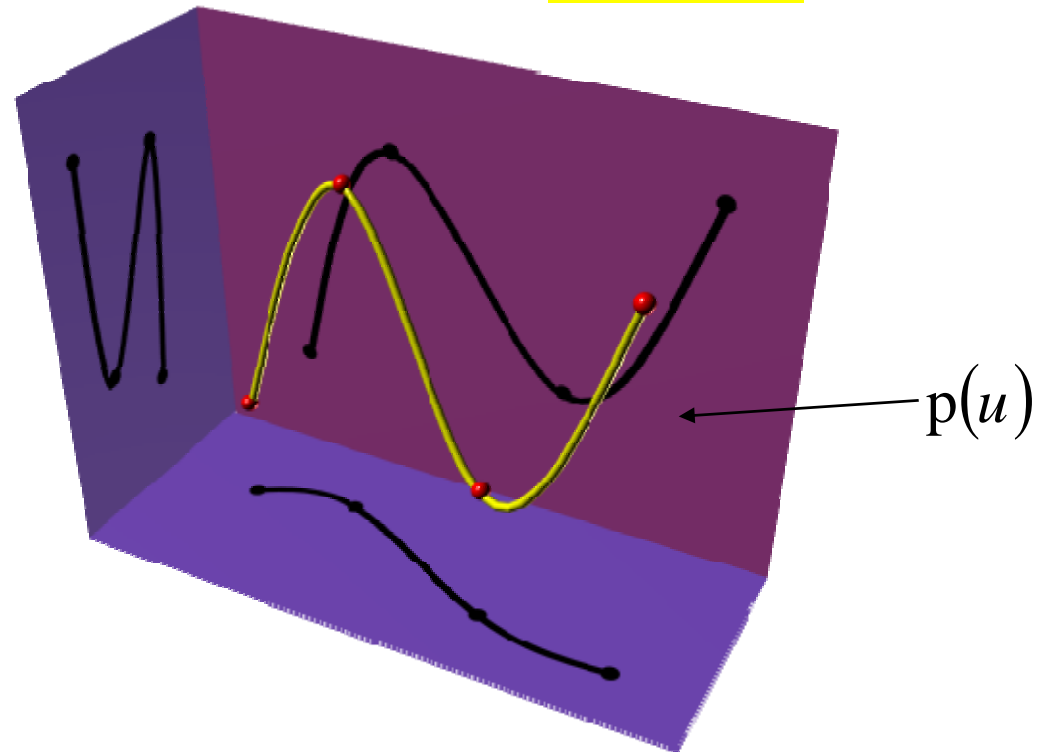
$$\rightarrow [x(u) \quad y(u) \quad z(u)] = [u^3 \quad u^2 \quad u \quad 1]$$

$$\begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix}$$

$$\rightarrow \mathbf{p}(u) = \mathbf{u} \cdot \mathbf{C}$$

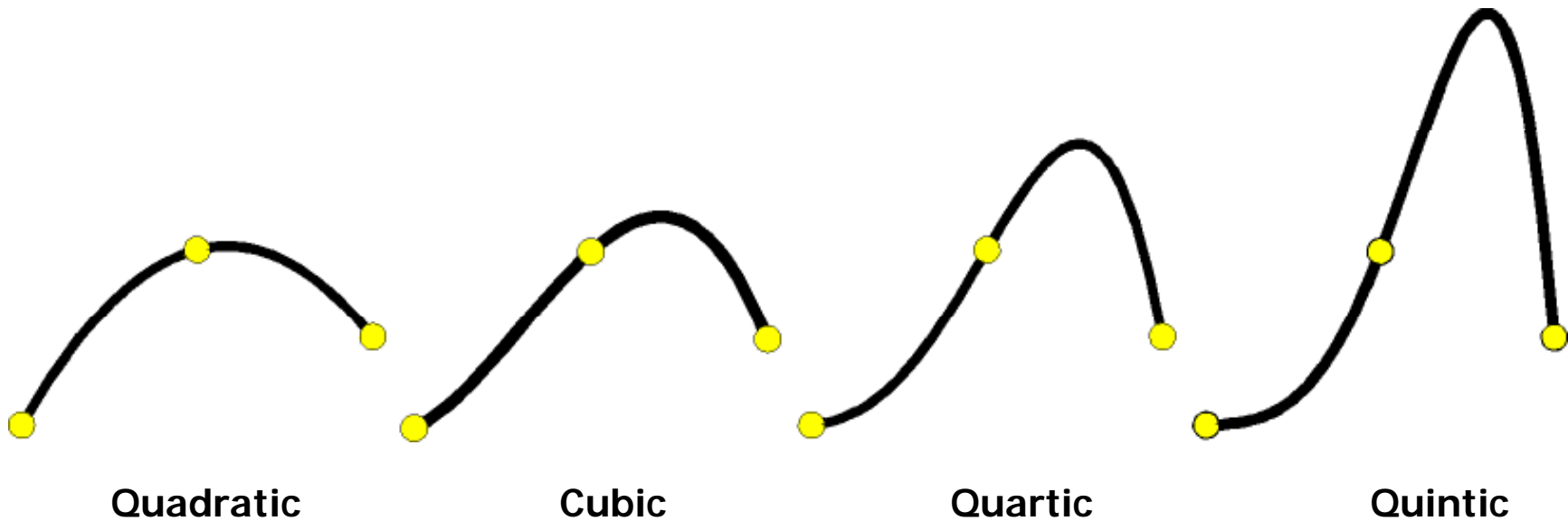
12 unknowns
 \therefore 4 3D points required

Defines the variation in x with distance u along the curve



Splines

- If we have more than 4 points we require a polynomial of higher degree
 - higher degree polynomials are more difficult to control
 - they exhibit unwanted *wiggles* (oscillations)

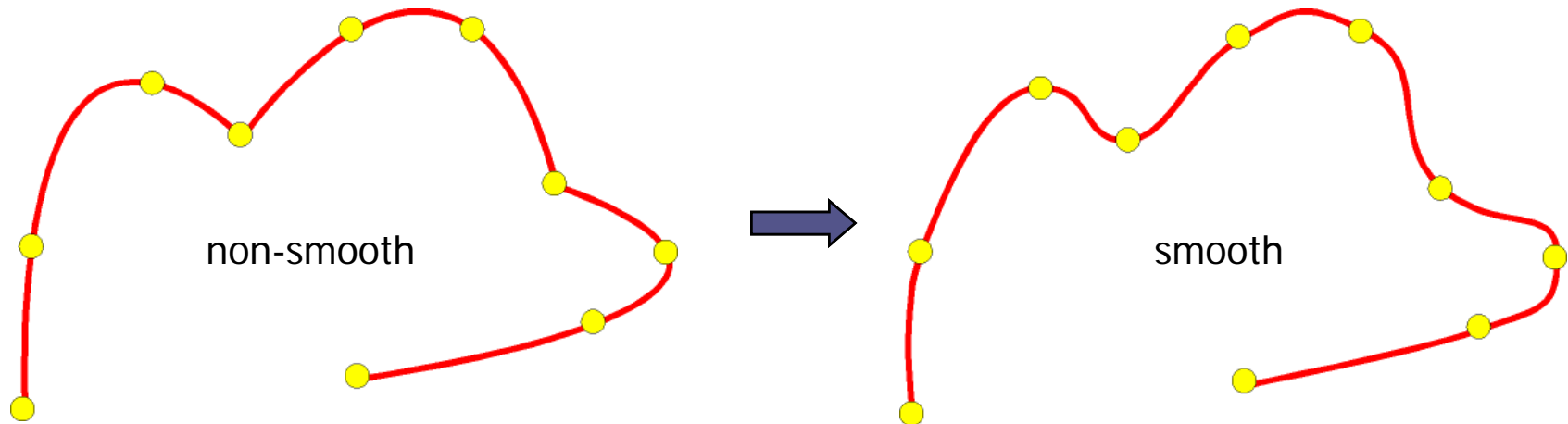


Splines

- In general we use cubic polynomials for curves in CG:
 - minimal wiggles and faster to compute than high degree polynomials
 - lowest degree which allows *non-planar curves* (quadratics require 3 points, 3 points always lie in the same plane)

Defining the Cubic Spline

- Normally we supply 4 points we wish the spline to pass through.
- If we have more than 4 points we must employ more than 1 spline \Rightarrow use a *piecewise cubic polynomial*
 - for n points, we have $(n-1)/3$ individual cubic segments
 - without further constraints these will not join smoothly



Curve Continuity

- To ensure a smooth connection between curve segments we enforce further *continuity* constraints
- 2 types of continuity:
 - *parametric continuity*, denoted C^n where n = degree of continuity
 - *geometric continuity*, denoted G^n
- Given a curve such that at point p , 2 segments $c_i(u)$ and $c_{i+1}(u)$ meet then:

$$p = c_i(1) = c_{i+1}(0)$$

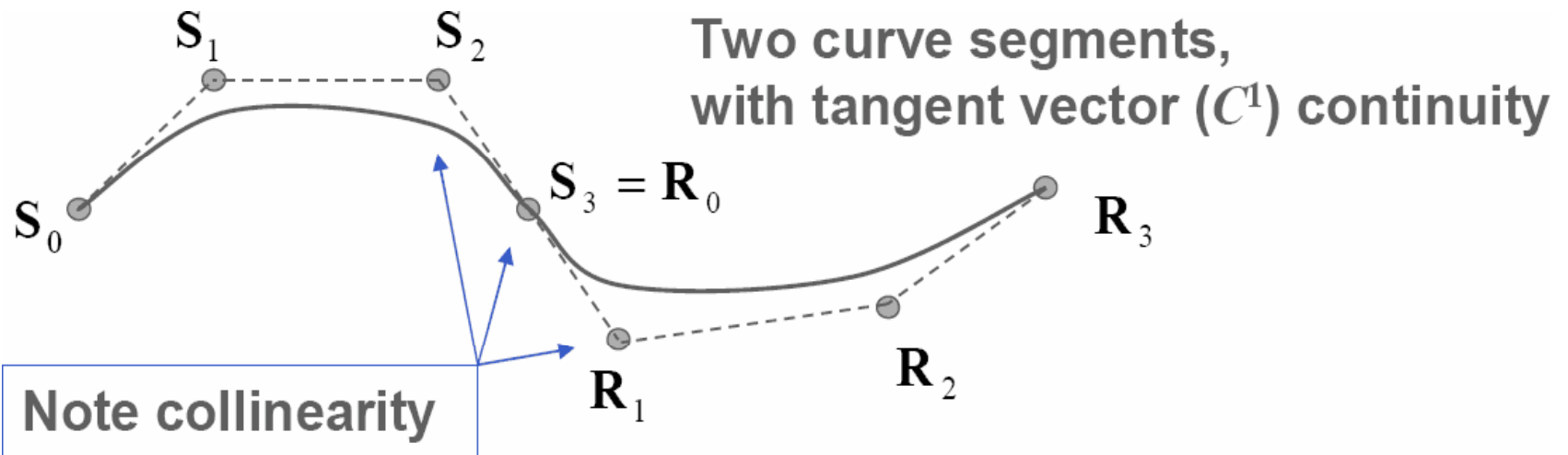
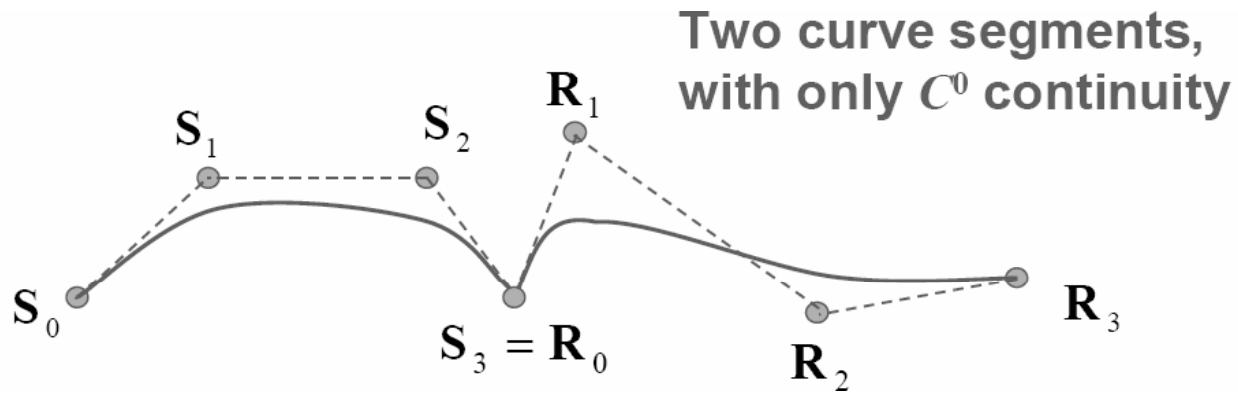
$$C^n \Rightarrow \left. \frac{d^n c_i(u)}{du^n} \right|_{u=1} = \left. \frac{d^n c_{i+1}(u)}{du^n} \right|_{u=0}$$

differentials are equal

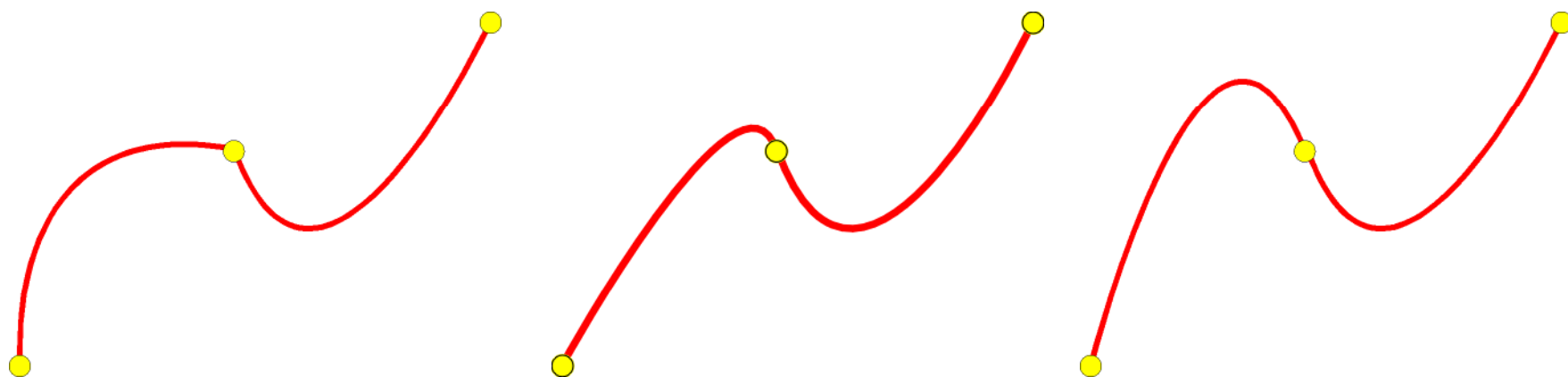
$$G^n \Rightarrow \left. \frac{d^n c_i(u)}{du^n} \right|_{u=1} = \alpha \left. \frac{d^n c_{i+1}(u)}{du^n} \right|_{u=0}$$

differentials are proportional

Curve Continuity



Examples of Continuity



Cubic Parametric Curves

- A curve segment $\mathbf{p}(u)$ is defined by constraints on end-points, tangent vectors, and continuity between curve segments.
- Each cubic polynomial has 4 co-efficients, so four constraints will be needed.
- Remember:

$$\left. \begin{aligned} x(u) &= a_x u^3 + b_x u^2 + c_x u + d_x \\ y(u) &= a_y u^3 + b_y u^2 + c_y u + d_y \\ z(u) &= a_z u^3 + b_z u^2 + c_z u + d_z \end{aligned} \right\} \rightarrow [x(u) \quad y(u) \quad z(u)] = [u^3 \quad u^2 \quad u \quad 1] \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix} \rightarrow \mathbf{p}(u) = \mathbf{u} \cdot \mathbf{C}$$

- This allows us to formulate 4 equations in the 4 unknowns, and then solve for the unknowns.

Geometry Matrix

- To see how the co-efficients can depend on 4 constraints, recall that a parametric cubic curve is defined by $p(u) = \mathbf{u} \cdot \mathbf{C}$
- Rewrite the co-efficient matrix as $\mathbf{C} = \mathbf{M} \cdot \mathbf{G}$ where \mathbf{M} is a 4x4 **basis matrix**, and \mathbf{G} is a 4-element matrix of geometric constraints, called the **geometry matrix**.
- The geometric constraints are just the conditions, such as endpoints, or tangent vectors, that define the curve.
 - G_x refers to the column vector of just the x components; G_y and G_z are similarly defined
- \mathbf{G} or \mathbf{M} , or both \mathbf{G} and \mathbf{M} , differ for each type of curve.

Geometry Matrix

- The elements of **G** and **M** are constants so the product **G.M.u** is just three cubic polynomials in u .
- Expanding:

$$\mathbf{p}(u) = [x(u) \quad y(u) \quad z(u)] = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix}$$

$$= \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \begin{bmatrix} g_{1x} & g_{1y} & g_{1z} \\ g_{2x} & g_{2y} & g_{2z} \\ g_{3x} & g_{3y} & g_{3z} \\ g_{4x} & g_{4y} & g_{4z} \end{bmatrix}$$

Blending Functions

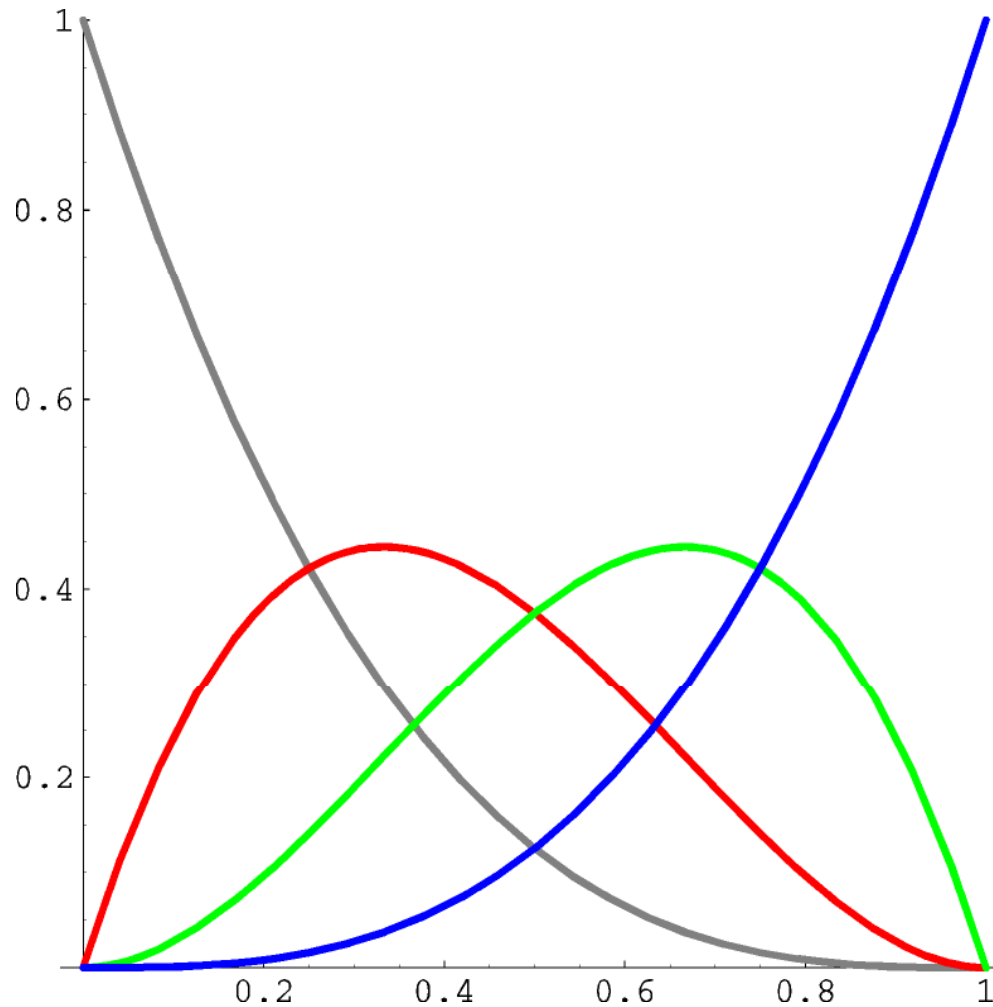
- We can read this equation in the following way:
 - The point $p(u)$ is a weighted sum of the columns of the geometry matrix G , each of which represents a point or a vector in 3-space
- Multiplying out just $x(u)$ gives:

$$\begin{aligned}x(u) = & (u^3 m_{11} + u^2 m_{21} + u m_{31} + m_{41}) g_{1x} + \\ & (u^3 m_{12} + u^2 m_{22} + u m_{32} + m_{42}) g_{2x} + \\ & (u^3 m_{13} + u^2 m_{23} + u m_{33} + m_{43}) g_{3x} + \\ & (u^3 m_{14} + u^2 m_{24} + u m_{34} + m_{44}) g_{4x}\end{aligned}$$

Blending Functions

- This emphasizes that the curve is a weighted sum of the elements of the geometry matrix.
- The weights are each cubic polynomials of the parameter u , and are called the blending functions.
- The blending functions B are given by $\mathbf{u} \cdot \mathbf{M}$
- This is similar to linear interpolation, for which only two geometric constraints (i.e. the endpoints of the line) are needed.
- Parametric cubics are really just a generalization of straight-line approximations:
 - The cubic curve $\mathbf{p}(u)$ is a combination of the **4** rows of the geometry matrix, whereas the line is an affine combination of **2** points.

Sample Blending Polynomials

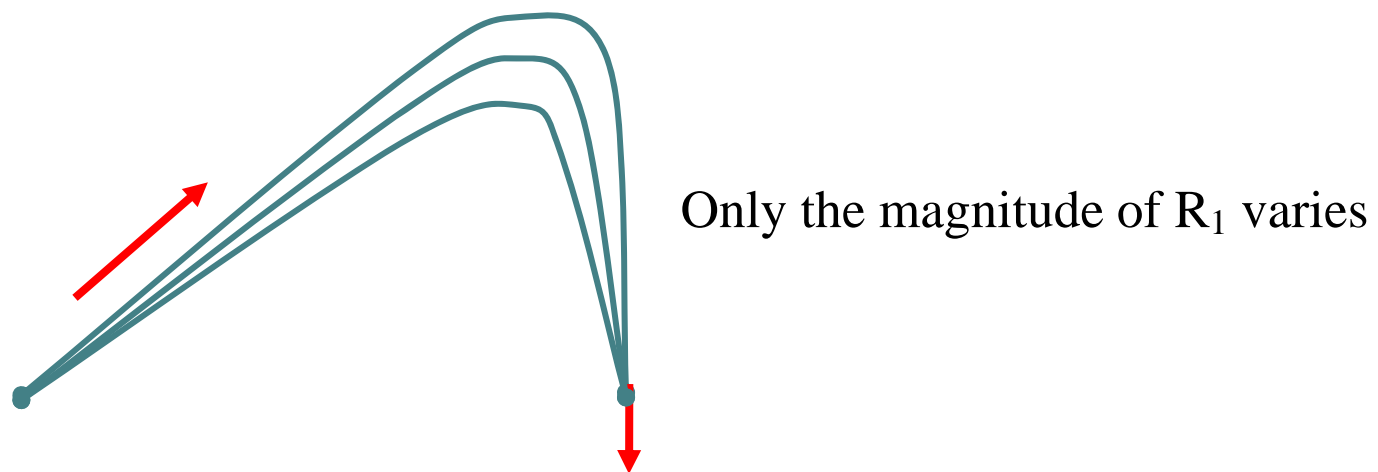
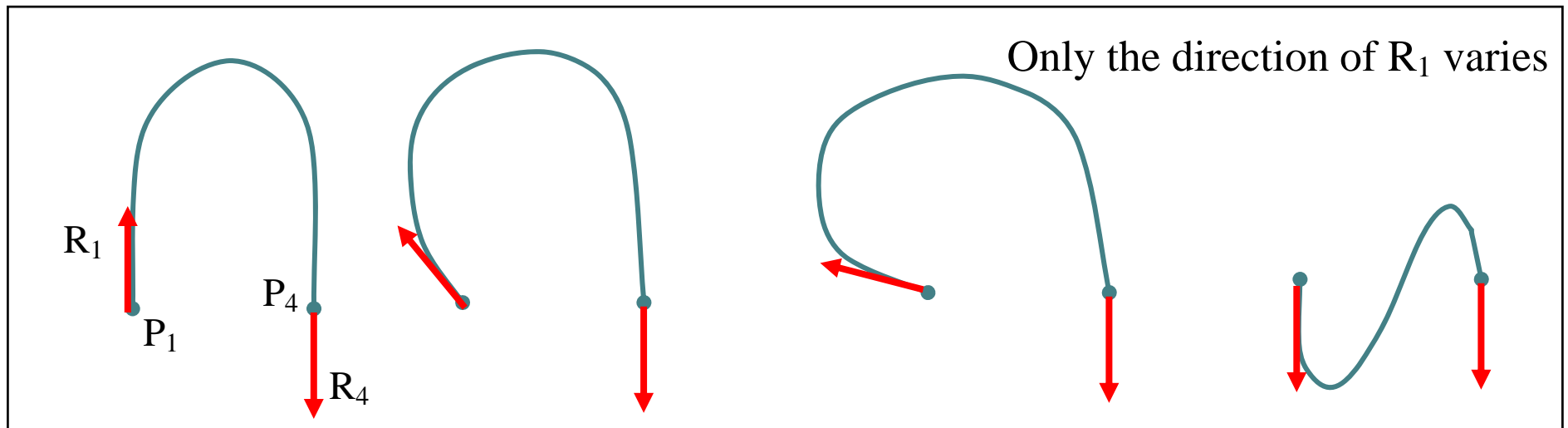


The Bernstein polynomials, weighting functions for Bézier curves

Hermite Curves

- The key to defining a parametric cubic curve therefore lies in the basis matrix M .
- Depending on the nature of this matrix, specific forms of curves may be created.
- The Hermite form of a cubic polynomial curve segment is determined by constraints on the endpoints P_1 and P_4 , and tangent vectors at the endpoints R_1 and R_4 .

Hermite Curves - Examples



Hermite Geometry Vector

- The Hermite Geometry vector G_H is

$$G_H = \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}$$

- G_{Hx} is the x component of G_H so:

$$G_{Hx} = \begin{bmatrix} P_{1x} \\ P_{4x} \\ R_{1x} \\ R_{4x} \end{bmatrix}$$

Hermite Curves

- The Hermite basis matrix, M_H , relates the Hermite Geometry vector G_H to the polynomial coefficients.
- Therefore:

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x = T \cdot M_H \cdot G_{Hx}$$

- where $T = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix}$

Hermite Curves

- The constraints on $x(0)$ and $x(1)$ are found by direct substitution into the previous equation:

$$x(0) = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \cdot M_H \cdot G_{Hx} = P_{1x}$$

$$x(1) = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \cdot M_H \cdot G_{Hx} = P_{4x}$$

Hermite Curves

- The tangent vector constraints on $x(0)$ and $x(1)$ are found by differentiation, i.e:

$$x'(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \cdot M_H \cdot G_{Hx}$$

- So: $x'(0) = R_{1x} = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \cdot M_H \cdot G_{Hx}$

- and $x'(1) = R_{4x} = \begin{bmatrix} 3 & 2 & 1 & 0 \end{bmatrix} \cdot M_H \cdot G_{Hx}$

Hermite Curves

- The four constraints can be written in matrix form as:

$$\begin{bmatrix} P_{1x} \\ P_{4x} \\ R_{1x} \\ R_{4x} \end{bmatrix} = G_{Hx} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \cdot M_H \cdot G_{Hx}$$

- The only way that this equation can be satisfied is if M_H is the inverse of the given 4x4 matrix, so:

$$M_H = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Hermite Blending Functions

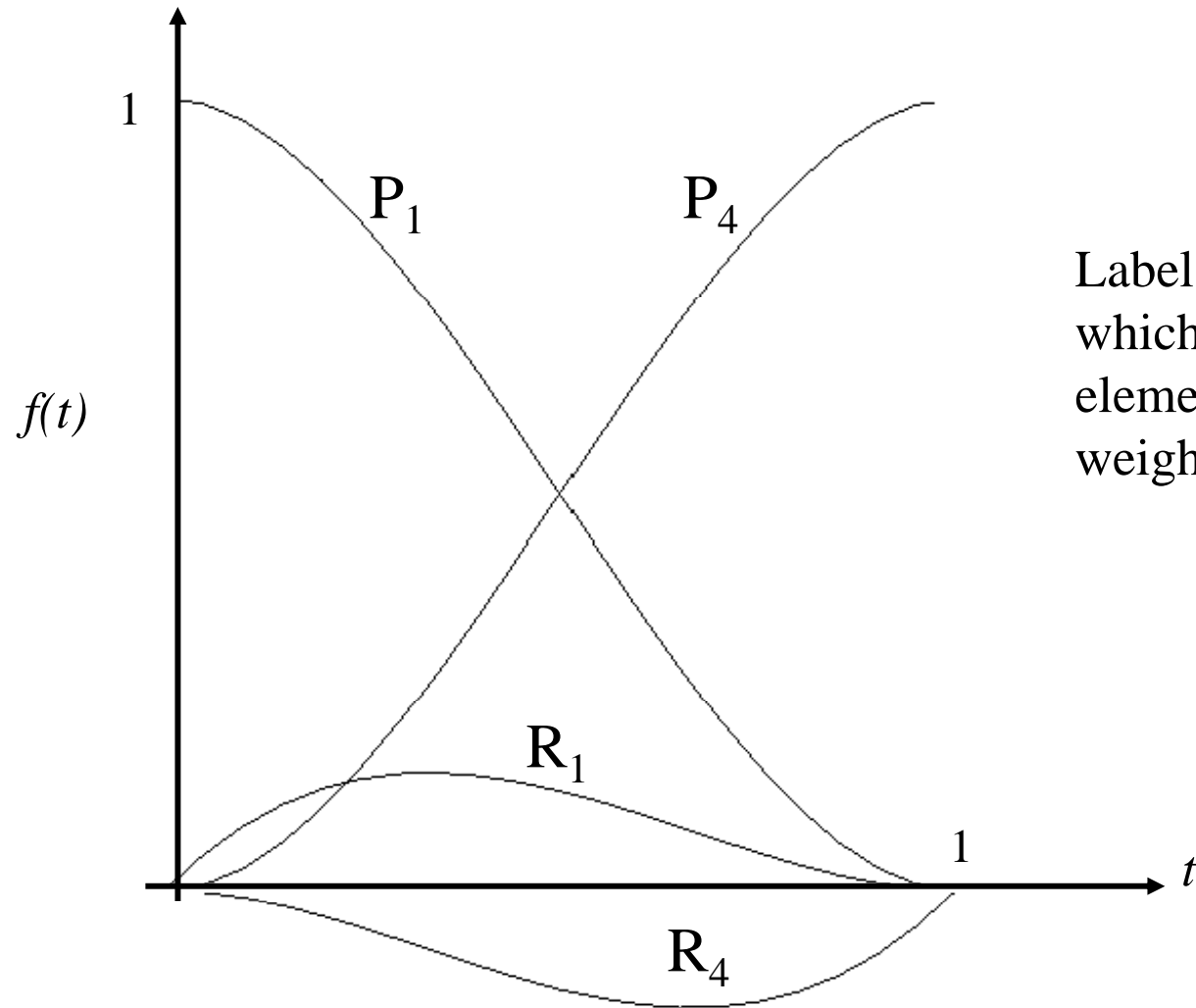
- We know that:

$$p(t) = T \cdot M_H \cdot G_H$$

- The Hermite blending functions B_H are given by $T \cdot M_H$, since these weight the geometry vector G_H .
- Therefore:

$$p(t) = T \cdot M_H \cdot G_H = B_H \cdot G_H = \begin{aligned} & (2t^3 - 3t^2 + 1)P_1 + \\ & (-2t^3 + 3t^2)P_4 + \\ & (t^3 - 2t^2 + t)R_1 + \\ & (t^3 - t^2)R_4 \end{aligned}$$

Hermite Curves - Blending Functions



Labels show
which geometry
element is
weighted.

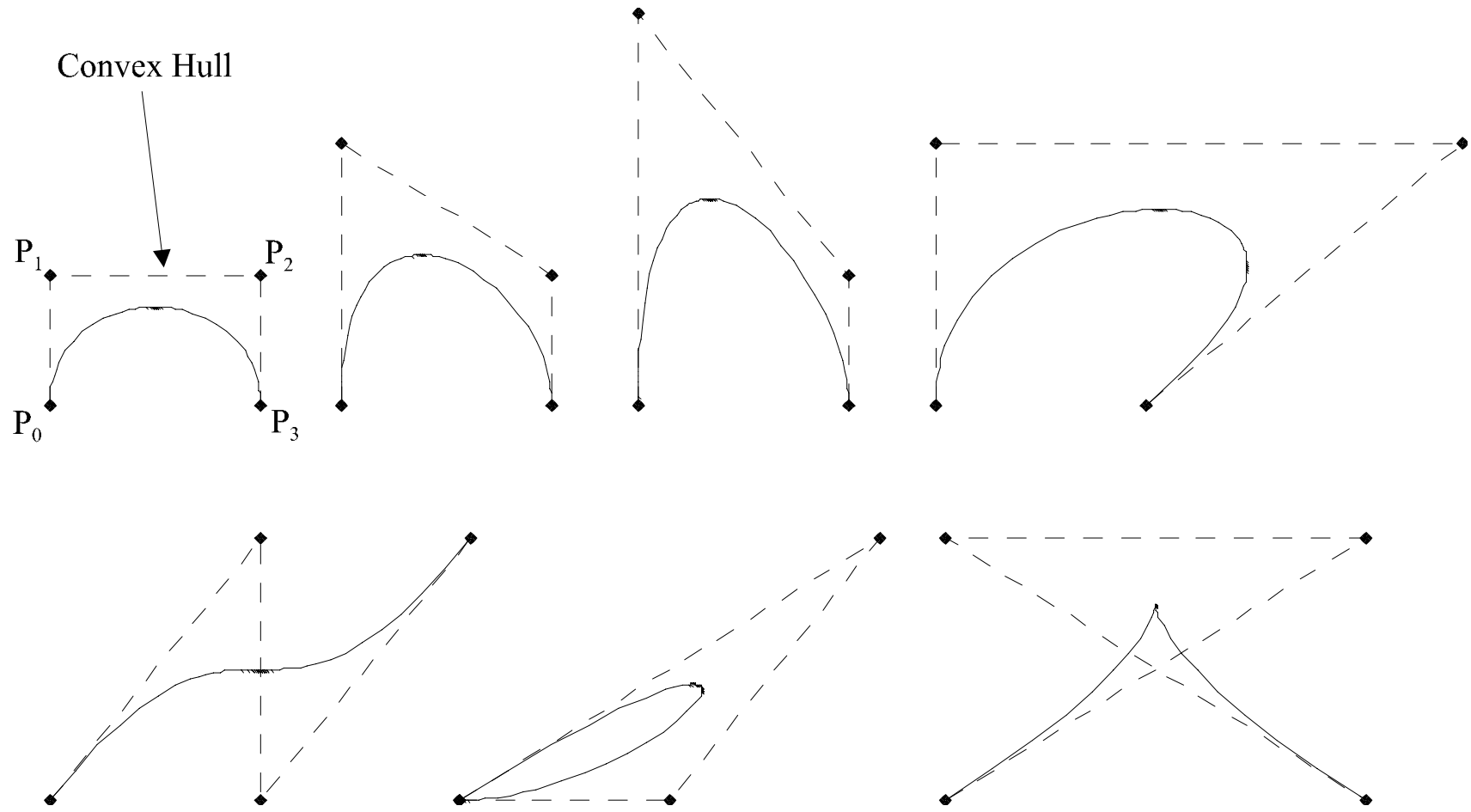
Bézier Curves

- The drawback of the Hermite form is the need to explicitly specify the tangent vectors.
- The Bézier form of the cubic polynomial curve segment indirectly specifies the endpoint tangent vector.
- Such curves are constrained by their endpoints: P_1 and P_4 , and also by intermediate points that are not on the curve: P_2 and P_3 .
- The starting and ending tangent vectors are determined by the vectors P_1P_2 and P_3P_4 and are related to the Hermite R_1 and R_4 by:

$$R_1 = p'(0) = 3(P_2 - P_1)$$

$$R_4 = p'(1) = 3(P_4 - P_3)$$

Examples of some Bézier Curves



Bézier Curves

- The reason for using the constant 3 is apparent from the following:
 - Consider the Bezier curve defined by the 4 equally spaced points: $(0,0)$, $(0,1)$, $(0,2)$, $(0,3)$.



- This curve has the definition: $p(t) = P_1 + t(P_4 - P_1)$
- Therefore: $p'(t) = P_4 - P_1$
- If velocity is to be constant everywhere on the line (the tangent vectors to the curve):

$$R_1 = p'(0) = P_4 - P_1 = 3(P_2 - P_1)$$

$$R_4 = p'(1) = P_4 - P_1 = 3(P_4 - P_3)$$

Bézier Geometry Vector and Change of Basis

- The Bézier geometry vector is: $G_B = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$
- A *change of basis* matrix M_{HB} defines the relationship between the Hermite geometry vector G_H and the Bézier geometry vector G_B as follows:

$$G_H = \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} = M_{HB} \cdot G_B$$

Bézier Basis Matrix

- To find the Bezier basis matrix, M_B , consider:

$$\begin{aligned} p(t) &= T \cdot M_H \cdot G_H \\ &= T \cdot M_H \cdot (M_{HB} \cdot G_B) \\ &= T \cdot (M_H \cdot M_{HB}) \cdot G_B \\ &= T \cdot M_B \cdot G_B \end{aligned}$$

- Therefore, we simply calculate:

$$M_B = M_H \cdot M_{HB} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

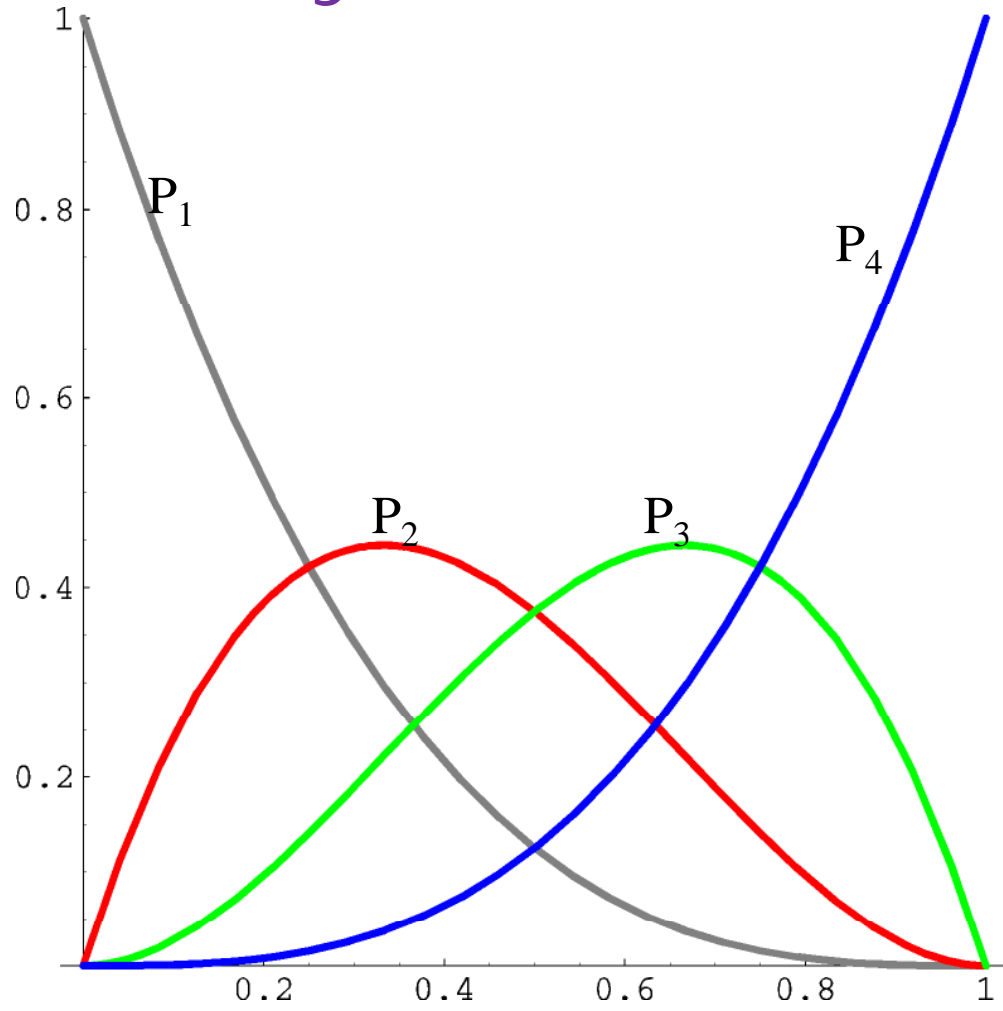
Bernstein Polynomials

- We now have:

$$p(t) = T \cdot M_B \cdot G_B = (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2(1-t) P_3 + t^3 P_4$$

- The four weights are known as the Bernstein Polynomials

Bernstein Polynomials



Labels show which geometry element is weighted.

General Bernstein Form for Bézier Curves

- The Bezier curve $\mathbf{p}(t)$ based on the $(L+1)$ points P_0, P_1, \dots, P_L is given by:

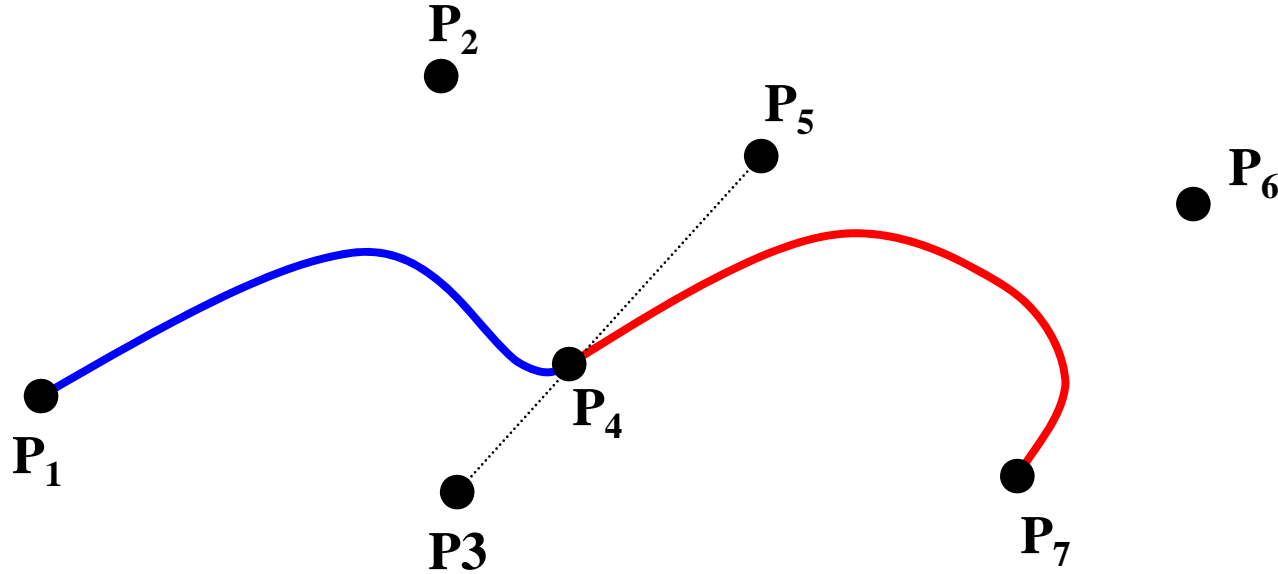
$$\mathbf{p}(t) = \sum_{k=0}^L P_k B_k^L(t)$$

- where $B_k^L(t)$ are the Bernstein polynomials, and the k -th Bernstein polynomial is defined as:

$$B_k^L(t) = \binom{L}{k} (1-t)^{L-k} t^k \quad \text{and} \quad \binom{L}{k} = \frac{L!}{k!(L-k)!} \text{ for } L \geq k$$

Joining Segments

- Consider the following two Bezier curve segments, joined at P_4 :



- Points P_3 , P_4 and P_5 are collinear

Curve Continuity

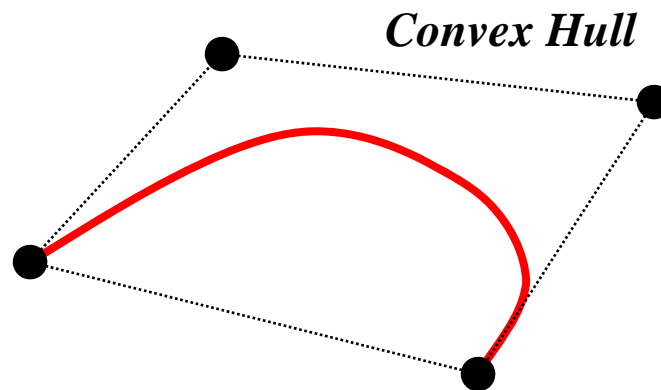
- G^1 continuity is provided at the endpoint when

$$P_3 - P_4 = k(P_4 - P_5), k > 0$$

- i.e. the 3 points P_3 , P_4 and P_5 must be distinct and collinear.
- In the more restrictive case when $k=1$, there is C^1 continuity in addition to G^1 continuity.

Convex Hull Property

- The Bernstein blending polynomials are everywhere non-negative.
- In addition, their sum is everywhere unity (i.e. 1).
- Thus, each curve segment, which is just the sum of four control points weighted by the polynomials, is completely contained within the convex hull of the four control points.



Convex Hull Property

- The convex hull for 2D curves is the convex polygon formed by the 4 control points (e.g. like a rubber band around them)
- For 3D curves, the convex hull is the convex polyhedron formed by the control points (e.g. like cling-film stretched around them.)
- The convex hull property holds for all cubics defined by weighted sums of control points if the blending functions are nonnegative and sum to one.

Convex Hull Property

- One advantageous result of the fact that the blending polynomials sum to 1, is that the value of the fourth polynomial can be found by subtracting the first three from 1.
- The convex hull property is useful for clipping and collision detection, where we can perform tests on the convex hull of a curve before having to perform expensive intersection tests.

B-splines

- A Cubic B-spline approximates a series of $m+1$ control points $P_0, P_1, \dots, P_m, m \geq 3$
- It does this with a curve consisting of $m-2$ cubic polynomial curve segments Q_3, Q_4, \dots, Q_m
- Such cubic curves might be defined each on its own domain $0 \leq t < 1$
- However, we can adjust the parameter (i.e. $t=t+k$) so that the parameter domains for the various curve segments are sequential.

B-splines

- Thus, we can say that the parameter range on which Q_i is defined is:

$$t_i \leq t < t_{i+1}, \text{ for } 3 \leq i \leq m$$

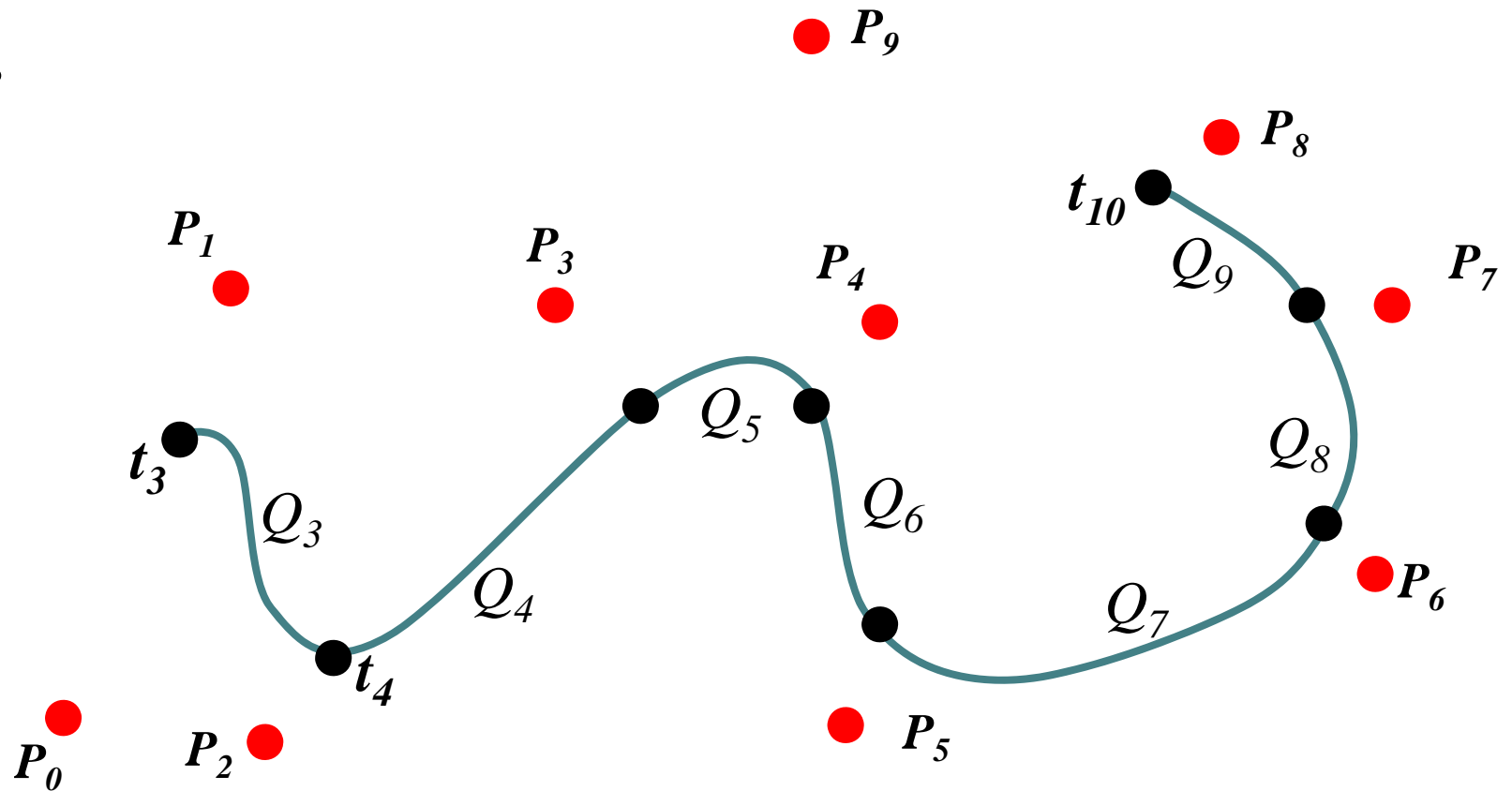
- In the particular case of $m=3$, there is a single curve segment Q_3 that is defined on the interval $t_3 \leq t < t_4$ by four control points, P_0 to P_3 .

Knots

- For each $i \geq 4$, there is a join point or ***knot*** between Q_{i-1} and Q_i at the parameter value t_i
 - t_i is called the ***knot value***.
- The initial and final points at t_3 and t_{m+1} are also called knots, so there are $m-1$ knots in total.

Knots and Control points

- knots
- controls



Uniform Nonrational B-splines

- The term uniform means that the knots are spaced at equal intervals of the parameter t .
- Then, $t_3=0$ and the interval $t_{i+1}-t_i=1$.
- The term nonrational is used to distinguish these splines from rational ones, discussed later.
- The "B" stands for Basis, since the splines can be represented as weighted sums of polynomial basis functions.

B-spline Geometry Vector

- Each of the $m-2$ curve segments is defined by four of the $m+1$ curve points.
- Thus, the B-spline geometry vector for curve segment Q_i is:

$$G_{Bs_i} = \begin{bmatrix} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{bmatrix}, \quad 3 \leq i \leq m$$

B-spline Definition

- Q_3 is defined by points P_0 through P_3 over the parameter range $t_3=0$ to $t_4=1$.
- Q_4 is defined by points P_1 through P_4 over the parameter range $t_4=1$ to $t_5=2$
- Q_m is defined by points P_{m-3} through P_m over the parameter range $t_m=m-3$ to $t_{m+1}=m-2$.
- In general, curve segment Q_i begins somewhere near point P_{i-2} and end somewhere near point P_{i-1} .

Local Control

- Each control point (except those at the start and end of the sequence) influences four curve segments.
- Moving a control point in a given direction moves the four curve segments it affects in the same direction.
- The other curve segments are totally unaffected.
- This is called local control and is the major advantage of B-splines over natural (interpolated) splines.

B-Spline formulation

- Let: $T_i = \begin{bmatrix} (t - t_i)^3 & (t - t_i)^2 & t - t_i & 1 \end{bmatrix}$
- Then the B-Spline formulation for curve segment i is:

$$Q_i(t) = T_i \cdot M_{Bs} \cdot G_{Bs_i}, t_i \leq t < t_{i+1}$$

- The entire curve is generated by applying this equation for each segment.

$$t_i \leq t < t_{i+1}$$

The B-spline basis matrix

- The B-spline basis matrix relates the geometrical constraints G_{Bs} to the blending functions and the polynomial coefficients:

$$M_{Bs} = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

- *We will not derive it here*

B-spline blending functions

- The B-spline blending functions are exactly the same for each curve segment:
 - For each segment i , the values of $t-t_i$ range from 0 at $t-t_i$, to 1 at $t-t_{i+1}$.

B-spline blending functions

- If we replace $t-t_i$ with t , and replace the interval $[t_i, t_{i+1}]$ with $[0, 1]$, then the B-spline blending functions B_{Bs} are given by:

$$B_{Bs} = T \cdot M_{Bs} = \begin{bmatrix} B_{Bs-3} & B_{Bs-2} & B_{Bs-1} & B_{Bs0} \end{bmatrix}$$

$$= \frac{1}{6} \begin{bmatrix} -t^3 + 3t^2 - 3t + 1 & 3t^3 - 6t^2 + 4 & -3t^2 + 3t^2 + 3t + 1 & t^3 \end{bmatrix}$$

$$= \frac{1}{6} \begin{bmatrix} (1-t)^3 & 3t^3 - 6t^2 + 4 & -3t^2 + 3t^2 + 3t + 1 & t^3 \end{bmatrix}, 0 \leq t < 1$$

B-splines

- So:

$$\begin{aligned} Q_i(t-t_i) &= G_{Bs_i} \cdot M_{Bs} \cdot T_i = G_{Bs_i} \cdot M_{Bs} \cdot T \\ &= G_{Bs_i} \cdot B_{Bs} = B_{Bs_{-3}} \cdot P_{i-3} + B_{Bs_{-2}} \cdot P_{i-2} + B_{Bs_{-1}} \cdot P_{i-1} + B_{Bs_0} \cdot P_i \\ &= \frac{(1-t)^3}{6} P_{i-3} + \frac{3t^3 - 6t^2 + 4}{6} P_{i-2} - \frac{3t^3 + 3t^2 + 3t + 1}{6} P_{i-1} + \frac{t^3}{6} P_i, \end{aligned}$$

$$0 \leq t < 1.$$

B-spline continuity

- We can show that Q_i and Q_{i+1} are C^0 , C^1 and C^2 continuous when they join:
 - Consider the x components of the adjacent segments (y and z are analogous).
 - We need only show that at the knot t_{i+1} where they join:

$$x_i(t_{i+1}) = x_{i+1}(t_{i+1}), \quad \frac{d}{dt} x_i(t_{i+1}) = \frac{d}{dt} x_{i+1}(t_{i+1})$$

and

$$\frac{d^2}{dt^2} x_i(t_{i+1}) = \frac{d^2}{dt^2} x_{i+1}(t_{i+1})$$

B-spline continuity

- The additional continuity afforded by B-splines is attractive, but it comes at the cost of less control of where the curve goes.
- We can force the curve to interpolate specific points by replicating control points $P_{i-2} = P_{i-1} = P_i$
 - E.g. if a control point is used 3 times, say:
 - Then: $Q_i(t) = P_{i-3} \cdot B_{Bs_{-3}} + P_i \cdot (B_{Bs_{-2}} + B_{Bs_{-1}} + B_{Bs_0})$
 - Curve segment Q_i is clearly a straight line.

Nonuniform, Nonrational B-splines

- The difference with nonuniform nonrational B-splines is that the parameter interval between successive knot values need not be uniform.
- The non-uniform knot-value sequence means that the blending functions are no longer the same for each interval, but vary from curve segment to curve segment.

Non-uniform splines

- The advantage of non-uniform splines is that the continuity at selected join points can be reduced right down to none, if desired.
- If the continuity is reduced to C^0 , then the curve interpolates a control point.
- This is without the undesirable effect of uniform B-splines, where the curve segments on either side of the interpolated control points are straight lines.
- In addition, starting and ending points can be easily interpolated exactly, without at the same time introducing linear segments.

Knot-value sequence

- We need a slightly different notation for non-uniform splines:
 - Again, the spline is a piecewise continuous curve made up of cubic polynomials, approximating the control points P_0 to P_m .
 - The knot-value sequence is a nondecreasing sequence of knot values t_0 to t_{m+4} (i.e 4 more knots than control points)

Knot Multiplicity

- The only restriction on the knot sequence is that it be nondecreasing, which allows successive knot values to be equal.
- When this occurs, it is called a multiple knot, and the number of identical parameter values is called the multiplicity of the knot.
 - E.g. in the knot sequence $(0,0,0,0,1,1,2,3,4,4)$, knot value 0 has multiplicity 4, values 1 and 4 have multiplicity 2, and 2 and 3 have multiplicity 1.

Curve segment definition

- Curve segment Q_i is defined by control points:

$$P_{i-3}, P_{i-2}, P_{i-1}, P_i$$

- and by blending functions:

$$B_{i-3,4}(t), B_{i-2,4}(t), B_{i-1,4}(t), B_{i,4}(t)$$

- As:

$$Q_i(t) = P_{i-3}B_{i-3,4}(t) + P_{i-2}B_{i-2,4}(t) + P_{i-1}B_{i-1,4}(t) + P_iB_{i,4}(t)$$

$$3 \leq i \leq m, t_i \leq t < t_{i+1}$$

Curve segment definition

- When $t_i=t_{i+1}$ (a multiple knot), curve segment Q_i is a single point.
- It is this idea of a curve segment reducing to a single point that provides the extra flexibility of non-uniform B-splines.
- There is no single set of blending functions, as they depend on intervals between knot values, and are defined recursively in terms of lower-order blending functions.

Non-uniform B-spline Blending functions

$$B_{i,1}(t) = \begin{cases} 1, & t_i \leq t < t_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

$$B_{i,2}(t) = \frac{t - t_i}{t_{i+1} - t_i} B_{i,1}(t) + \frac{t_{i+2} - t}{t_{i+2} - t_{i+1}} B_{i+1,1}(t)$$

$$B_{i,3}(t) = \frac{t - t_i}{t_{i+2} - t_i} B_{i,2}(t) + \frac{t_{i+3} - t}{t_{i+3} - t_{i+1}} B_{i+1,2}(t)$$

$$B_{i,4}(t) = \frac{t - t_i}{t_{i+3} - t_i} B_{i,3}(t) + \frac{t_{i+4} - t}{t_{i+4} - t_{i+1}} B_{i+1,3}(t)$$

Because we are dealing with cubic splines, the recursion ends at 4.

Rational Cubic Polynomial Curve Segments

- General rational cubic curve segments are ratios of polynomials:

$$x(t) = \frac{X(t)}{W(t)}, y(t) = \frac{Y(t)}{W(t)}, z(t) = \frac{Z(t)}{W(t)}$$

- Where $X(t)$, $Y(t)$, $Z(t)$ and $W(t)$ are all cubic polynomial curves whose control points are defined in homogeneous co-ordinates.

Rational Cubic Polynomial Curve Segments

- We can think of the curve as existing in homogeneous space as: $Q(t) = [X(t) \ Y(t) \ Z(t) \ W(t)]$
- Moving from homogeneous space to 3 space involves dividing by $W(t)$.
- We can transform any nonrational curve to a rational curve by adding $W(t)=1$ as a fourth element.

Non-Uniform Rational B-splines (NURBS)

- The polynomials in a rational curve can be Bezier, Hermite, or any other type.
- If they are non-uniform B-splines, they are called NURBS.
- The rational curves are useful because:
 - They are invariant under rotation, scaling, translation and perspective transformation of the control points.
 - The alternative to rational curves would be to generate points on the curve itself, and then transform them... a much less efficient process.

Parametric Bicubic Surfaces

- A generalization of parametric cubic curves.
- We have: $Q(s) = S \cdot M \cdot G$ where G the geometry vector is a constant, and s is the parameter.
- If we now allow the points in G to vary in 3D along some path, parameterized on t , then we have:

$$Q(s, t) = S \cdot M \cdot G(t) = S \cdot M \cdot \begin{bmatrix} G_1(t) \\ G_2(t) \\ G_3(t) \\ G_4(t) \end{bmatrix}$$

Parametric Bicubic Surface

- For fixed t_i , $Q(s, t_i)$ is a curve, because $G(t_i)$ is constant. For t_{i+1} , $Q(s, t_{i+1})$ is a different curve.
- Repeating this for many other values of t between 0 and 1, an entire family of curves is defined, each arbitrarily close to another curve.
- The set of all such curves defines a surface.
- If the $G_i(t)$ are themselves cubic polynomials, then the surface is a parametric bicubic surface.

Parametric Bicubic Surface

- If each $G_i(t)$ is a cubic polynomial, then they can be represented as: $G_i(t) = T \cdot M \cdot G_i$
- where: $G_i = [g_{i1} \quad g_{i2} \quad g_{i3} \quad g_{i4}]^T$
- Hence g_{i1} is the first control point for curve $G_i(t)$, and so on.

- Now, using the identity: $(A \cdot B \cdot C)^T = C^T \cdot B^T \cdot A^T$

- we have:
$$\begin{aligned} G_i(t) &= T \cdot M \cdot G_i \\ &= G_i^T \cdot M^T \cdot T^T \\ &= [g_{i1} \quad g_{i2} \quad g_{i3} \quad g_{i4}] \cdot M^T \cdot T^T \end{aligned}$$

- Giving:

$$Q(s, t) = S \cdot M \cdot \mathbf{G} \cdot M^T \cdot T^T, 0 \leq s, t \leq 1$$

Parametric Bicubic Surface Definition

- Expanding:

$$Q(s,t) = S \cdot M \cdot \begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \\ g_{41} & g_{42} & g_{43} & g_{44} \end{bmatrix} \cdot M^T \cdot T^T$$

- So:

$$x(s,t) = S \cdot M \cdot G_x \cdot M^T \cdot T^T$$

$$y(s,t) = S \cdot M \cdot G_y \cdot M^T \cdot T^T$$

$$z(s,t) = S \cdot M \cdot G_z \cdot M^T \cdot T^T$$

Hermite Surfaces

- Given the general form, now we look at ways to specify surfaces using different basis and geometry matrices.
- Hermite surfaces are completely defined by 4x4 geometry matrix G_H .
- Expanding just $x(s,t)$ we get:

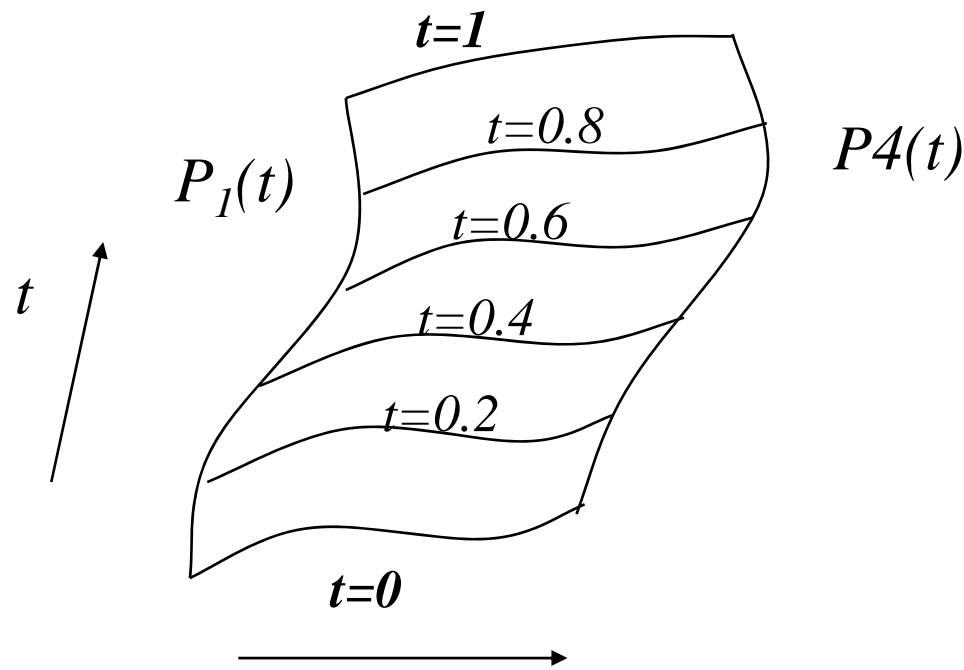
$$x(s,t) = S \cdot M_H \cdot G_{H_x}(t) = S \cdot M_H \cdot \begin{bmatrix} P_{1x}(t) \\ P_{4x}(t) \\ R_{1x}(t) \\ R_{4x}(t) \end{bmatrix}$$

Hermite Surfaces

- The functions $P_{1x}(t)$ and $P_{4x}(t)$ define the x components of the starting and ending points for the curve in parameter s .
- Similarly, $R_{1x}(t)$ and $R_{4x}(t)$ are the tangent vectors at these points.
- For any value of t , there are two specific endpoints and tangent vectors.

Hermite Surfaces

- This shows $P_1(t)$ and $P_4(t)$ and the cubic in s that is defined when $t=0.0, 0.2, 0.4, 0.6, 0.8$ and 1.0 .



- The surface patch is essentially a cubic interpolation between $F_1(t)=Q(0,t)$ and $F_4(t)=Q(1,t)$, or alternatively between $Q(s,0)$ and $Q(s,1)$.

Hermite Surfaces

- In the special case that the four interpolants $Q(0,t)$, $Q(1,t)$, $Q(s,0)$ and $Q(s,1)$ are straight lines, the result is a ruled surface.
- If the interpolants are coplanar, then the surface is a 4-sided planar polygon.
- We can derive $x(s,t)$ as before, (y and z follow, as usual):

$$x(s,t) = S \cdot M_H \cdot G_{H_x} \cdot M_H^T \cdot T^T$$

Bézier Surfaces

- The Bezier surfaces can be derived in exactly the same way, as:

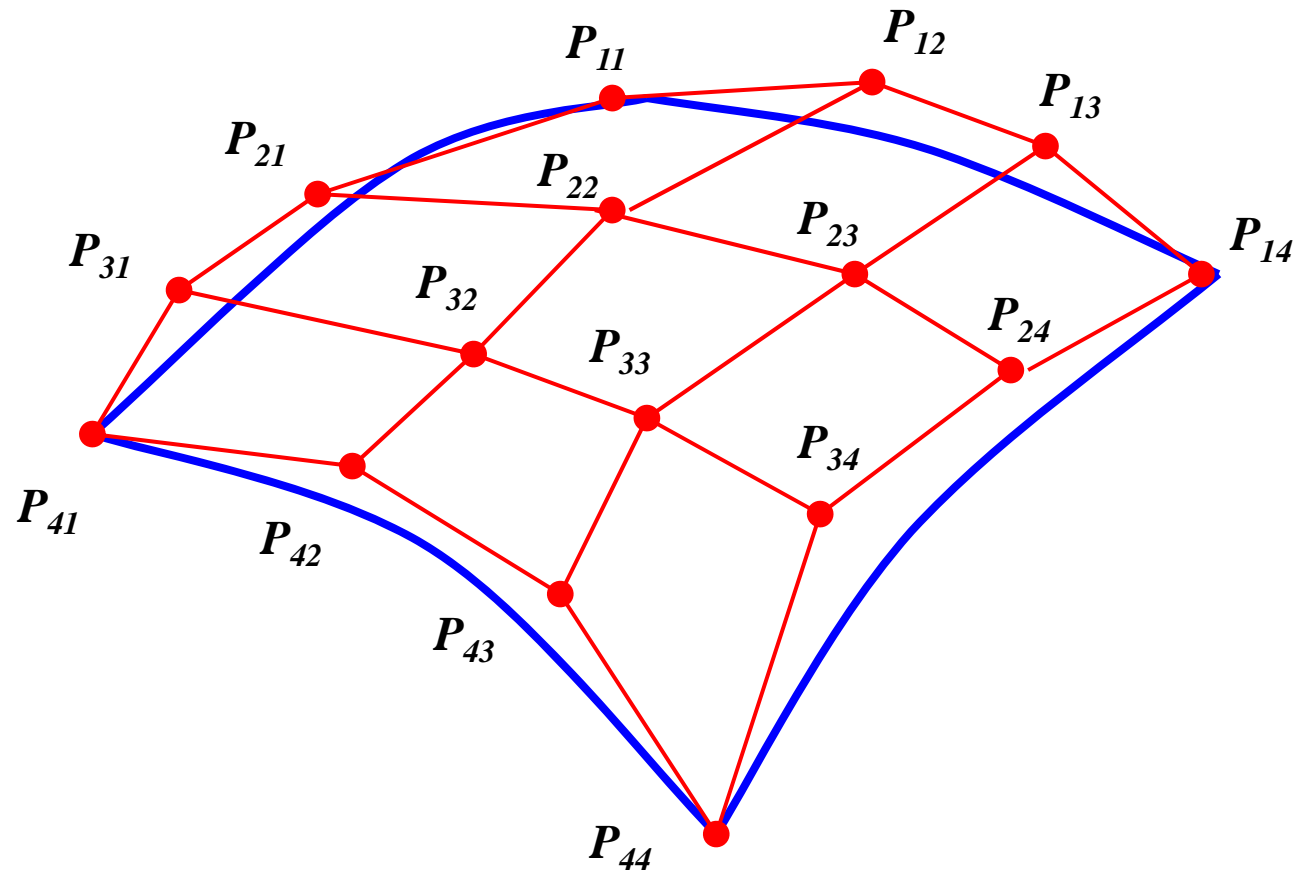
$$x(s, t) = S \cdot M_B \cdot G_{B_x} \cdot M_B^T \cdot T^T$$

$$y(s, t) = S \cdot M_B \cdot G_{B_y} \cdot M_B^T \cdot T^T$$

$$z(s, t) = S \cdot M_B \cdot G_{B_z} \cdot M_B^T \cdot T^T$$

- The Bézier geometry matrix G_B consists of 16 control points.

A Bézier Bicubic Patch



Bézier Patches

- These surfaces are attractive in interactive design.
- Some control points interpolate the surface, giving precise control, and the tangent vectors also can be controlled explicitly.
- Their convex-hull properties and easy subdivision are also attractive.

Rendering Curves and Surfaces

- One way of rendering a curve or surface is to compute intersections with rays from the eye through each pixel.
 - costly for real-time rendering
- Another approach is to evaluate the curve or surface at enough points to approximate it with standard flat objects (i.e. lines or polygons)
- Recursive subdivision techniques can also be used and are very efficient - good for adaptive rendering.

Curves and Surfaces in OpenGL

- OpenGL supports Bézier curves and surfaces through mechanisms called evaluators.
- These are used to compute values for the Bernstein polynomials of any order.
- Evaluators do not require uniform spacing of control points, and can be used to generate other types of polynomial curves and surfaces.

Evaluators in OpenGL

- The OpenGL evaluator functions allow you to use a polynomial mapping to produce vertices, normals, texture coordinates, and colors.
- These calculated values are then passed on to the processing pipeline as if they had been directly specified.
- The evaluator functions are also the basis for the NURBS (Non-Uniform Rational B-Spline) functions, which allow you to define curves and surfaces

Evaluators in OpenGL

- A one-dimensional evaluator is defined by:

```
glMap1f(type, u_min, u_max, stride, order, point_array)
```

- *target* defines the kind of values that are generated by the evaluator, e.g.
 - GL_MAP1_VERTEX_3: Each control point is three floating-point values representing *x*, *y*, and *z*.
 - GL_MAP1_NORMAL: Each control point is three floating-point values representing the *x*, *y*, and *z* components of a normal vector.
 - GL_MAP1_TEXTURE_COORD_X: Each control point holds the texture coordinates.

Evaluators in OpenGL

- $u1, u2$: defines the domain for parameter u .
- *Stride*: The number of floats or doubles between the beginning of one control point and the beginning of the next one in the data structure referenced in *points*. This allows control points to be embedded in arbitrary data structures. The only constraint is that the values for a particular control point must occupy contiguous memory locations.
- *Order*: The number of control points. Must be positive.
- *Points*: A pointer to the array of control points.

Bézier Curves in OpenGL

- For the one-dimensional evaluator:

```
glMap1f(type,u_min,u_max,stride,order,point_array)
```

- Example:

```
GLfloat pts[4][3] = { {-2.0, -2.0, -1.0},  
                      {-1.0,  2.0,  1.0},  
                      { 1.0, -2.0, -2.0},  
                      { 2.0,  2.0,  3.0} };  
glMap1f(GL_MAP1_VERTEX_3,0.0,1.0,3,4, &pts[0][0]);  
glEnable(GL_MAP1_VERTEX_3);    //enable evaluator
```

Bézier Curves in OpenGL

- Once an evaluator has been set up, we generate the values from the active evaluator as follows with `glEvalCoord1f(u)`:

```
glBegin(GL_LINE_STRIP);  
    for (i = 0; i <= NUM_STEPS; i++)  
        glEvalCoord1f((GLfloat)i/NUM_STEPS);  
glEnd();
```

- Alternatively, if the values of u are equally spaced, we can use:

```
glMapGrid1f(NUM_STEPS, 0, 1);  
glEvalMesh1(GL_LINE, 0, NUM_STEPS);
```

Bézier Surfaces in OpenGL

- Surfaces are generated in a manner similar to curves, using the functions `glMap2`, `glEvalCoord2`, `glMapGrid2f` and `glEvalMesh2` instead.
- For example, set it up with:

```
glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4,  
        0, 1, 12, 4, &ctrlpoints[0][0]);  
  
glEnable(GL_MAP2_VERTEX_3);
```

- then render with:

```
glMapGrid2f(8, 0.0, 1.0, 16, 0.0, 1.0);  
glEvalMesh2(GL_LINE, 0, NUM S STEPS, 0, NUM T STEPS);
```

NURBS functions

- Evaluators can be used to generate non-uniform spacing of points also.
- Any polynomial form can be converted to Bezier form by proper generation of control points.
- The OpenGL GLU library simplifies these steps by providing a set of NURBS functions.
- These allow finer control of the shape and rendering of the surface.