Computer Graphics I

Lecture 5:
Geometric modeling 1

Xiaopei LIU

School of Information Science and Technology
ShanghaiTech University

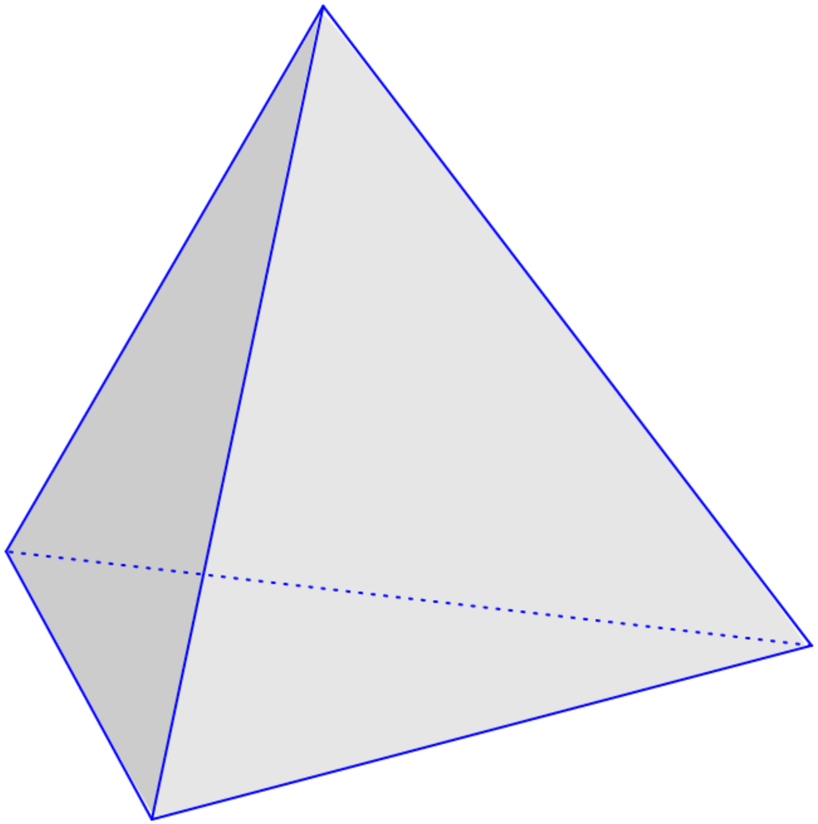
What is geometric modeling?

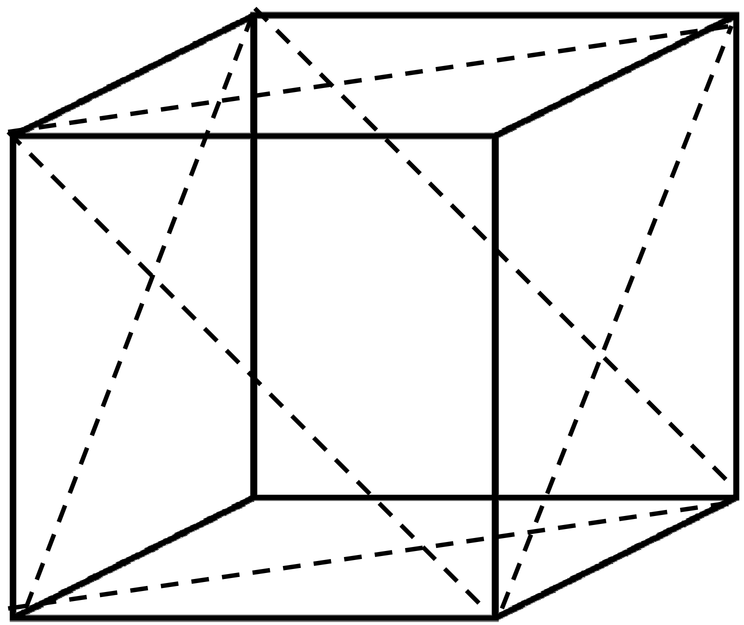
- **A branch of applied mathematics and computational geometry**
  - study methods and algorithms for the mathematical description of shapes
  - central to computer-aided design and manufacturing (CAD/CAM)
  - widely used in many applied technical fields such as civil and mechanical engineering, architecture, geology and medical image processing
  - an important area in computer graphics

1. Modeling for simple geometries

Tetrahedron and cubes

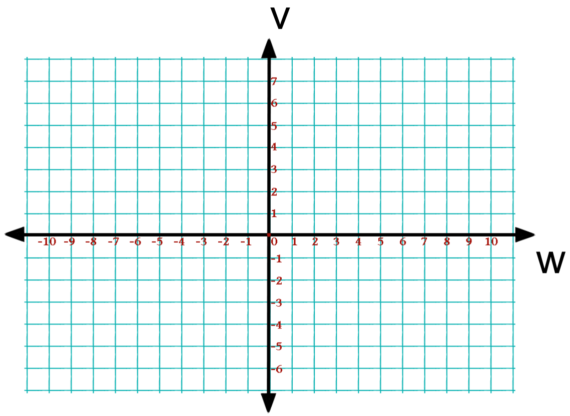
- **Created by a combination of triangles**

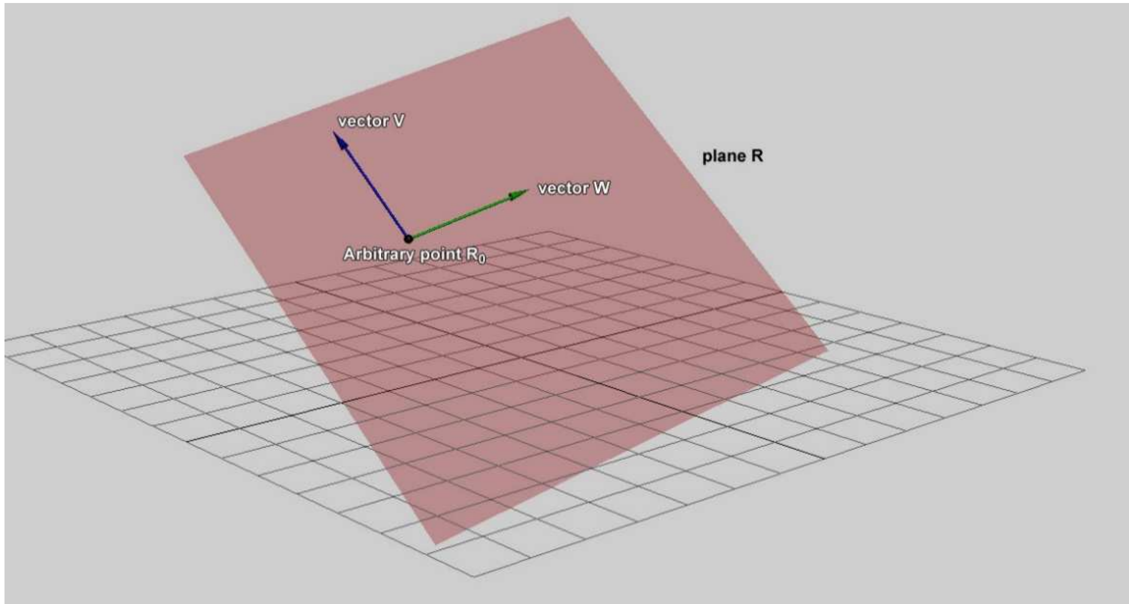




Plane

- **How to create a plane**
  - a large quadrilateral
  - or a set of tessellated triangles
  - How to create?
    - sample in 2D; translate and rotate to the desired state



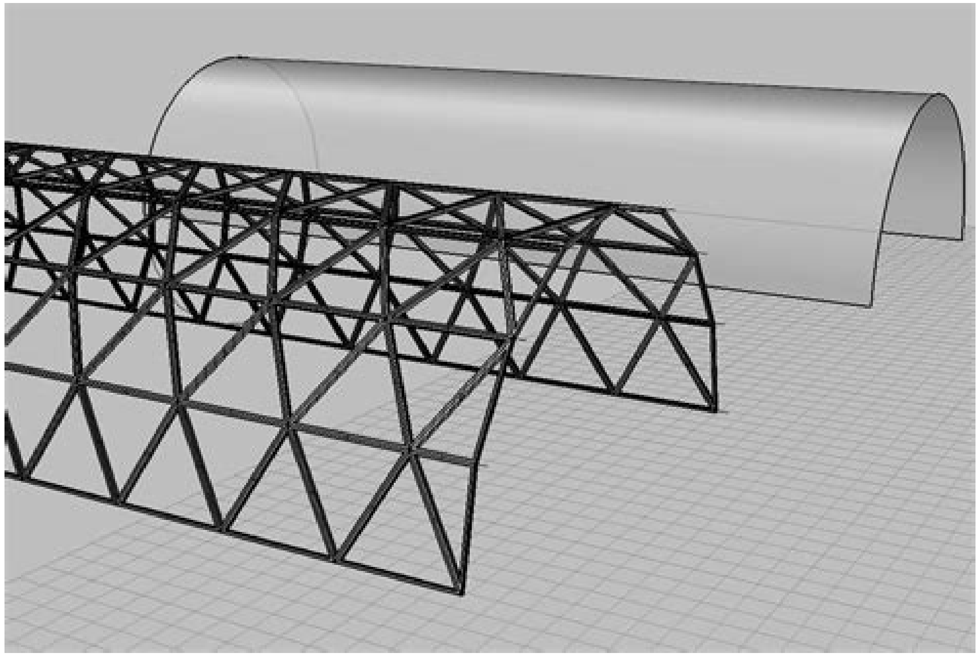


Cylinder

- **Representation of circles by parametric equations**
  - meshing in polar coordinates for x, y samples

$$x = a\cos(t)$$
$$y = a\sin(t)$$

  - sample in Z direction uniformly or staggered



Sphere

- **Analytical equations**
  - Cartesian coordinates:

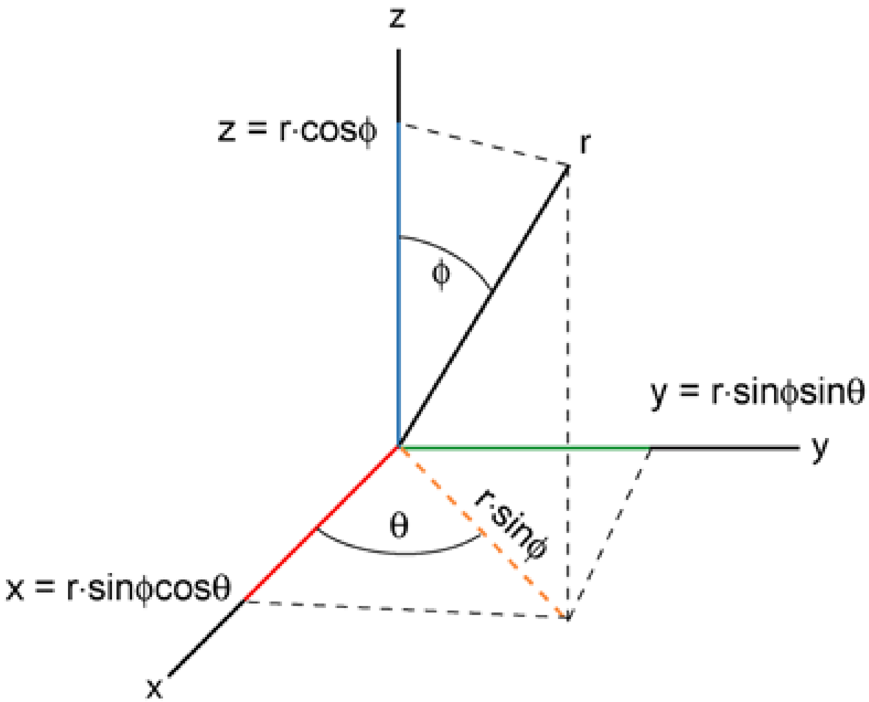
$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r^2$$

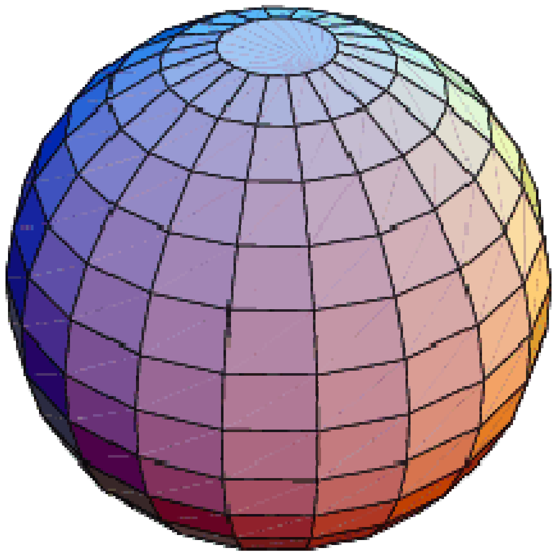
  - spherical coordinate parameterization:

$$\begin{aligned} x &= x_0 + r\cos\theta\ \sin\varphi \\ y &= y_0 + r\sin\theta\ \sin\varphi \qquad (0 \le \theta \le 2\pi \text{ and } 0 \le \varphi \le \pi) \\ z &= z_0 + r\cos\varphi \end{aligned}$$

Sphere mesh

- **Quadrilateral mesh**
  - meshing in spherical coordinates
  - uniformly subdivide θ and ϕ





Ellipsoid

- **Analytical equations**
  - Cartesian coordinates

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$$

  - spherical coordinate parameterization:

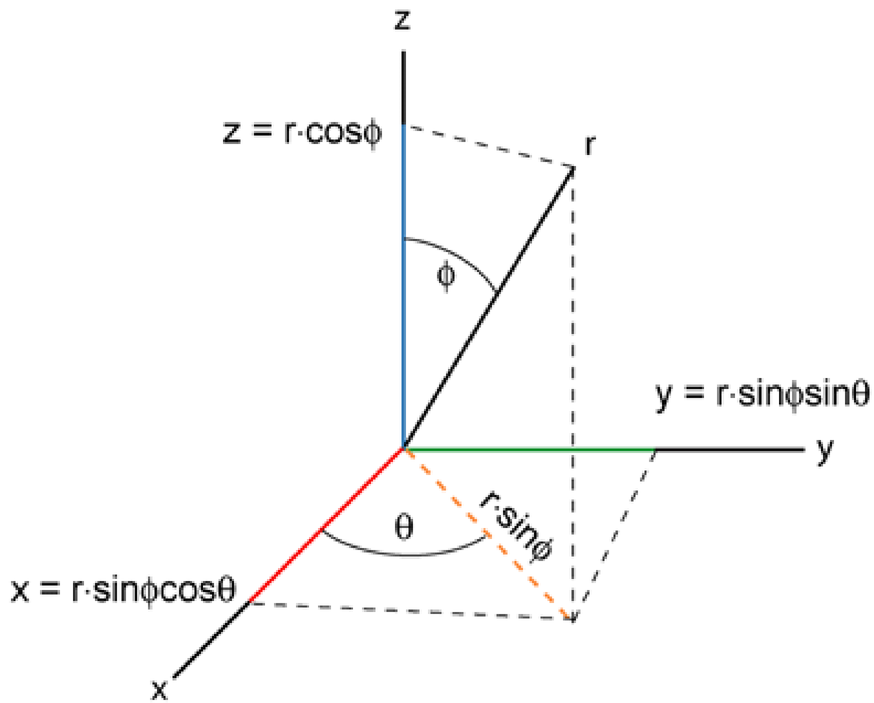
$$\begin{aligned} x &= a \ \cos(u)\cos(v), \\ y &= b \ \cos(u)\sin(v), \\ z &= c \ \sin(u), \end{aligned}$$

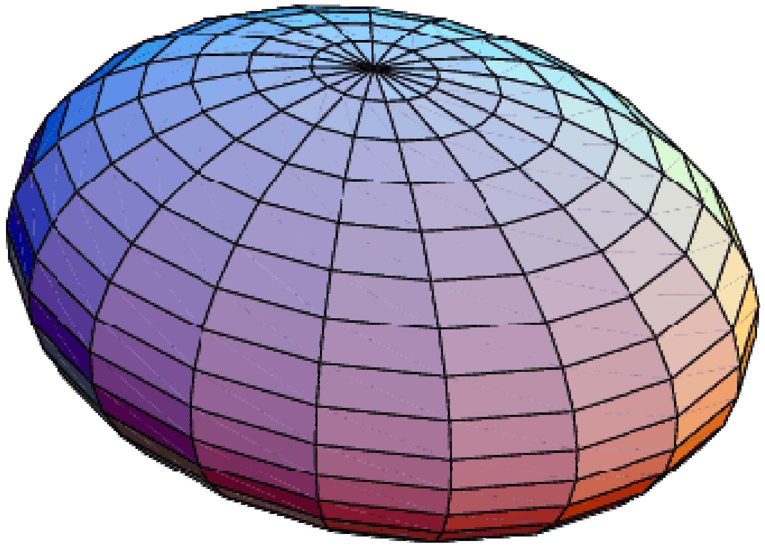
where

$$-\frac{\pi}{2} \le u \le \frac{\pi}{2}, \qquad -\pi \le v \le \pi.$$

Ellipsoid mesh

- **Quadrilateral mesh**
  - meshing in spherical coordinates
  - uniformly subdivide θ and ϕ





Cone

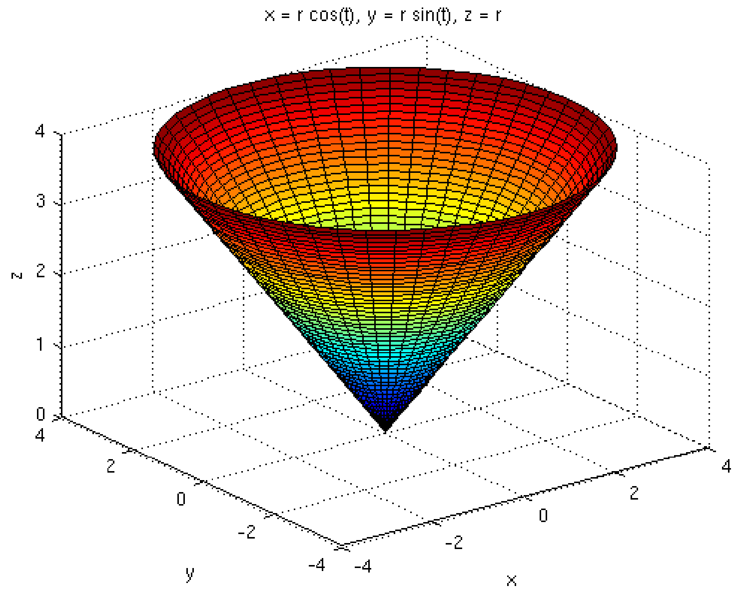
- **How to represent and meshing?**
  - general cone equation

$$\frac{x^2}{a^2} + \frac{y^2}{a^2} = z^2$$

  - meshing in polar coordinates for x, y samples:

$$x = a\cos(t)$$
$$y = a\sin(t)$$

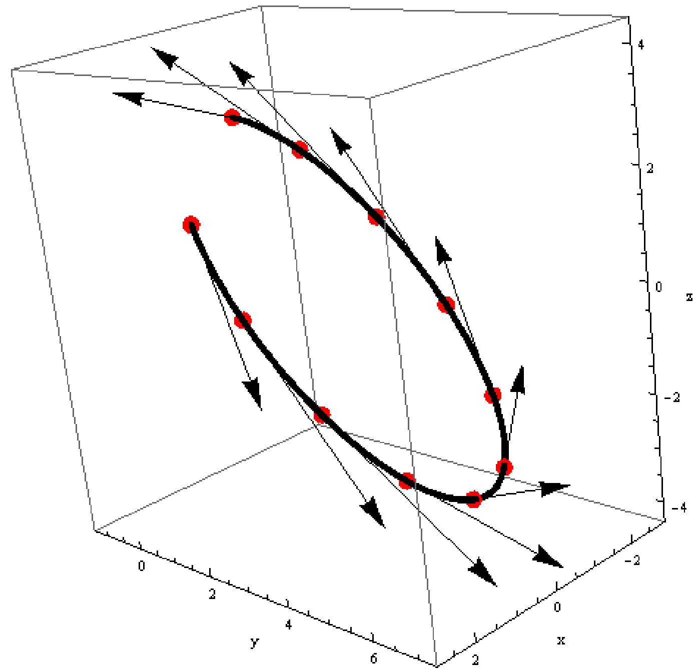
  - a=z/h, z in [o,h], h is the cone height



Tangent plane and normal computation

- **Parametric form of a curve**

$$x = X(u),\ \ y = Y(u),\ \ \ z = Z(u)$$



  - tangent vector

$$\boldsymbol{t} = \left[\frac{\partial X}{\partial u}, \frac{\partial Y}{\partial u}, \frac{\partial Z}{\partial u}\right]$$

  - normal vector

$$\boldsymbol{t} \cdot \boldsymbol{n} = 0$$

Tangent plane and normal computation

- **Parametric form of a surface**

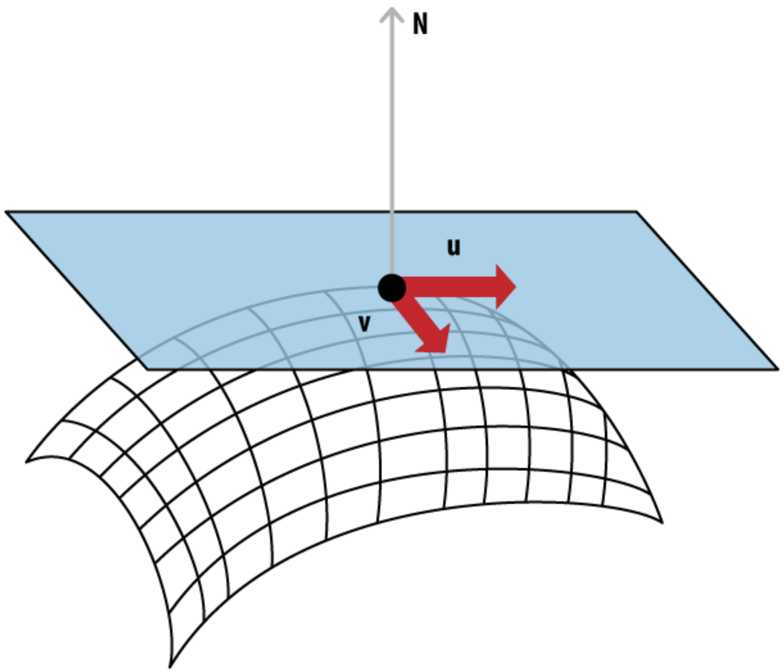
$$x = X(u, v), \quad y = Y(u, v), \quad z = Z(u, v)$$

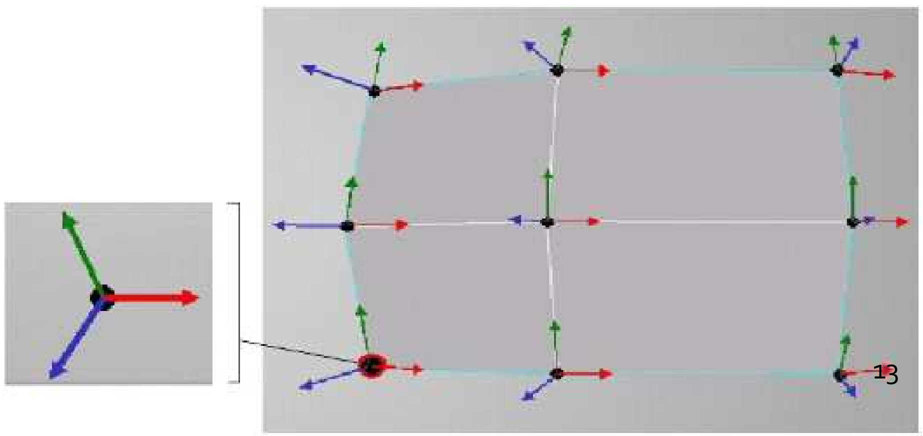
  - tangent vector

$$\boldsymbol{t}_u = \left[\frac{\partial X}{\partial u}, \frac{\partial Y}{\partial u}, \frac{\partial Z}{\partial u}\right] \qquad \boldsymbol{t}_v = \left[\frac{\partial X}{\partial v}, \frac{\partial Y}{\partial v}, \frac{\partial Z}{\partial v}\right]$$

  - normal vector

$$\boldsymbol{n} = \boldsymbol{t}_u \times \boldsymbol{t}_v$$

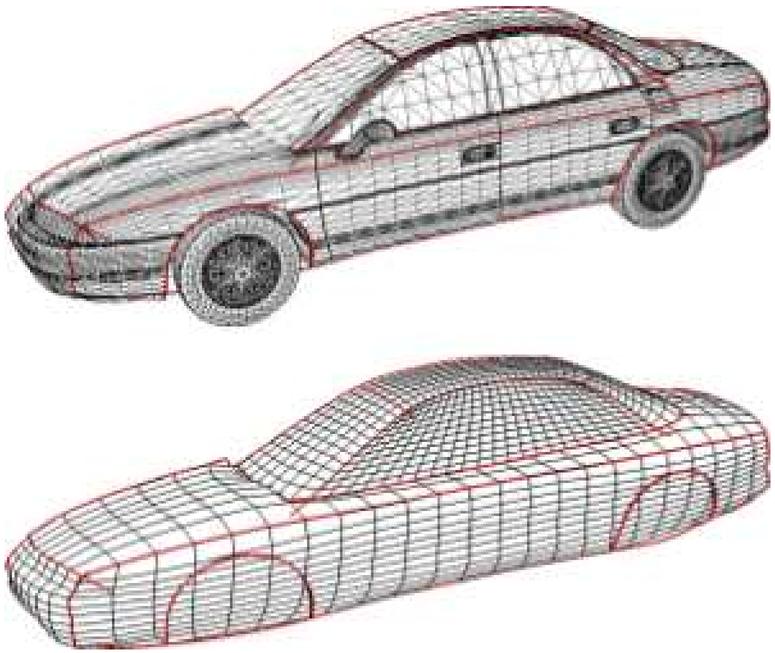




2. Free-form geometric modeling

Free-form surface modeling

- **Surfaces which do not have fixed shapes**
  - unlike regular surfaces such as planes , cylinders, spheres, and conic surfaces, etc.

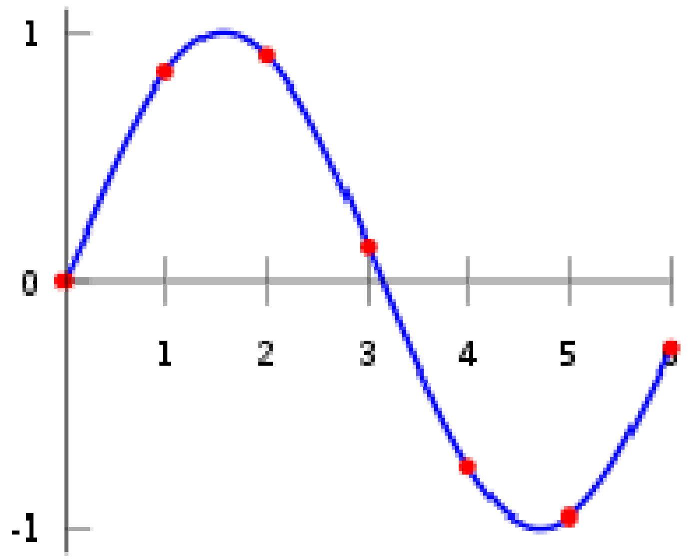


2.1. Polynomial interpolation

Polynomial interpolation

- Given a set of n + 1 data points (x_i, y_i) where no two x_i are the same, one is looking for a polynomial p of degree at most n with the property

$$p(x_i) = y_i, \qquad i = 0, \ldots, n.$$



Polynomial interpolation

- **Suppose that the interpolation polynomial is in the form:**

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0$$

$$p(x_i) = y_i \qquad \text{for all } i \in \{0, 1, \ldots, n\}$$

$$\begin{bmatrix} x_0^n & x_0^{n-1} & x_0^{n-2} & \ldots & x_0 & 1 \\ x_1^n & x_1^{n-1} & x_1^{n-2} & \ldots & x_1 & 1 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ x_n^n & x_n^{n-1} & x_n^{n-2} & \ldots & x_n & 1 \end{bmatrix} \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_0 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

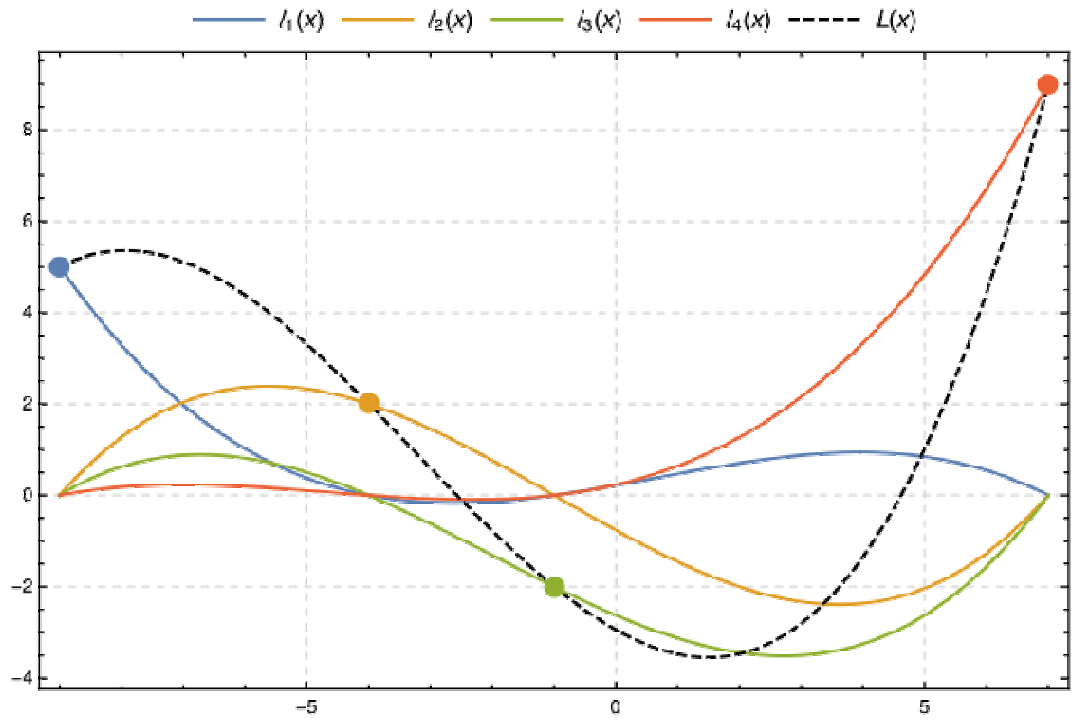
The condition number of the Vandermonde matrix may be large

Polynomial interpolation

- **Lagrange polynomials**

$$p(x) = \frac{(x-x_1)(x-x_2)\cdots(x-x_n)}{(x_0-x_1)(x_0-x_2)\cdots(x_0-x_n)}y_0 + \frac{(x-x_0)(x-x_2)\cdots(x-x_n)}{(x_1-x_0)(x_1-x_2)\cdots(x_1-x_n)}y_1 + \ldots + \frac{(x-x_0)(x-x_1)\cdots(x-x_{n-1})}{(x_n-x_0)(x_n-x_1)\cdots(x_n-x_{n-1})}y_n$$

$$= \sum_{i=0}^{n} \left(\prod_{\substack{0\le j\le n \\ j\ne i}} \frac{x-x_j}{x_i-x_j} \right) y_i$$



Degree of a polynomial

- **Degree of a monomial**
  - the sum of powers of all terms
  - the degree of $x^a y^b z^c$ is a+b+c

- **Highest degree of its monomials (individual terms) with non-zero coefficients**
  - the degree of polynomial

$$p(x,y)=w_1x^{a1}y^{b1}+ w_2x^{a2}y^{b2}+ \dots + w_nx^{an}y^{bn}$$

is

$$\max\{a1+b1, a2+b2, \dots, an+bn\}$$

for example: degree 5 polynomial for $7x^2y^3 + 4x - 9$

Hermite interpolation

- **Hermite interpolation matches an unknown function both in observed value, and the observed value of its first m derivatives**

$$
\begin{array}{llll}
(x_0, y_0), & (x_1, y_1), & \ldots, & (x_{n-1}, y_{n-1}), \\
(x_0, y'_0), & (x_1, y'_1), & \ldots, & (x_{n-1}, y'_{n-1}), \\
\vdots & \vdots & & \vdots \\
(x_0, y_0^{(m)}), & (x_1, y_1^{(m)}), & \ldots, & (x_{n-1}, y_{n-1}^{(m)})
\end{array}
$$

  – the resulting polynomial may have degree at most n(m+1)-1

Polynomial interpolation

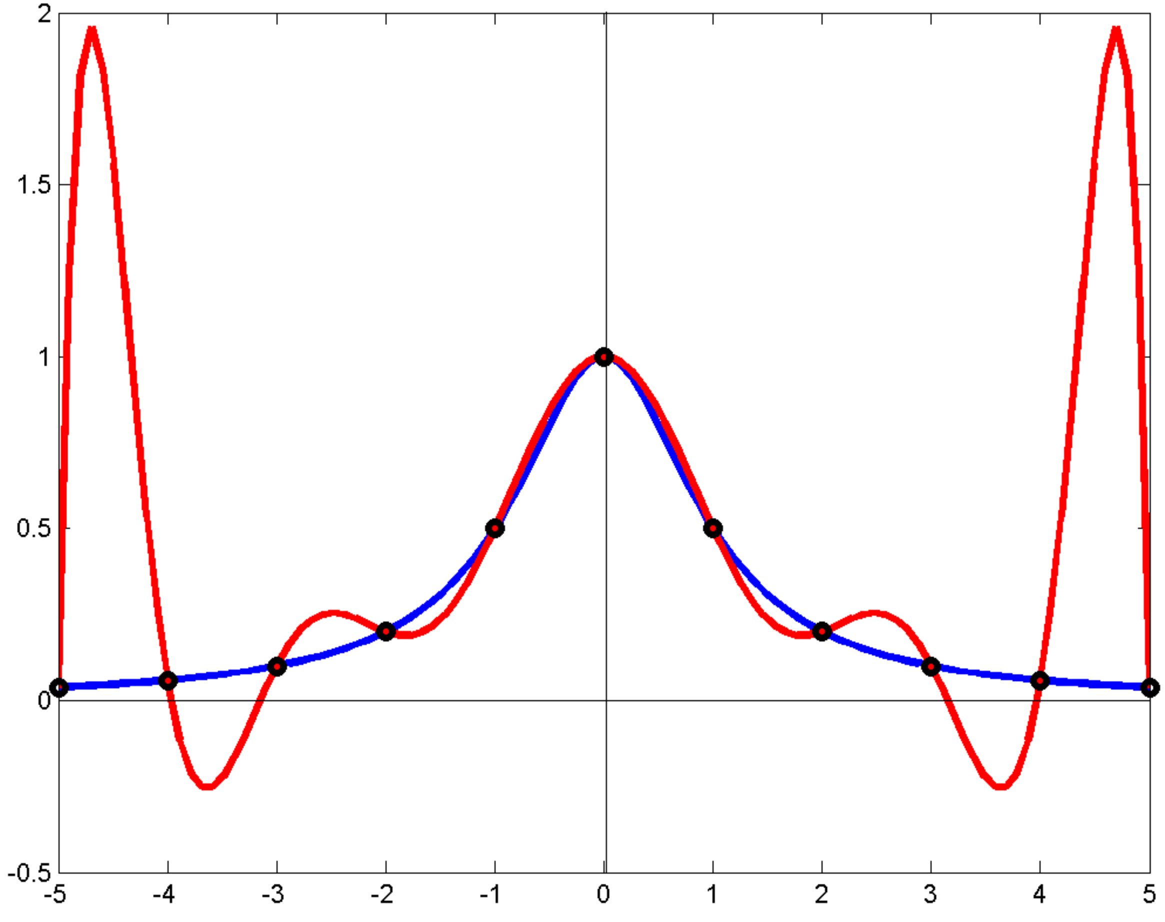
- **Basis functions**
  - An element of a particular basis for a function space
  - Each element is independent of other elements (think about the basis vector)
  - Basis function is also called blending function in numerical analysis and approximation theory
  - Every continuous function in the function space can be represented as a linear combination of the basis functions

$$f(\mathbf{x}) = \sum_i \omega_i \phi_i(\mathbf{x})$$

  - Function space: the space of functions that can be generated by basis functions with linear blending

Polynomial interpolation

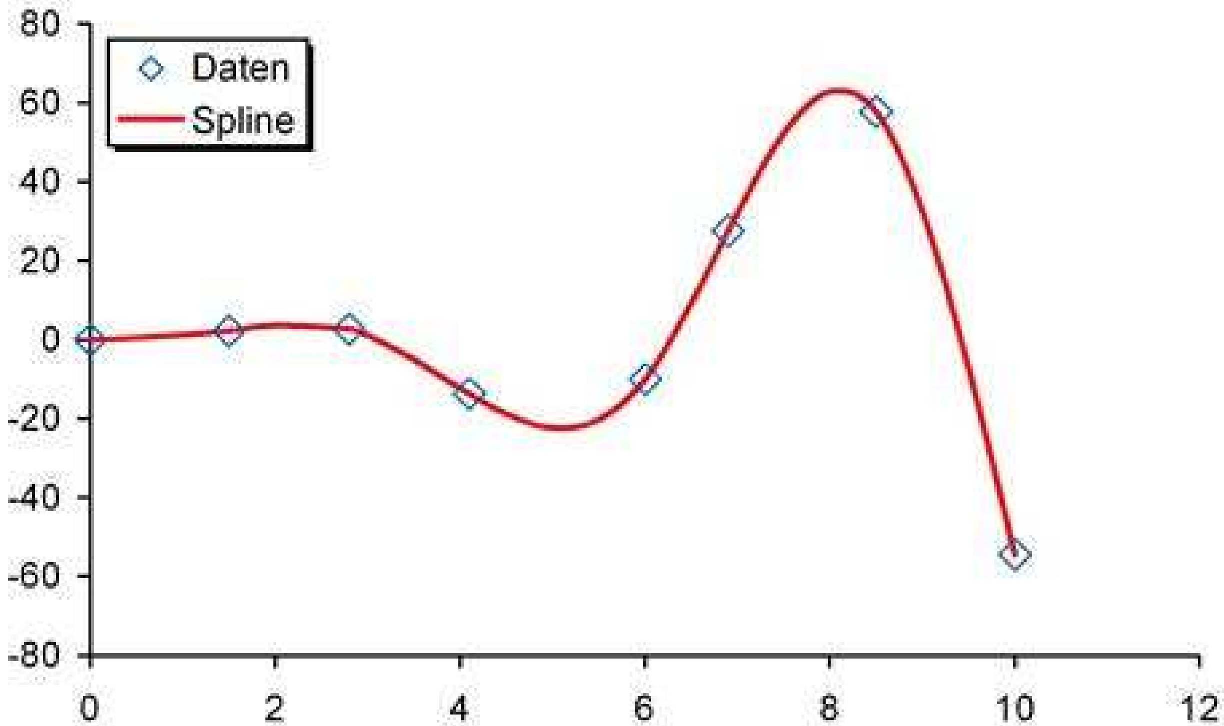
- **Runge phenomena**
  - a problem of oscillation in between the interpolation points
  - when using polynomial interpolation with high degree



2.2. Spline interpolation

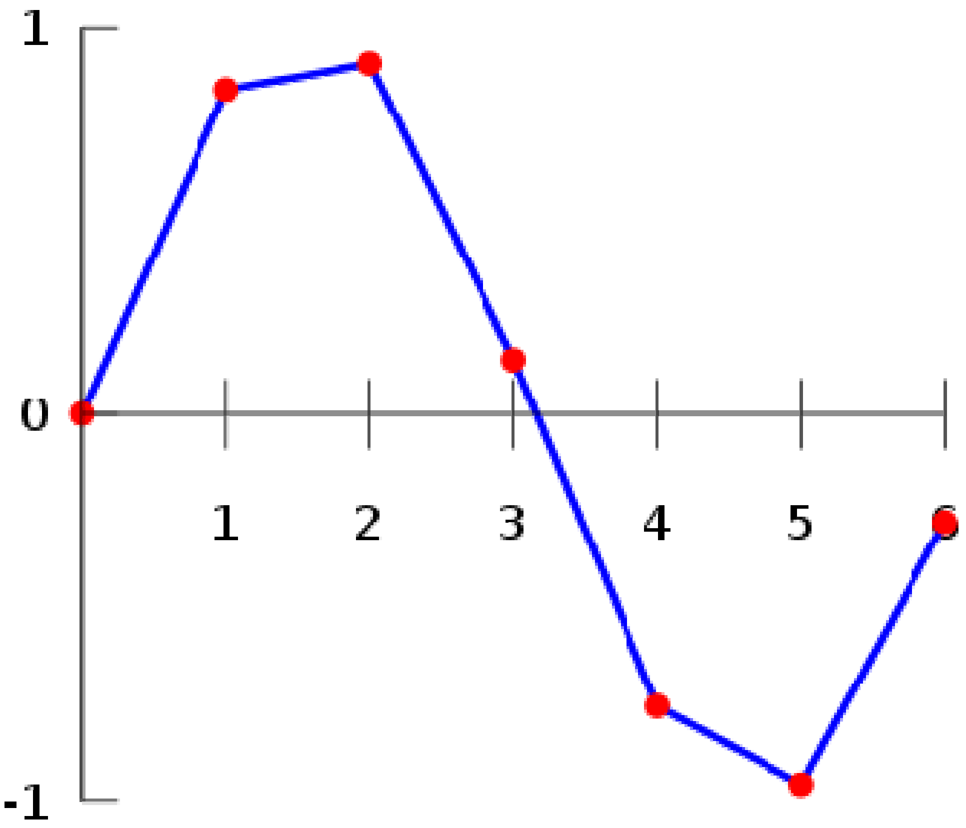
Spline

- **A spline is a special function defined piecewise by polynomials of low degree**
  - avoid Runge's phenomenon for more sample points
  - originally, high-degree polynomials should be used



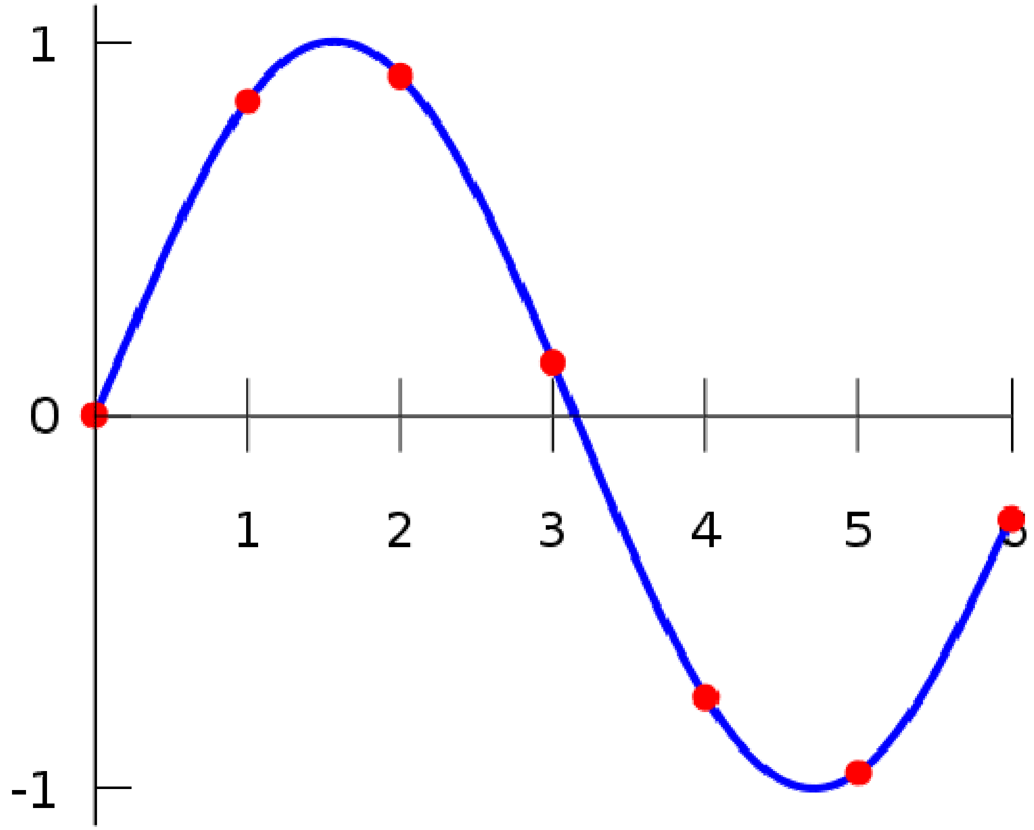
Spline

- **Piecewise linear spline**
  - the interpolation function is piecewise defined by linear functions (lines)



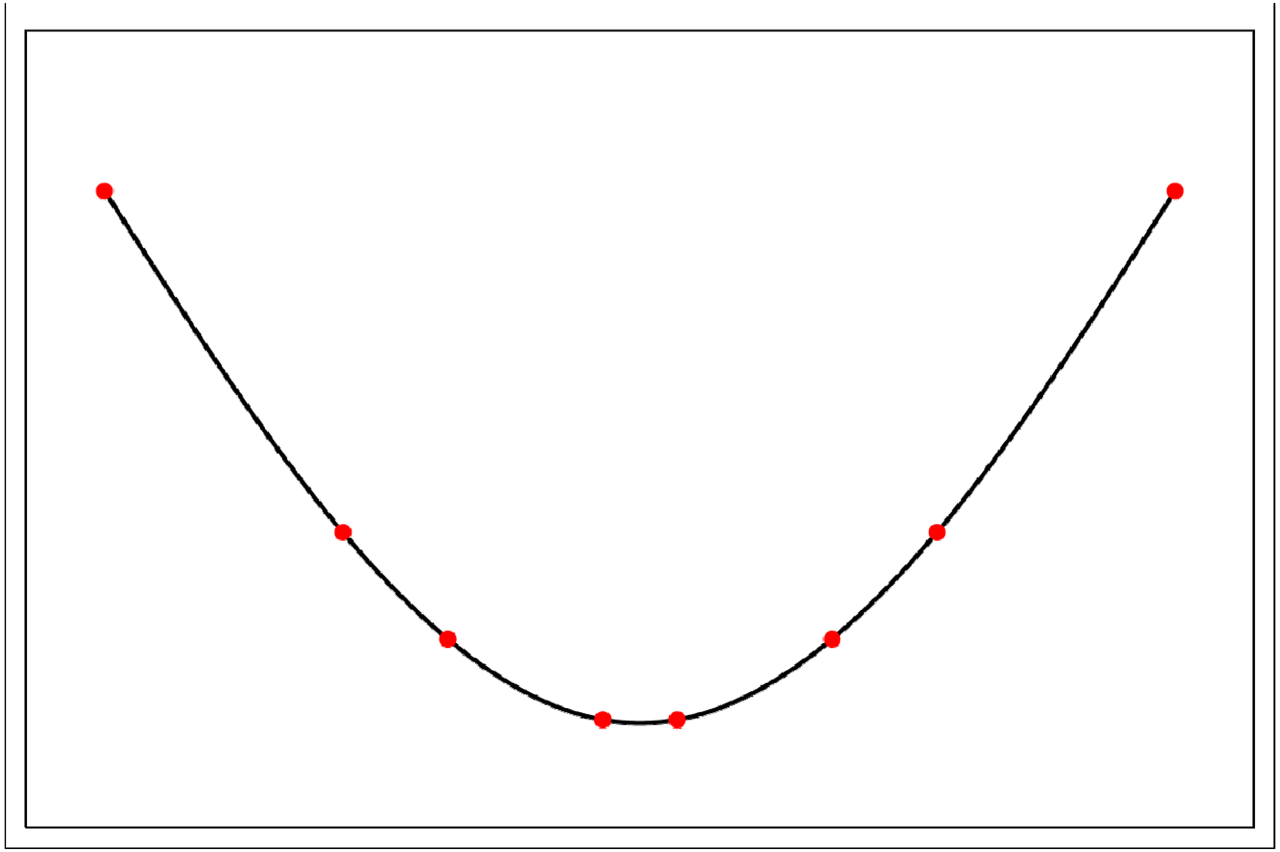
Spline

- **Piecewise quadratic spline**
  - the interpolation function is <u>piecewise</u> defined by quadratic polynomials



Spline

- **Piecewise cubic spline**
  - The interpolation function is <u>piecewise</u> defined by cubic polynomials



Cubic spline interpolation

- **A cubic polynomial**

$$p(x) = a + bx + cx^2 + dx^3$$

  - specified by 4 coefficients
  - twice continuously differentiable
  - has the flexibility to satisfy general types of boundary conditions
  - while the spline may agree with $f(x)$ at the nodes, we cannot guarantee that the derivatives of the spline agree with the derivatives of f

Cubic spline interpolation

- **Given a function f (x) defined on [a, b] and a set of nodes**

$$a = x_0 < x_1 < x_2 < \cdots < x_n = b,$$

- **A <u>cubic spline interpolant</u>, S, for f is a piecewise cubic polynomial, S_j on $[x_j ; x_{j+1}]$ for j = 0, 1, … , n -1**

$$S(x) = \begin{cases} a_0 + b_0(x - x_0) + c_0(x - x_0)^2 + d_0(x - x_0)^3 & \text{if } x_0 \le x \le x_1 \\ a_1 + b_1(x - x_1) + c_1(x - x_1)^2 + d_1(x - x_1)^3 & \text{if } x_1 \le x \le x_2 \\ \vdots & \vdots \\ a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 & \text{if } x_i \le x \le x_{i+1} \\ \vdots & \vdots \\ a_{n-1} + b_{n-1}(x - x_{n-1}) + c_{n-1}(x - x_{n-1})^2 + d_{n-1}(x - x_{n-1})^3 & \text{if } x_{n-1} \le x \le x_n \end{cases}$$

Cubic spline interpolation

- **The cubic spline interpolant will have the following properties:**

  - $S(x_j) = f(x_j)$ for $j = 0, 1, \ldots, n$.
  - $S_j(x_{j+1}) = S_{j+1}(x_{j+1})$ for $j = 0, 1, \ldots, n-2$.
  - $S_j'(x_{j+1}) = S_{j+1}'(x_{j+1})$ for $j = 0, 1, \ldots, n-2$.
  - $S_j''(x_{j+1}) = S_{j+1}''(x_{j+1})$ for $j = 0, 1, \ldots, n-2$.
  - One of the following boundary conditions (BCs) is satisfied:
    - $S''(x_0) = S''(x_n) = 0$ (**free** or **natural** BCs).
    - $S'(x_0) = f'(x_0)$ and $S'(x_n) = f'(x_n)$ (**clamped** BCs).

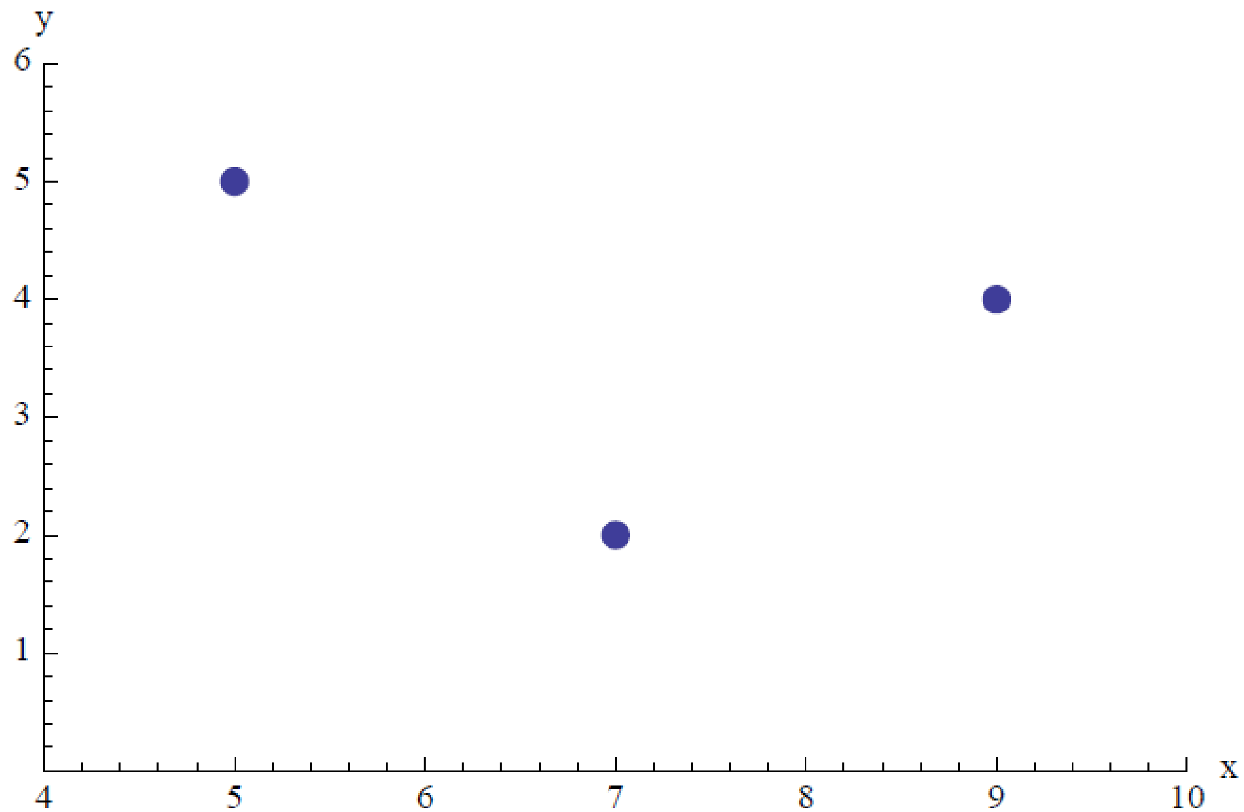
Cubic spline interpolation

- **Example**

Construct a piecewise cubic spline interpolant for the curve passing through

$$\{(5,5),\ (7,2),\ (9,4)\},$$

with natural boundary conditions.



Cubic spline interpolation

- **This will require two cubics:**

$$S_0(x) = a_0 + b_0(x-5) + c_0(x-5)^2 + d_0(x-5)^3$$
$$S_1(x) = a_1 + b_1(x-7) + c_1(x-7)^2 + d_1(x-7)^3$$

  - Since there are 8 coefficients, we must derive 8 equations to solve.
  - The splines must agree with the function (the y-coordinates) at the nodes (the x-coordinates)

$$\begin{aligned} 5 &= S_0(5) = a_0 \\ 2 &= S_0(7) = a_0 + 2b_0 + 4c_0 + 8d_0 \\ 2 &= S_1(7) = a_1 \\ 4 &= S_1(9) = a_1 + 2b_1 + 4c_1 + 8d_1 \end{aligned}$$

Cubic spline interpolation

- The first and second derivatives of the cubics must agree at their shared node x = 7:

$$\begin{aligned} S_0'(7) = b_0 + 4c_0 + 12d_0 &= b_1 = S_1'(7) \\ S_0''(7) = 2c_0 + 12d_0 = 2c_1 &= S_1''(7) \end{aligned}$$

- The final two equations come from the natural boundary conditions:

$$\begin{aligned} S_0''(5) &= 0 = 2c_0 \\ S_1''(9) &= 0 = 2c_1 + 12d_1 \end{aligned}$$

Cubic spline interpolation

- **Solving a linear equation system**
  - all eight linear equations together form the system
  - note that the system is generally sparse

$$
\begin{aligned}
5 &= a_0 \\
2 &= a_0 + 2b_0 + 4c_0 + 8d_0 \\
2 &= a_1 \\
4 &= a_1 + 2b_1 + 4c_1 + 8d_1 \\
0 &= b_0 + 4c_0 + 12d_0 - b_1 \\
0 &= 2c_0 + 12d_0 - 2c_1 \\
0 &= 2c_0 \\
0 &= 2c_1 + 12d_1
\end{aligned}
$$

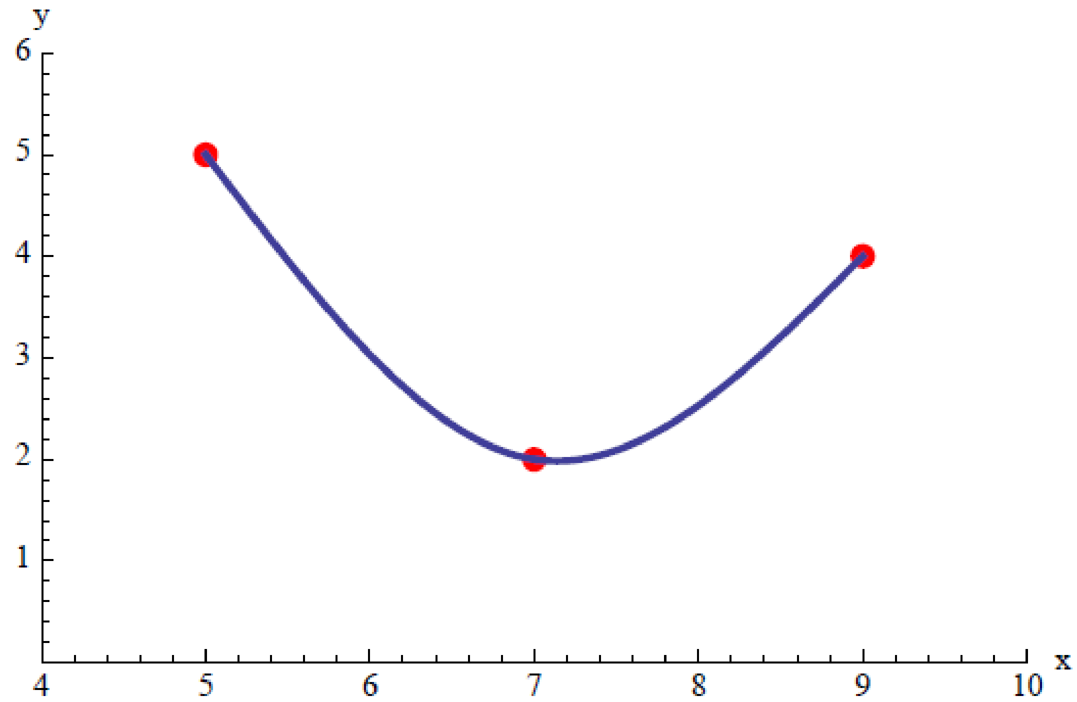
$\Rightarrow$

i	a_i	b_i	c_i	d_i
0	5	$-\frac{17}{8}$	0	$\frac{5}{32}$
1	2	$-\frac{1}{4}$	$\frac{15}{16}$	$-\frac{5}{32}$

Cubic spline interpolation

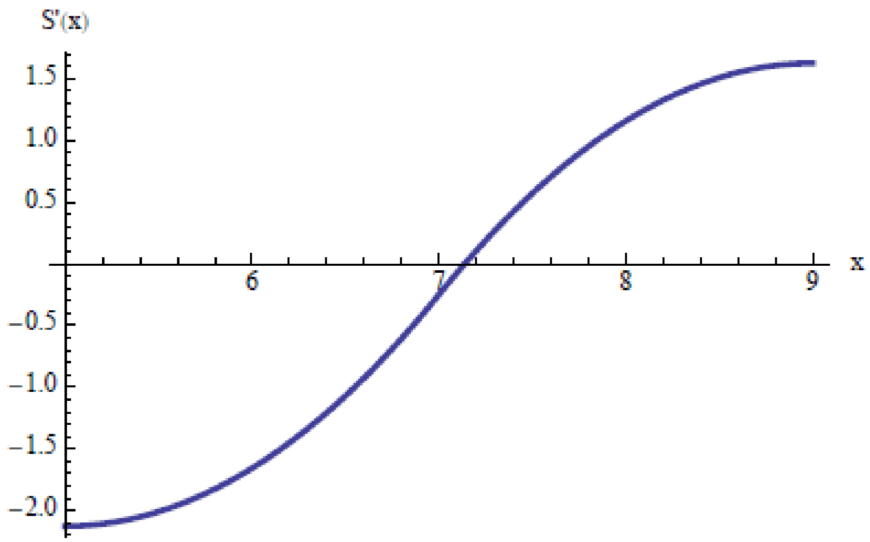
- **The natural cubic spline can be expressed as:**

$$S(x) = \begin{cases} 5 - \frac{17}{8}(x-5) + \frac{5}{32}(x-5)^3 & \text{if } 5 \le x \le 7 \\ 2 - \frac{1}{4}(x-7) + \frac{15}{16}(x-7)^2 - \frac{5}{32}(x-7)^3 & \text{if } 7 \le x \le 9 \end{cases}$$

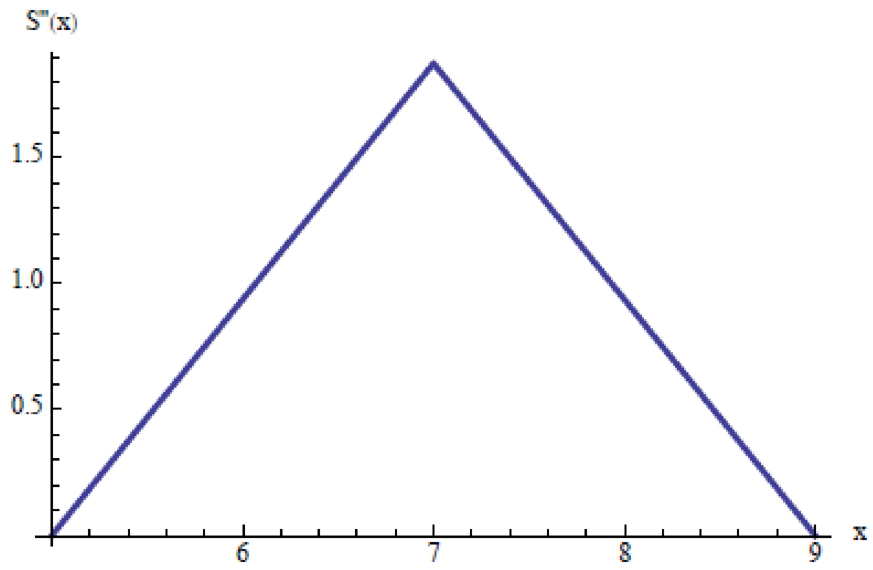


Cubic spline interpolation

- **We can verify the continuity of the first and second derivatives from the following graphs**



First derivative



Second derivative

Cubic spline interpolation

- **General construction process**

Given $n+1$ nodal/data values:
$\{(x_0, f(x_0)), (x_1, f(x_1)), \ldots, (x_n, f(x_n))\}$ we will create n cubic polynomials.

For $j = 0, 1, \ldots, n-1$ assume

$$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3.$$

We must find a_j, b_j, c_j and d_j (a total of $4n$ unknowns) subject to the conditions specified in the definition.

Cubic spline interpolation

- **Redefinition of equations**

Let $h_j = x_{j+1} - x_j$ then

$$\begin{aligned} S_j(x_j) &= a_j = f(x_j) \\ S_{j+1}(x_{j+1}) &= a_{j+1} = S_j(x_{j+1}) = a_j + b_j h_j + c_j h_j^2 + d_j h_j^3. \end{aligned}$$

So far we know a_j for $j = 0, 1, \ldots, n-1$ and have n equations and $3n$ unknowns.

$$\begin{aligned} a_1 &= a_0 + b_0 h_0 + c_0 h_0^2 + d_0 h_0^3 \\ &\vdots \\ a_{j+1} &= a_j + b_j h_j + c_j h_j^2 + d_j h_j^3 \\ &\vdots \\ a_n &= a_{n-1} + b_{n-1} h_{n-1} + c_{n-1} h_{n-1}^2 + d_{n-1} h_{n-1}^3 \end{aligned}$$

Cubic spline interpolation

- **First derivative relations**
  - The continuity of the first derivative at the nodal points produces n more equations

For $j = 0, 1, \ldots, n-1$ we have

$$S_j'(x) = b_j + 2c_j(x - x_j) + 3d_j(x - x_j)^2.$$

Thus

$$\begin{aligned} S_j'(x_j) &= b_j \\ S_{j+1}'(x_{j+1}) &= b_{j+1} = S_j'(x_{j+1}) = b_j + 2c_j h_j + 3d_j h_j^2 \end{aligned}$$

Now we have $2n$ equations and $3n$ unknowns.

Cubic spline interpolation

- **Equations derived so far**

$$
\begin{aligned}
a_{j+1} &= a_j + b_j h_j + c_j h_j^2 + d_j h_j^3 \quad (\text{for } j = 0, 1, \ldots, n-1) \\
b_1 &= b_0 + 2c_0 h_0 + 3d_0 h_0^2 \\
&\vdots \\
b_{j+1} &= b_j + 2c_j h_j + 3d_j h_j^2 \\
&\vdots \\
b_n &= b_{n-1} + 2c_{n-1} h_{n-1} + 3d_{n-1} h_{n-1}^2
\end{aligned}
$$

The unknowns are b_j, c_j, and d_j for $j = 0, 1, \ldots, n-1$.

Cubic spline interpolation

- **Second derivative relations**
  - The continuity of the second derivative at the nodal points produces n more equations

For $j = 0, 1, \ldots, n-1$ we have

$$S_j''(x) = 2c_j + 6d_j(x - x_j).$$

Thus

$$\begin{aligned} S_j''(x_j) &= 2c_j \\ S_{j+1}''(x_{j+1}) &= 2c_{j+1} = S_j''(x_{j+1}) = 2c_j + 6d_jh_j \end{aligned}$$

Now we have $3n$ equations and $3n$ unknowns.

Cubic spline interpolation

- **Summary of equations**

$$
\begin{aligned}
a_{j+1} &= a_j + b_j h_j + c_j h_j^2 + d_j h_j^3 \\
b_{j+1} &= b_j + 2c_j h_j + 3d_j h_j^2 \\
c_{j+1} &= c_j + 3d_j h_j .
\end{aligned}
$$

Note: The quantities a_j and h_j are known.

Solve the third equation for d_j and substitute into the other two equations.

$$
d_j = \frac{c_{j+1} - c_j}{3h_j}
$$

Cubic spline interpolation

- **Substitution**

$$d_j = \frac{c_{j+1} - c_j}{3h_j}$$

$$\begin{aligned}
a_{j+1} &= a_j + b_j h_j + c_j h_j^2 + \left(\frac{c_{j+1} - c_j}{3h_j}\right) h_j^3 \\
&= a_j + b_j h_j + \frac{h_j^2}{3}(2c_j + c_{j+1}) \\
b_{j+1} &= b_j + 2c_j h_j + 3\left(\frac{c_{j+1} - c_j}{3h_j}\right) h_j^2 \\
&= b_j + h_j(c_j + c_{j+1})
\end{aligned}$$

Solve the first equation for b_j.

$$b_j = \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2c_j + c_{j+1})$$

Cubic spline interpolation

- **Substitution**

Replace index j by $j-1$ to obtain

$$b_{j-1} = \frac{1}{h_{j-1}}(a_j - a_{j-1}) - \frac{h_{j-1}}{3}(2c_{j-1} + c_j).$$

We can also re-index the earlier equation

$$b_{j+1} = b_j + h_j(c_j + c_{j+1})$$

to obtain

$$b_j = b_{j-1} + h_{j-1}(c_{j-1} + c_j).$$

Substitute the equations for b_{j-1} and b_j into the remaining equation. This step eliminate n equations of the first type.

Cubic spline interpolation

- **Substitution**

$$\frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2c_j + c_{j+1})$$
$$= \quad \frac{1}{h_{j-1}}(a_j - a_{j-1}) - \frac{h_{j-1}}{3}(2c_{j-1} + c_j) + h_{j-1}(c_{j-1} + c_j)$$

Collect all terms involving c to one side.

$$h_{j-1}c_{j-1} + 2c_j(h_{j-1} + h_j) + h_j c_{j+1} = \frac{3}{h_j}(a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1})$$

for $j = 1, 2, \ldots, n-1$.

Remark: we have $n-1$ equations and $n+1$ unknowns.

If $S''(x_0) = S_0''(x_0) = 2c_0 = 0$ then $c_0 = 0$ and if
$S''(x_n) = S_{n-1}''(x_n) = 2c_n = 0$ then $c_n = 0$.

Cubic spline interpolation

- **In matrix form, the system of n + 1 equations has the form *A*c = y where**

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ h_0 & 2(h_0 + h_1) & h_1 & 0 & \cdots & 0 \\ 0 & h_1 & 2(h_1 + h_2) & h_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} 0 \\ \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \frac{3}{h_2}(a_3 - a_2) - \frac{3}{h_1}(a_2 - a_1) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \\ 0 \end{bmatrix}$$

Note: ***A*** is a tridiagonal matrix

Cubic spline interpolation

- **Solve the linear equation system**

$$A \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \\ c_n \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \frac{3}{h_2}(a_3 - a_2) - \frac{3}{h_1}(a_2 - a_1) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \\ 0 \end{bmatrix}$$

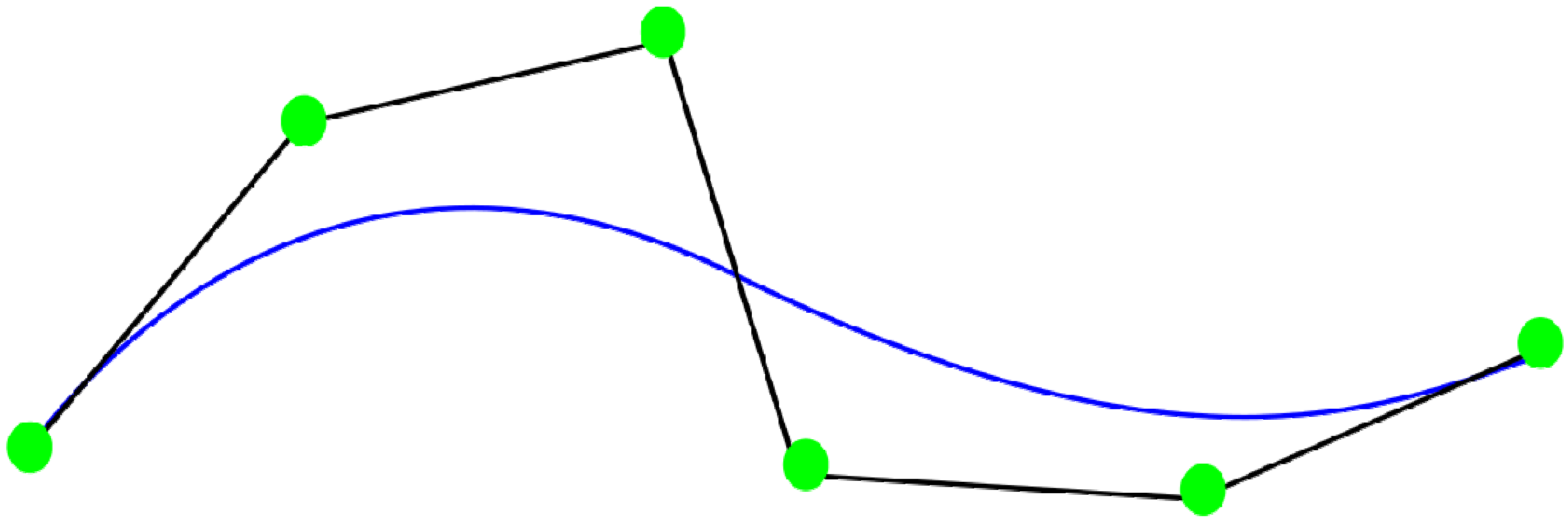
We solve this linear system of equations using a common algorithm for handling tridiagonal systems

Note that such a process is not quite computationally efficient if the number of sample points is large!!

2.3. Approximating polynomials

Polynomial approximation

- **The polynomials are generated by control points**
  - the curve does not necessarily pass through control points
  - control points are used to control the shape of the curve



2.3.1. Bézier curve

Bernstein polynomial

- **Bernstein polynomial**
  - the $n + 1$ Bernstein basis polynomials of degree n are defined as:

$$b_{\nu,n}(x) = \binom{n}{\nu} x^{\nu} (1 - x)^{n-\nu}, \quad \nu = 0, \ldots, n.$$

  - A linear combination of Bernstein basis polynomials:

$$B_n(x) = \sum_{\nu=0}^{n} \beta_{\nu} b_{\nu,n}(x)$$

Bernstein polynomial

- **The first a few Bernstein basis polynomials are:**

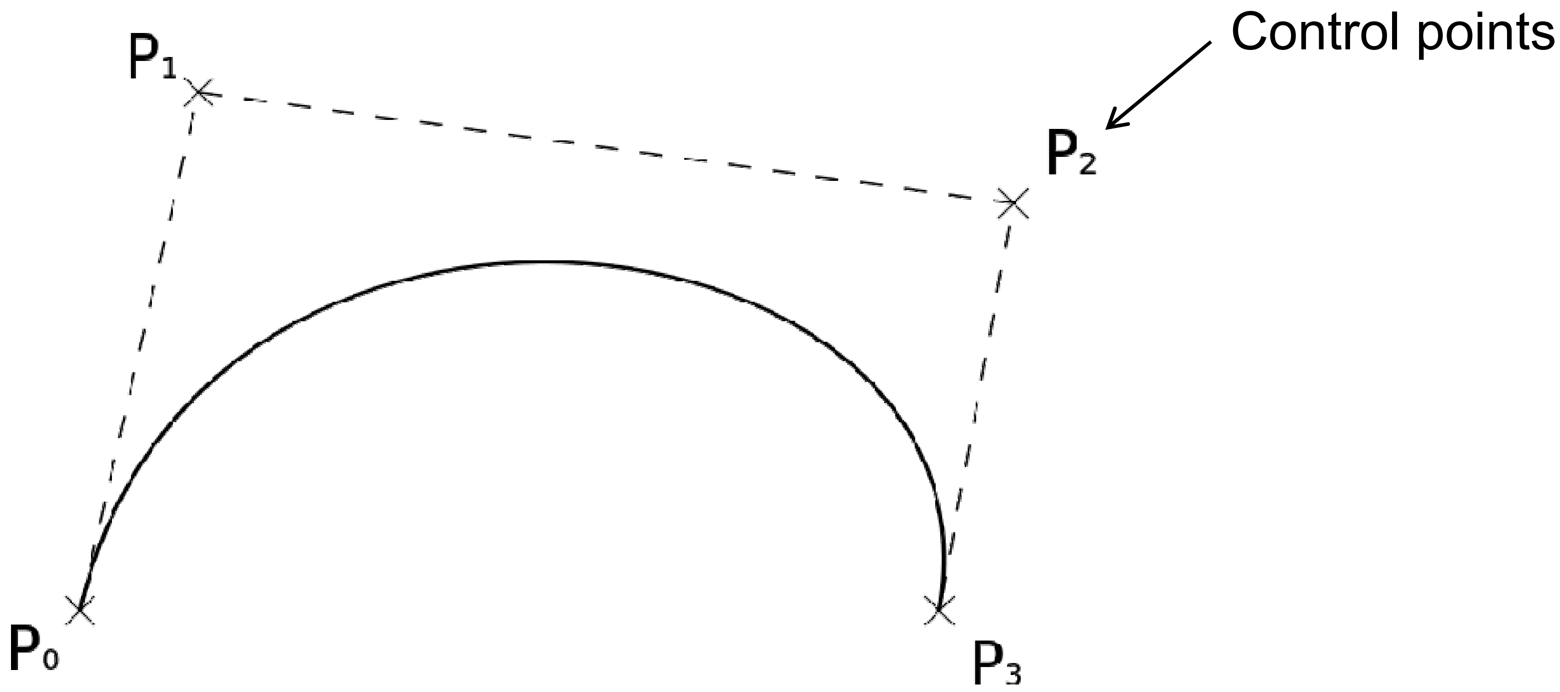
$$
\begin{aligned}
&b_{0,0}(x) = 1, \\
&b_{0,1}(x) = 1 - x, && b_{1,1}(x) = x \\
&b_{0,2}(x) = (1 - x)^2, && b_{1,2}(x) = 2x(1 - x), && b_{2,2}(x) = x^2 \\
&b_{0,3}(x) = (1 - x)^3, && b_{1,3}(x) = 3x(1 - x)^2, && b_{2,3}(x) = 3x^2(1 - x), && b_{3,3}(x) = x^3
\end{aligned}
$$

- **Approximating continuous functions**
  - let f be a continuous function on the interval [0, 1]

$$B_n(f)(x) = \sum_{\nu=0}^{n} f\left(\frac{\nu}{n}\right) b_{\nu,n}(x) \qquad \lim_{n\to\infty} B_n(f)(x) = f(x)$$

Bézier curve

- **A Bézier curve is a parametric curve**
  - used to model smooth curves that can be scaled indefinitely



Bézier curve

- **The mathematical basis for Bézier curves — the Bernstein polynomial**
  - known since 1912
  - its applicability to graphics was not realized for another half century
  - Bézier curves were widely publicized in 1962 by the French engineer Pierre Bézier, who used them to design automobile bodies at Renault

Bézier curve

- **Linear Bézier curves**

$$\mathbf{B}(t) = \mathbf{P}_0 + t(\mathbf{P}_1 - \mathbf{P}_0) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1,\ 0 \le t \le 1$$

- **Quadratic Bézier curves**

$$\mathbf{B}(t) = (1-t)[(1-t)\mathbf{P}_0 + t\mathbf{P}_1] + t[(1-t)\mathbf{P}_1 + t\mathbf{P}_2],\ 0 \le t \le 1$$

$$\mathbf{B}(t) = (1-t)^2\mathbf{P}_0 + 2(1-t)t\mathbf{P}_1 + t^2\mathbf{P}_2,\ 0 \le t \le 1$$

Bézier curve

- **Cubic Bézier curves**

$$\mathbf{B}(t) = (1-t)\mathbf{B}_{\mathbf{P}_0,\mathbf{P}_1,\mathbf{P}_2}(t) + t\mathbf{B}_{\mathbf{P}_1,\mathbf{P}_2,\mathbf{P}_3}(t)$$

$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2 t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3,\ 0 \le t \le 1$$

- **General definition**
  - recursive definition

$$\mathbf{B}_{\mathbf{P}_0}(t) = \mathbf{P}_0, \text{ and}$$

$$\mathbf{B}(t) = \mathbf{B}_{\mathbf{P}_0\mathbf{P}_1\ldots\mathbf{P}_n}(t) = (1-t)\mathbf{B}_{\mathbf{P}_0\mathbf{P}_1\ldots\mathbf{P}_{n-1}}(t) + t\mathbf{B}_{\mathbf{P}_1\mathbf{P}_2\ldots\mathbf{P}_n}(t)$$

Bézier curve

- **General definition**
  - explicit definition

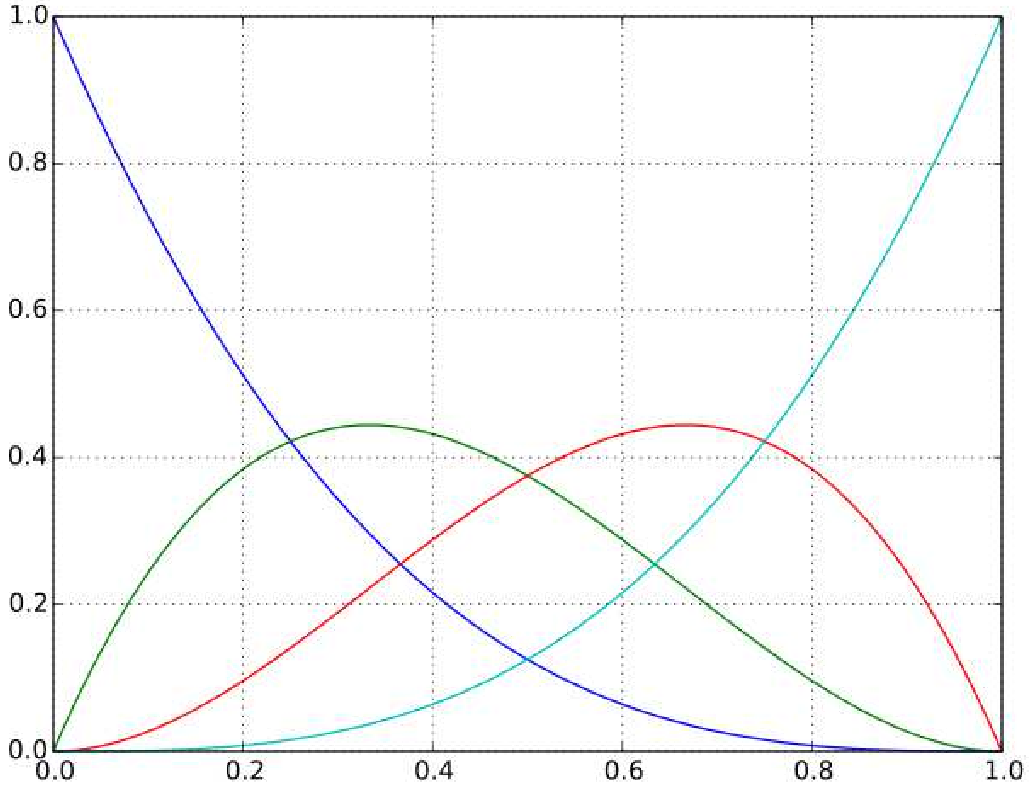
$$\begin{aligned}\mathbf{B}(t) &= \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \mathbf{P}_i \\ &= (1-t)^n \mathbf{P}_0 + \binom{n}{1} (1-t)^{n-1} t \mathbf{P}_1 + \cdots \\ &\quad \cdots + \binom{n}{n-1} (1-t) t^{n-1} \mathbf{P}_{n-1} + t^n \mathbf{P}_n, \quad 0 \leq t \leq 1\end{aligned}$$

- **Representation using Bernstein polynomial**

$$\mathbf{B}(t) = \sum_{i=0}^{n} b_{i,n}(t) \mathbf{P}_i, \quad 0 \leq t \leq 1$$

Bézier curve

- **The basis functions on the range t in [0,1] for cubic Bézier curves**



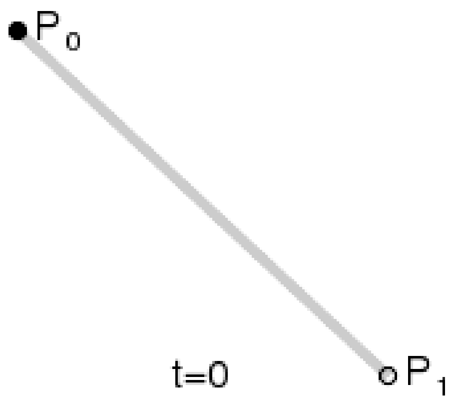
blue: $y_0 = (1 - t)^3$
green: $y_1 = 3(1 - t)^2\, t$
red: $y_2 = 3(1 - t)\, t^2$
cyan: $y_3 = t^3$

Bézier curve

- **Evaluation**
  - de Casteljau's algorithm
    - recurrence relation

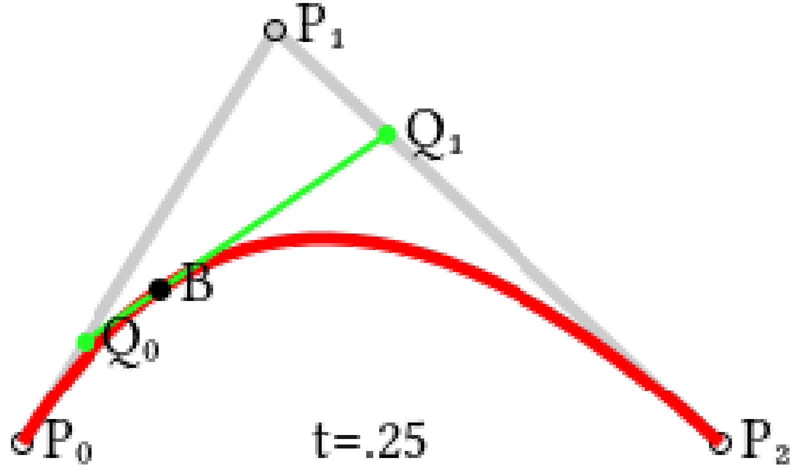
$$\beta_i^{(0)} := \beta_i, \; i = 0, \ldots, n$$
$$\beta_i^{(j)} := \beta_i^{(j-1)}(1 - t_0) + \beta_{i+1}^{(j-1)} t_0, \; i = 0, \ldots, n - j, \; j = 1, \ldots, n$$

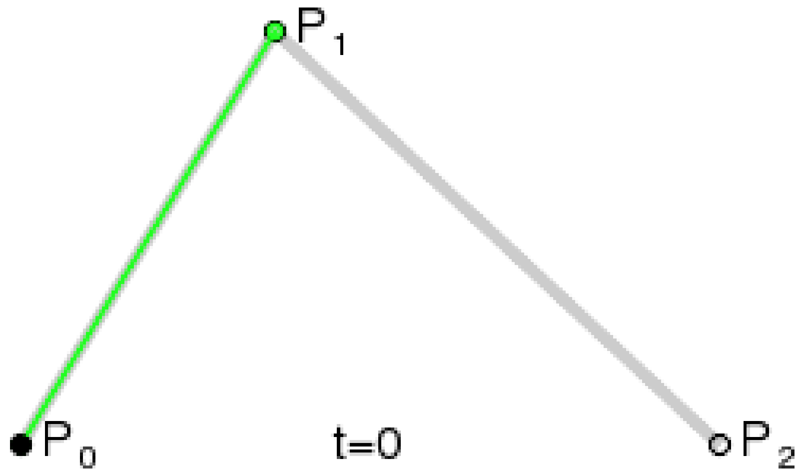
- **Linear curves**



Bézier curve

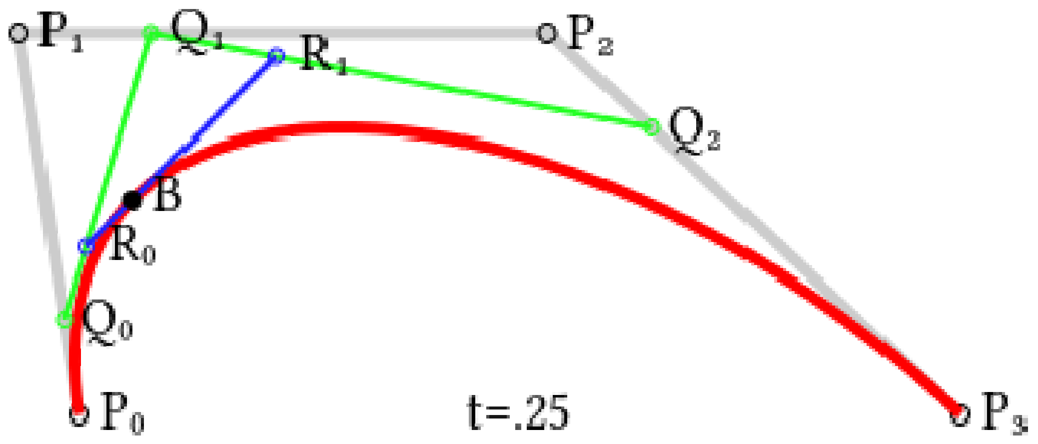
- **Quadratic curves**

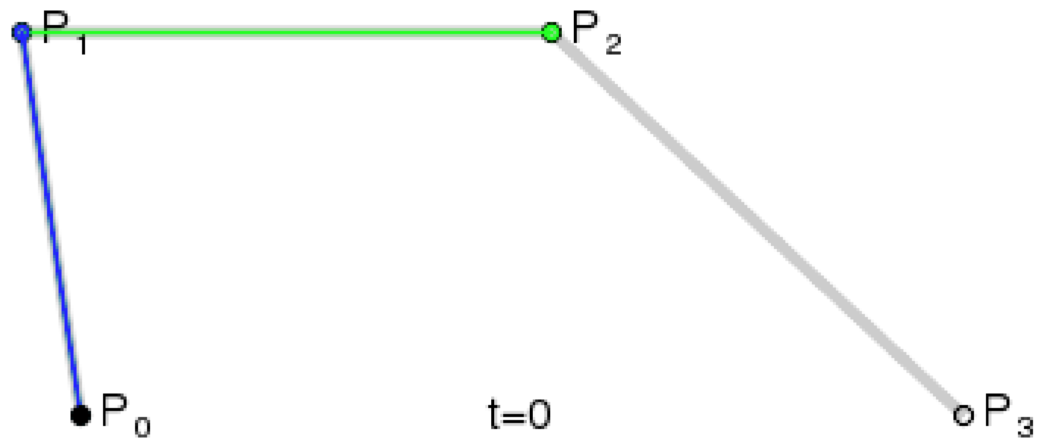




Bézier curve

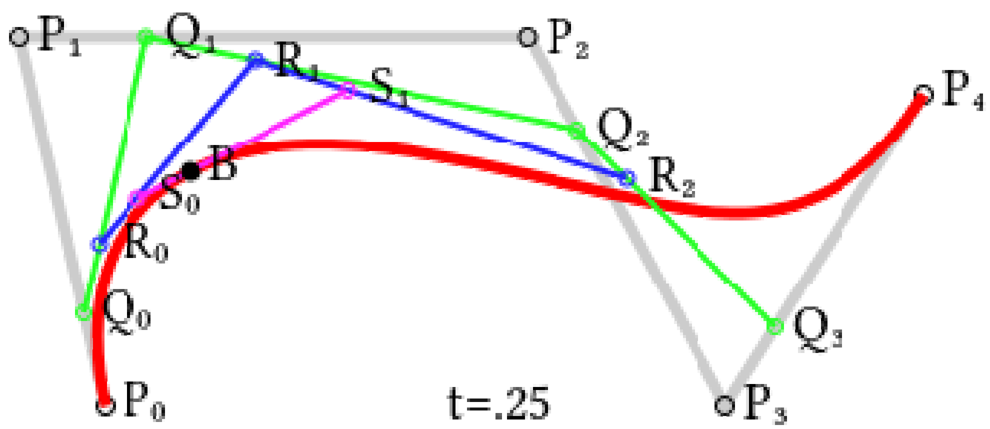
- **Higher-order curves**
  - cubic Bézier curve

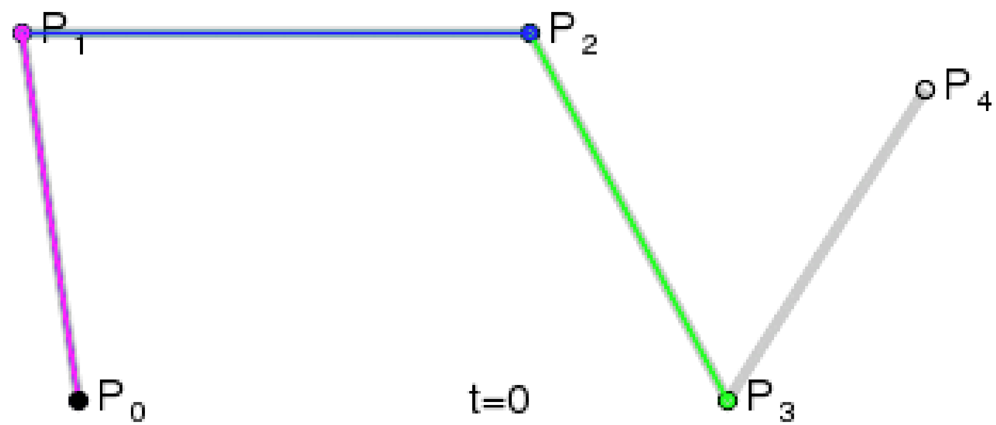




Bézier curve

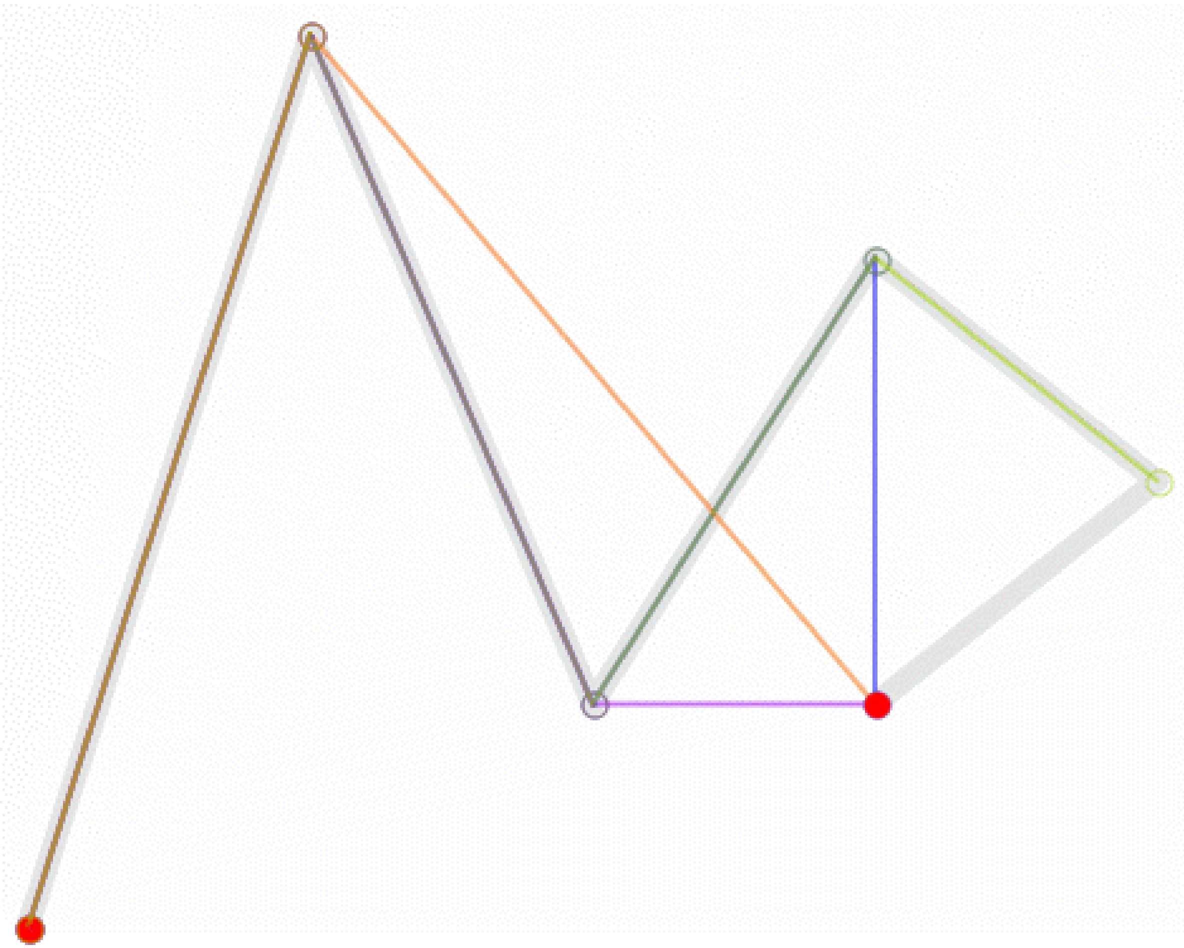
- **Higher-order curves**
  - fourth-order curves





Bézier curve

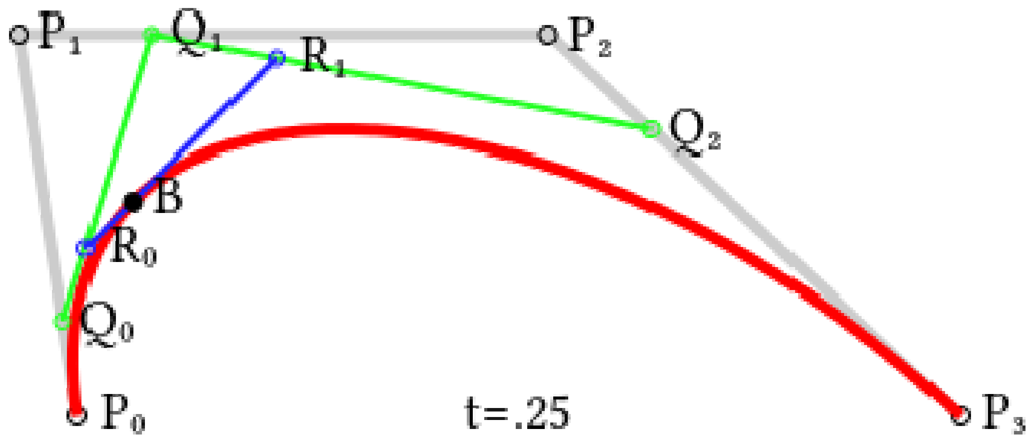
- **Higher-order curves**
  - For fifth-order curves



Bézier curve

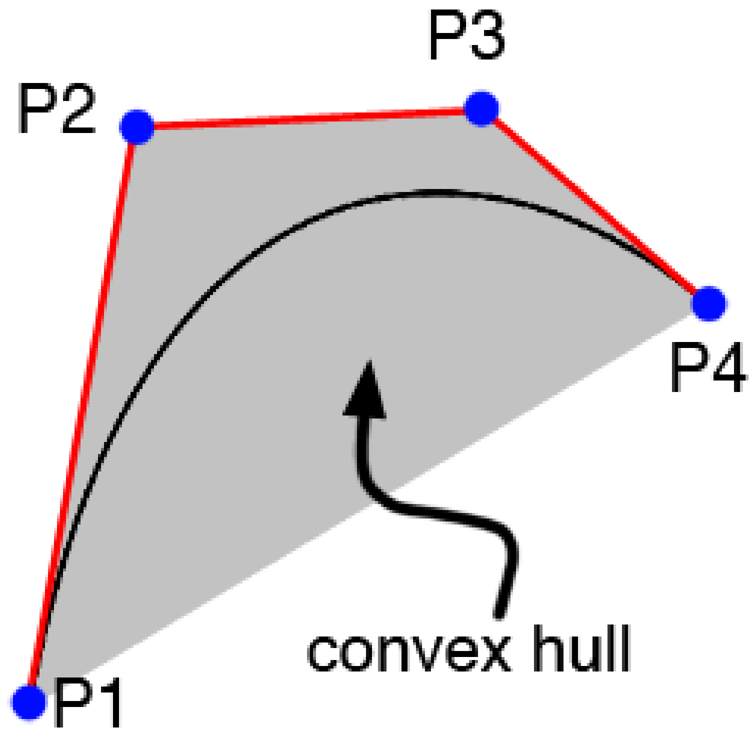
- **Computing the tangent vector**
  – the tangent can be directly obtained from the evaluation process by de Casteljau's algorithm

$$\mathbf{t} = \mathbf{v}_{R0R1}/||\mathbf{v}_{R0R1}||$$



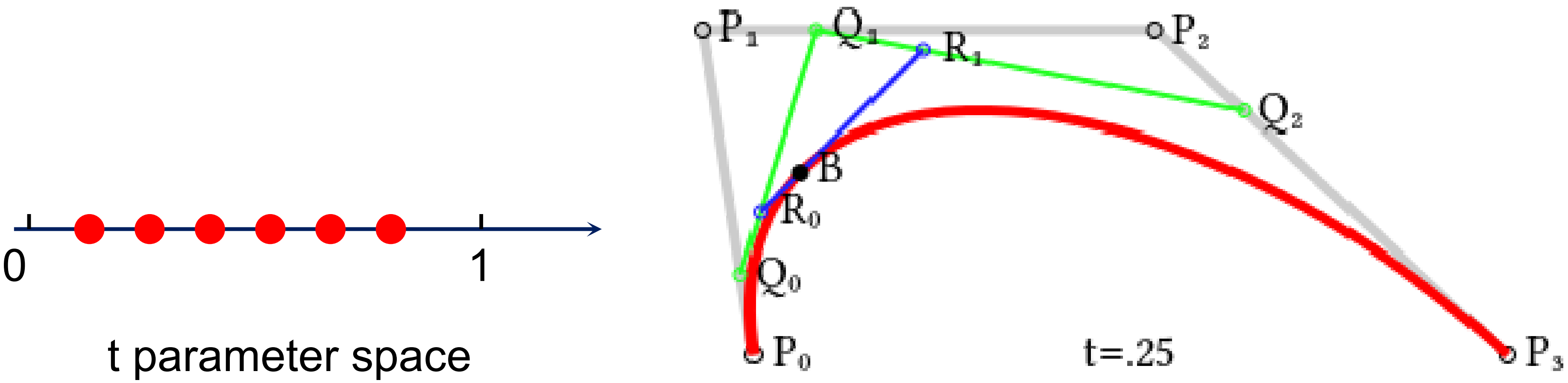
Bézier curve

- **Convex hull**
  - All Bézier curves always lie inside the convex hull
  - Convex hull edges tangential to the curve at end points



Meshing a Bézier curve

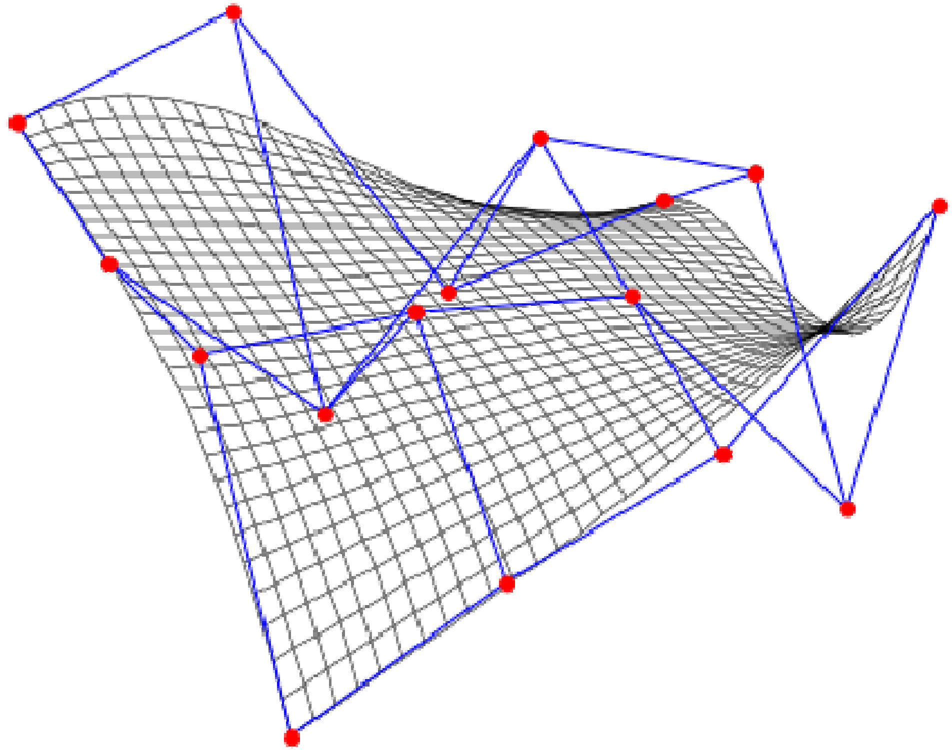
- **Meshing in parameter space**
  - Sample in parameter space and connect sample points by line segments



2.3.2. Bézier surface

Bézier surface

- **A Bézier surface of degree (n, m) is defined by a set of (n + 1)(m + 1) control points $\mathbf{k}_{i,j}$**
  – it maps the unit square into a smooth-continuous surface



Bézier surface

- **A two-dimensional Bézier surface can be defined as a parametric surface**
  - a tensor product of 1D Bézier curve

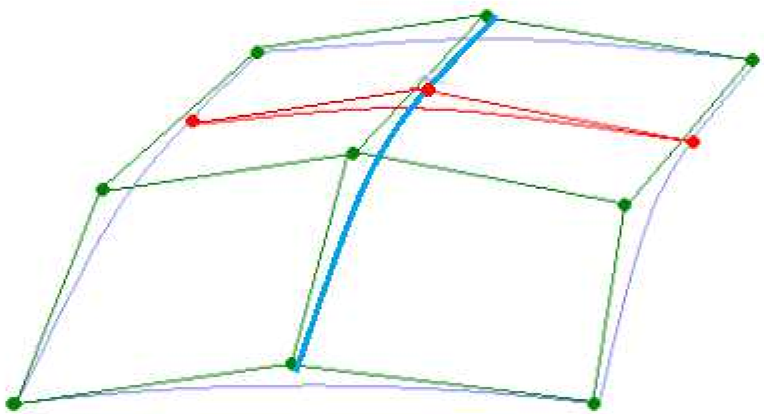
$$\mathbf{p}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_i^n(u)\; B_j^m(v)\; \mathbf{k}_{i,j}$$

evaluated over the unit square, where:

$$B_i^n(u) = \binom{n}{i}\, u^i\,(1-u)^{n-i}$$

Bézier surface

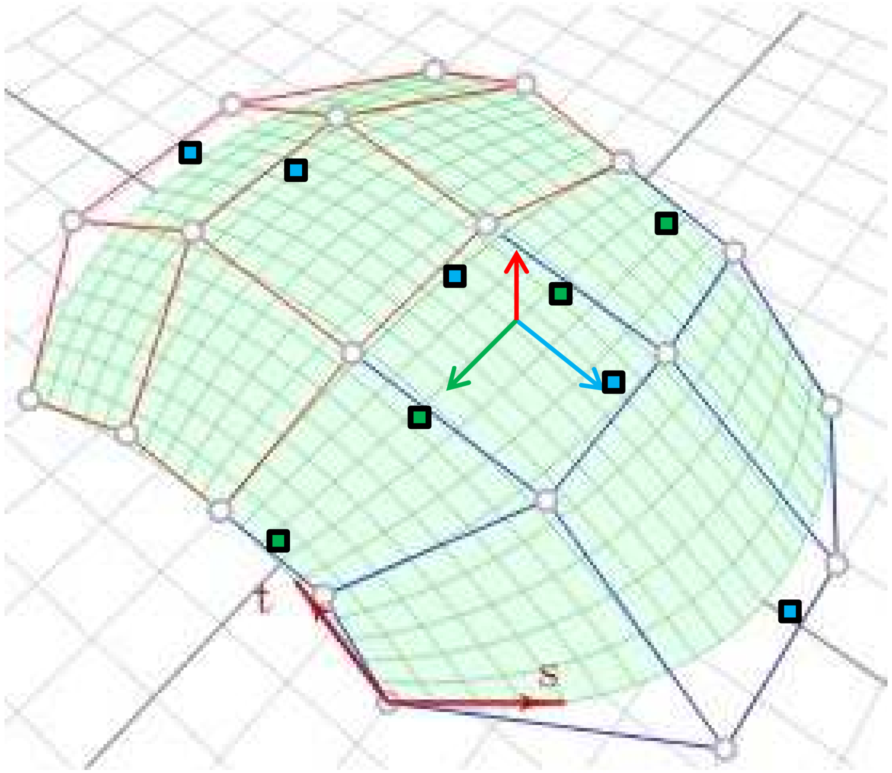
- **Evaluation**
  - recursively apply de Casteljau's algorithm
  - first, evaluate control points by de Casteljau's algorithm along one parameter direction
  - then, evaluate the final point by de Casteljau's algorithm again with the evaluated control points



$$\mathbf{p}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_i^n(u)\; B_j^m(v)\; \mathbf{k}_{i,j}$$

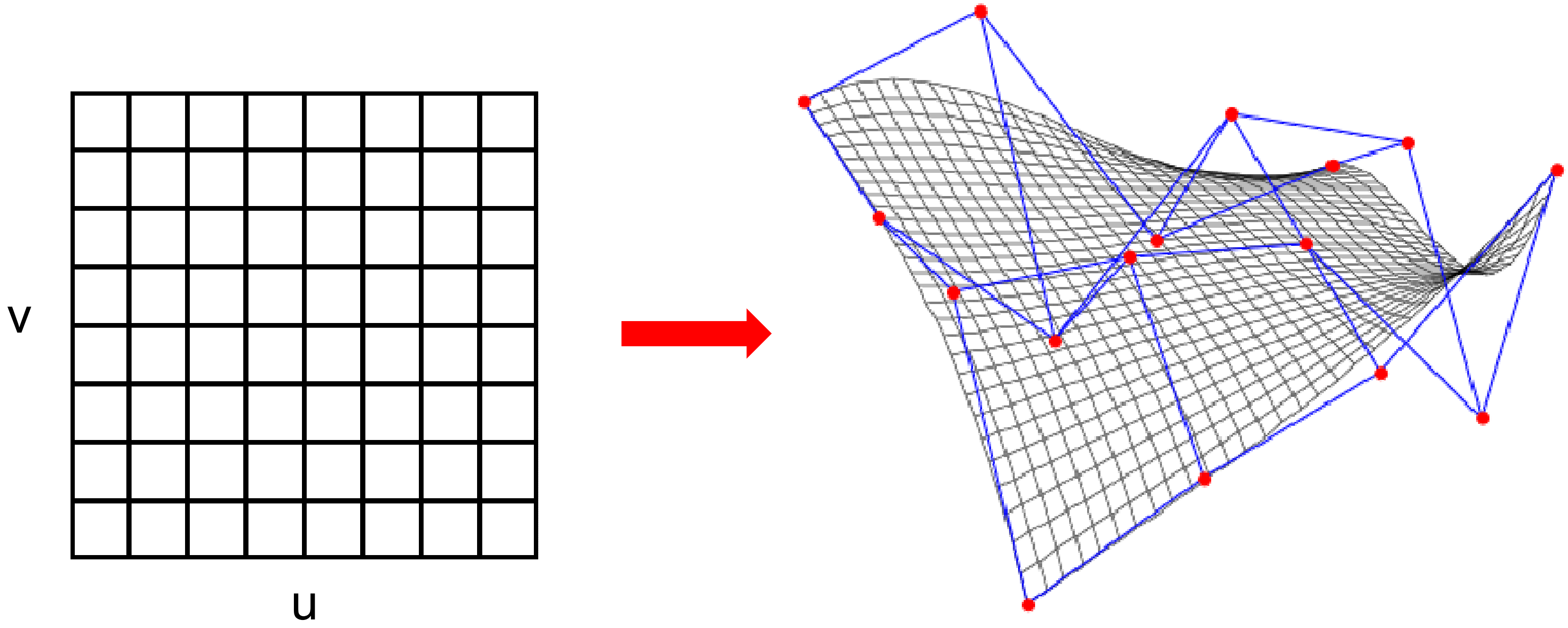
Bézier surface

- **Computing the tangents and normal**
  - compute the tangent of two crossing Bézier curves
  - then take the cross product of these two tangents to form the normal



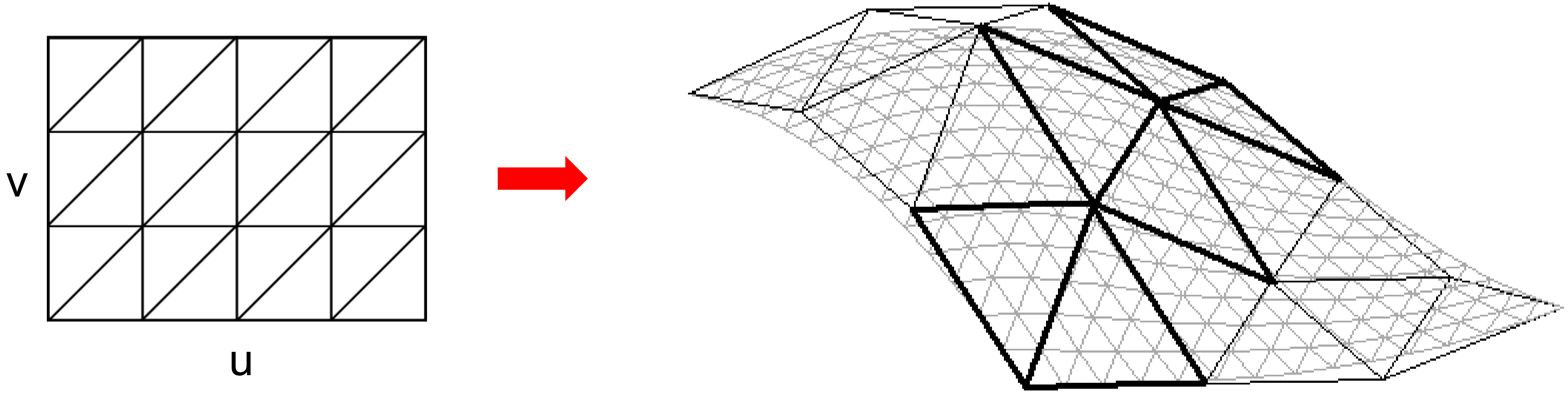
Meshing a Bézier surface

- **Meshing in parameter space**
  - gridding in u,v parameter space



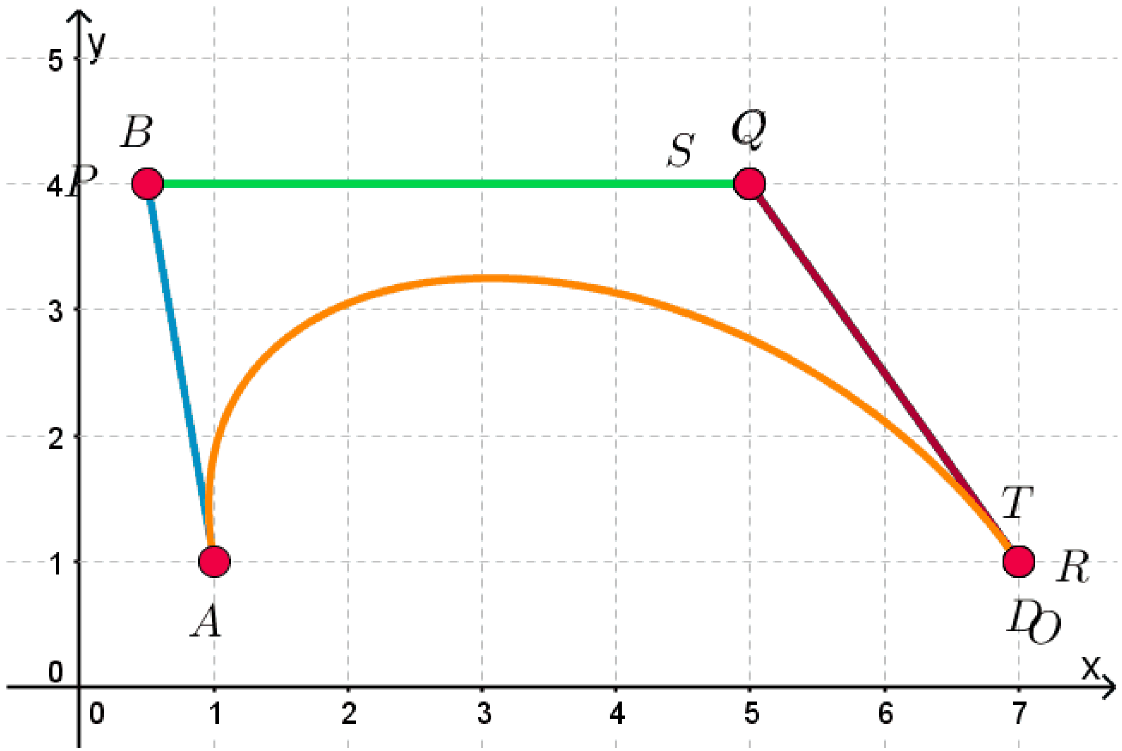
Meshing a Bézier surface

- **Meshing in parameter space**
  - triangulation in u,v parameter space



Problem of Bézier curve/surface

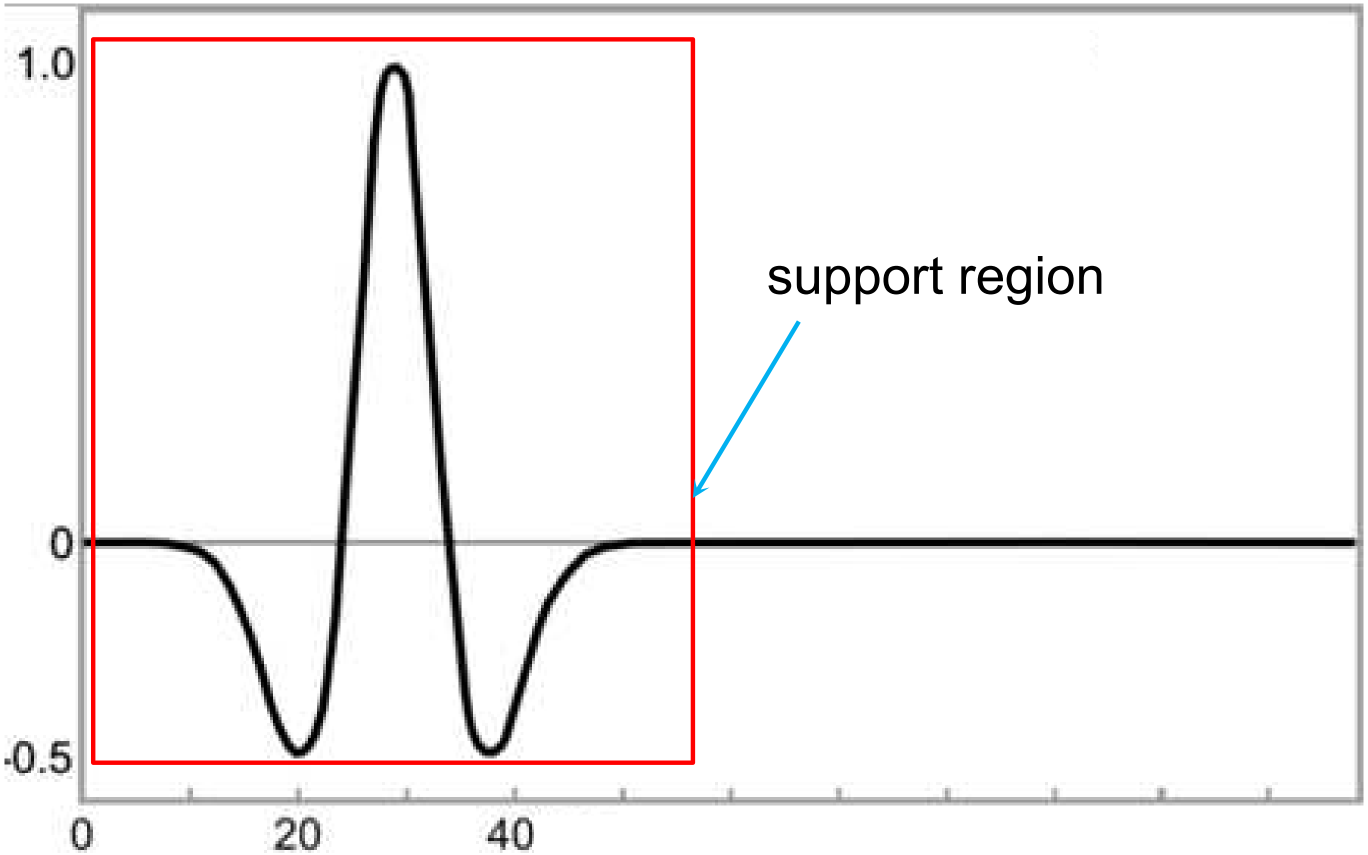
- **Change of local control points**
  - affect the whole curve/surface
  - change the shape of the whole curve/surface
  - require re-evaluation of the whole curve/surface



2.3.3. B-spline curve

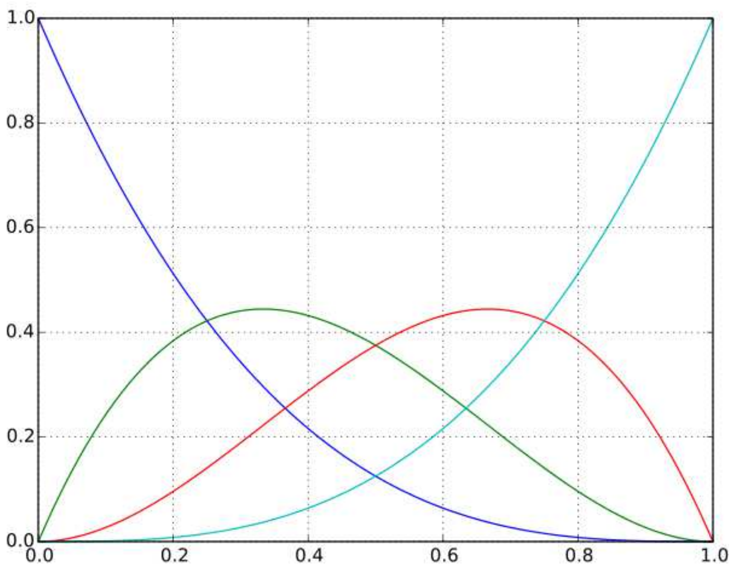
Support of basis functions

- **Definition**
  - regions of definition domain where the function value is non-zero

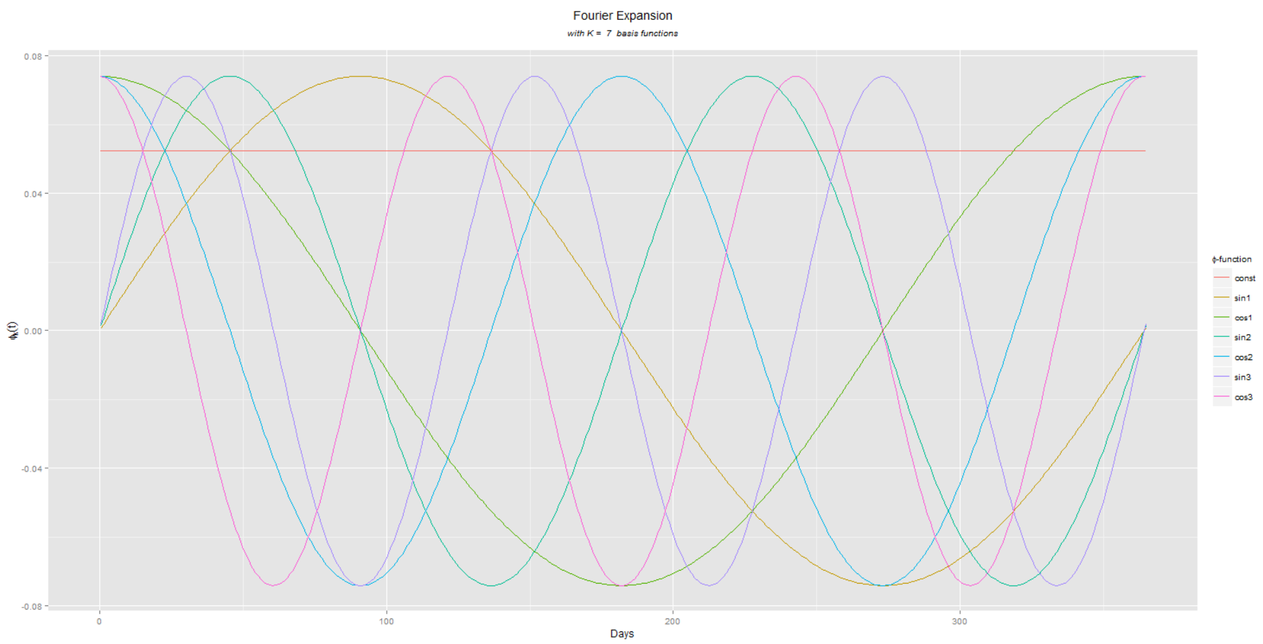


Support of basis functions

- **Global support**
  - support range over the whole definition domain
  - for example, Bernstein basis, Fourier basis, etc.



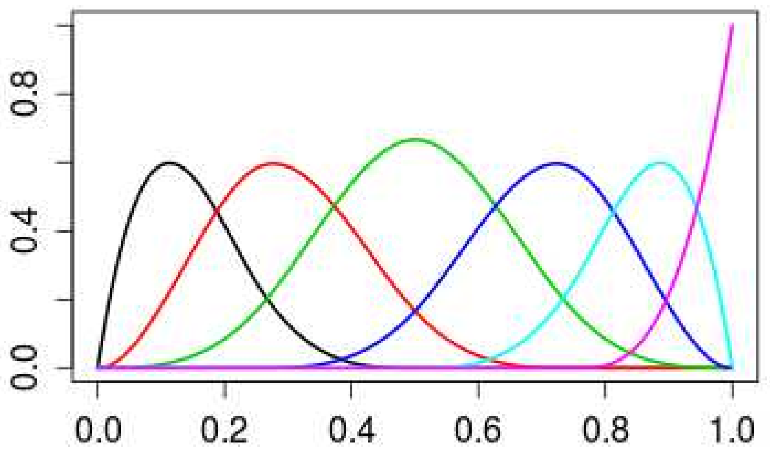
Bernstein basis



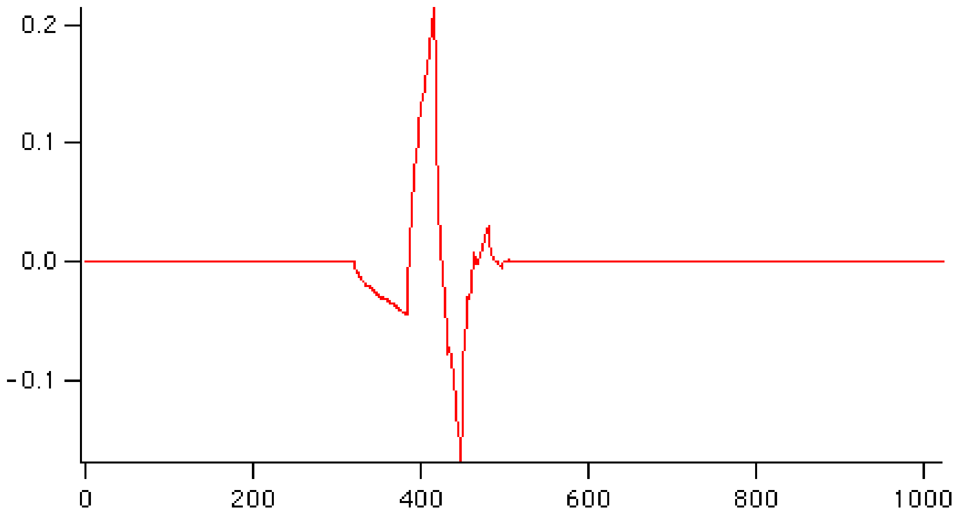
Fourier basis

Support of basis functions

- **Local support**
  - support range over a relatively narrow region in the definition domain
  - for example, B-spline basis, wavelet basis, etc.



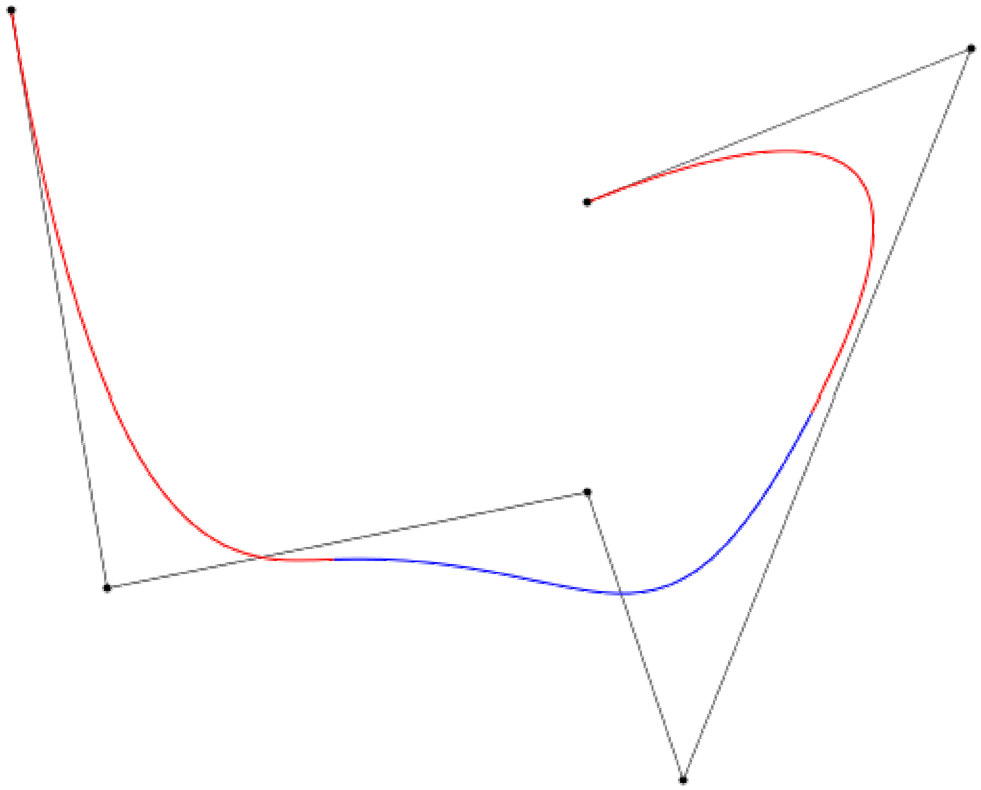
B-spline basis



wavelet basis

B-spline curve

- **Basis spline – B-spline**
  - a spline function that has minimal support with respect to a given degree



B-spline curve

- **In the computer-aided design and computer graphics**
  - linear combinations of B-spline basis functions with a set of control points
  - a spline function is a piecewise polynomial function of degree <k
  - the places where the pieces meet are known as knots
  - the number of internal knots must be equal to, or greater than k-1
  - the spline function has limited support

B-spline curve

- **Definition**
  - a B-spline of order n is a piecewise polynomial function of degree <n in a variable x

$$S_{n,t}(x) = \sum_i \alpha_i B_{i,n}(x)$$

  - Cox-de Boor recursion formula

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

$$B_{i,k}(x) := \frac{x - t_i}{t_{i+k-1} - t_i} B_{i,k-1}(x) + \frac{t_{i+k} - x}{t_{i+k} - t_{i+1}} B_{i+1,k-1}(x)$$

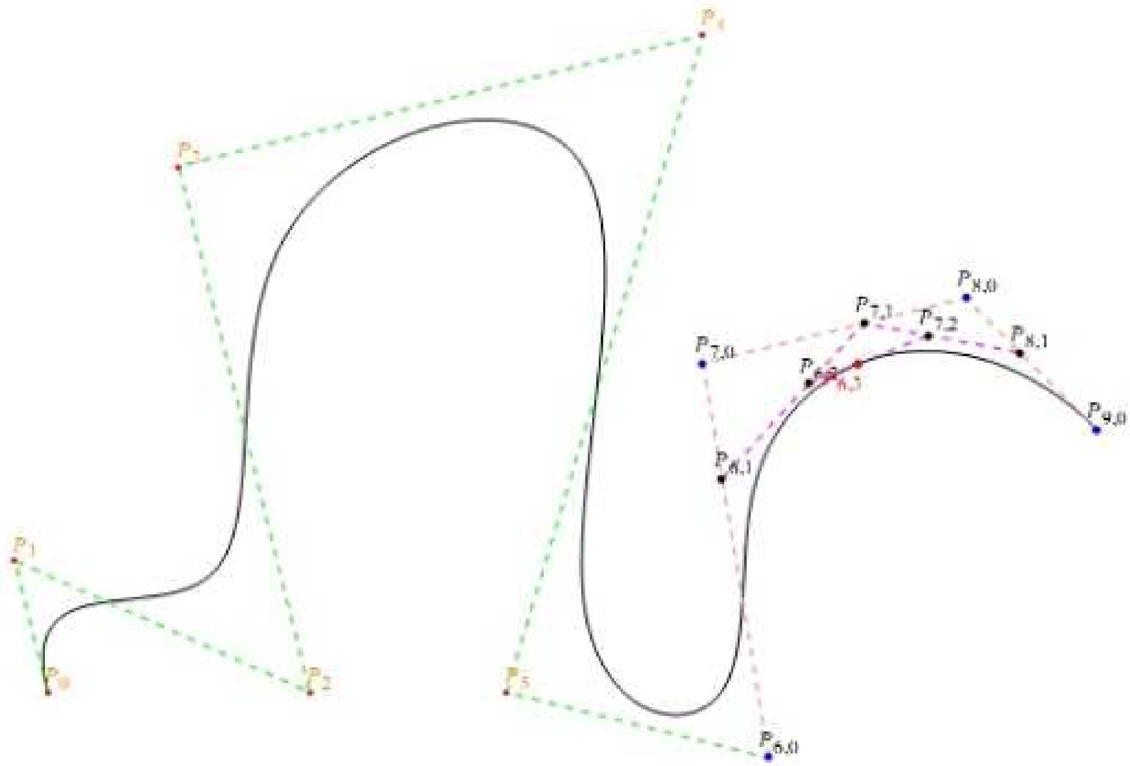
B-spline curve

- **Recursion formula with the knots at 0, 1, 2, and 3 gives the pieces of the uniform B-spline of degree 2:**

$$B_1 = x^2/2 \qquad 0 \leq x \leq 1$$
$$B_2 = (-2x^2 + 6x - 3)/2 \qquad 1 \leq x \leq 2$$
$$B_3 = (3 - x)^2/2 \qquad 2 \leq x \leq 3$$

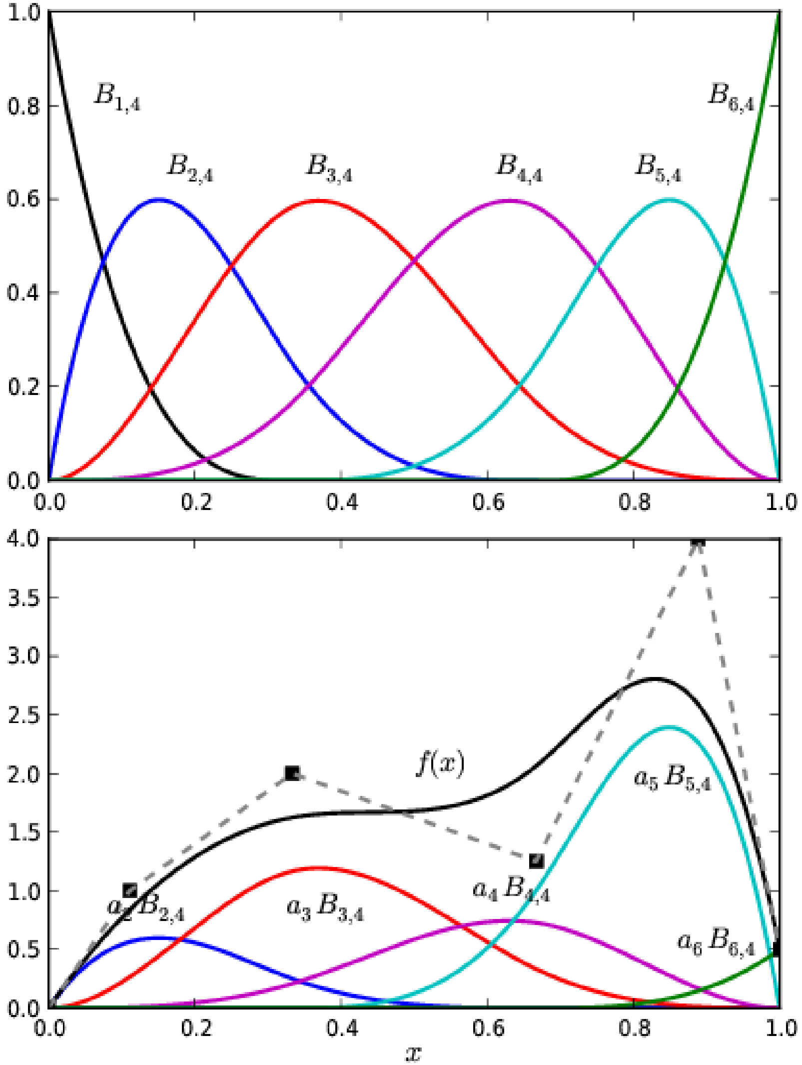
B-spline curve

- **Evaluation**
  - de Boor algorithm
  - find the support range of the current parameter
  - apply recursive evaluation like in Bézier curve evaluation



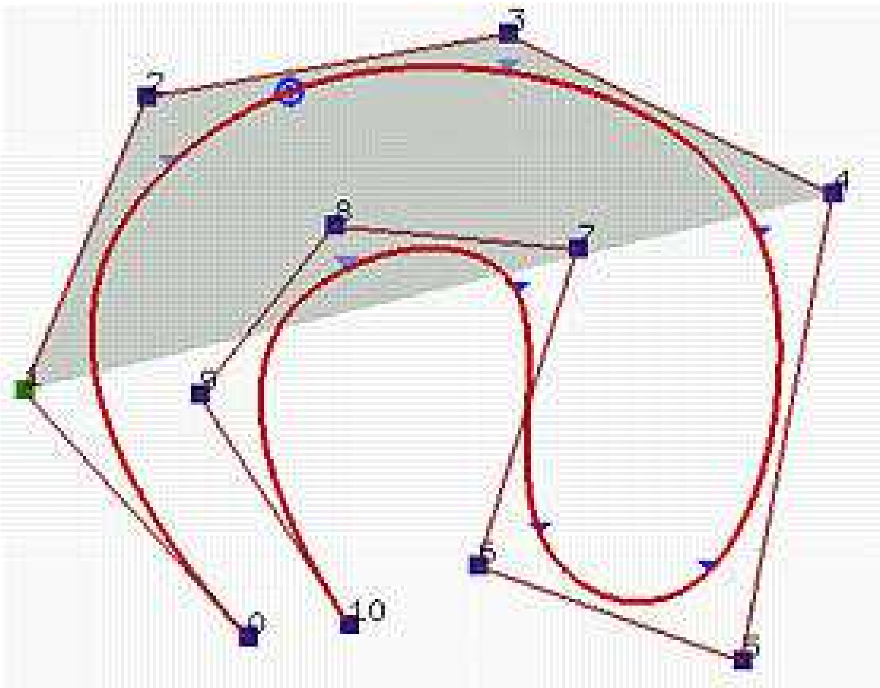
B-spline curve

- **B-spline basis and curve synthesis**



B-spline curve

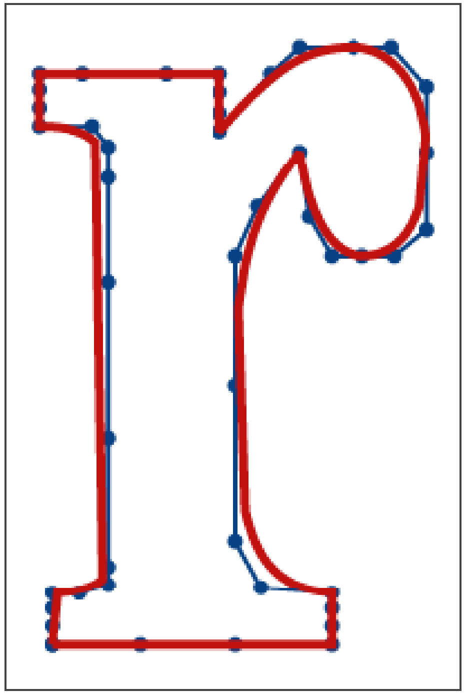
- **Convex hull**
  - like Bézier curves, all B-spline curves always lie inside the convex hull
  - convex hull is defined locally and changed with respect to different parts of the curve



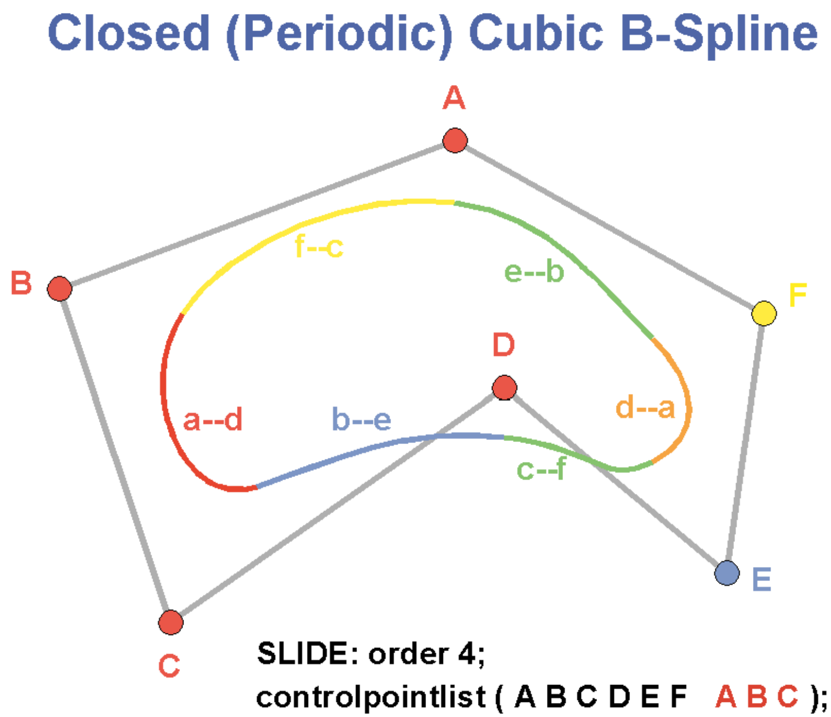
B-spline curve

- **More complex examples**

Font as a B-spline curve



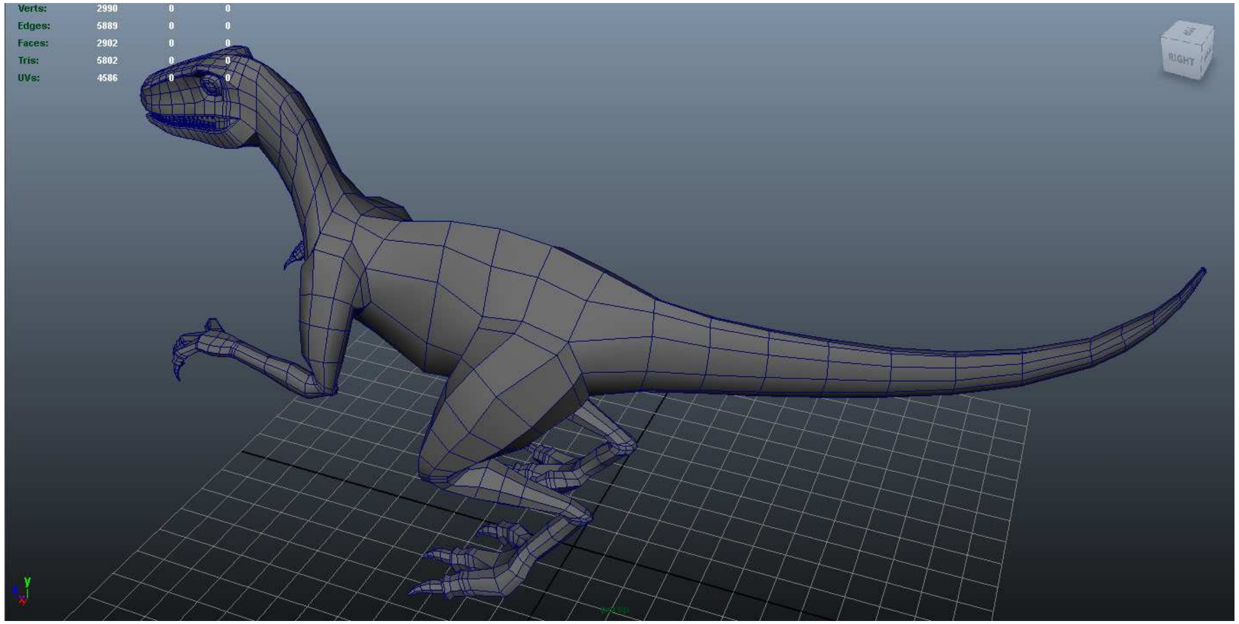
Data: G.Farin, Curves and Surfaces for Computer Aided Geometric Design

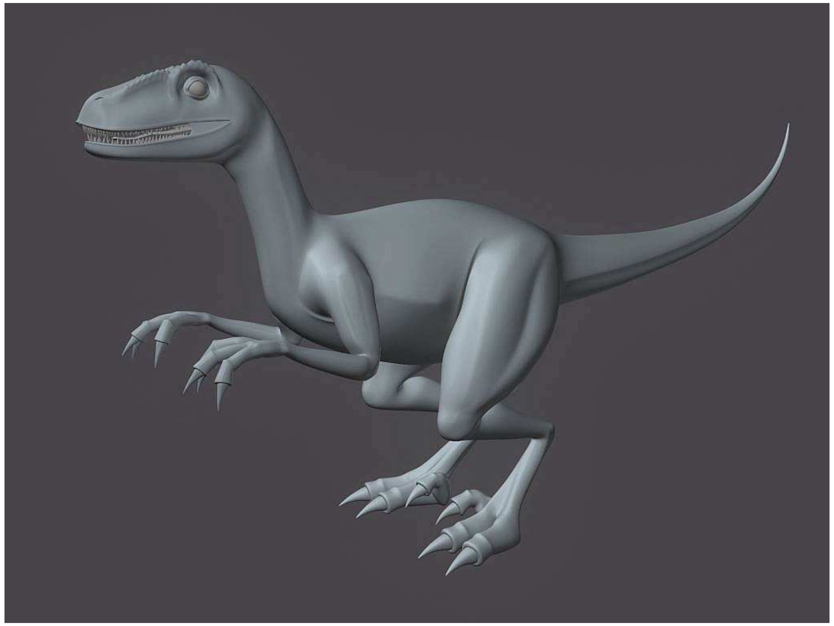


2.3.4. B-Spline/NURBS surface

B-spline surface

- **Like Bézier surface, B-spline surface can be constructed with tensor-product**
  - meshing in u-v parameter space





NURBS

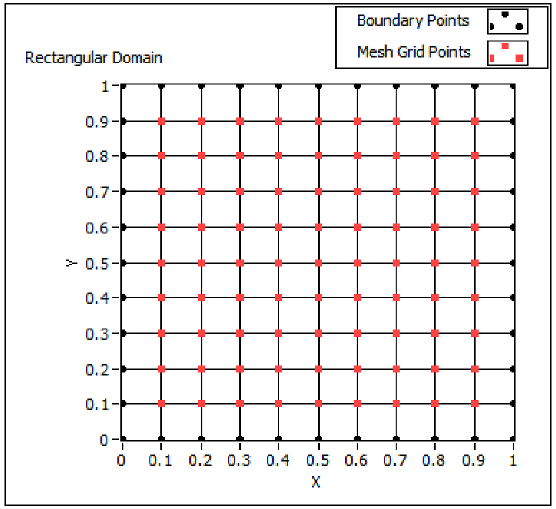
- **Non-uniform rational B-Spline**
  - formulation

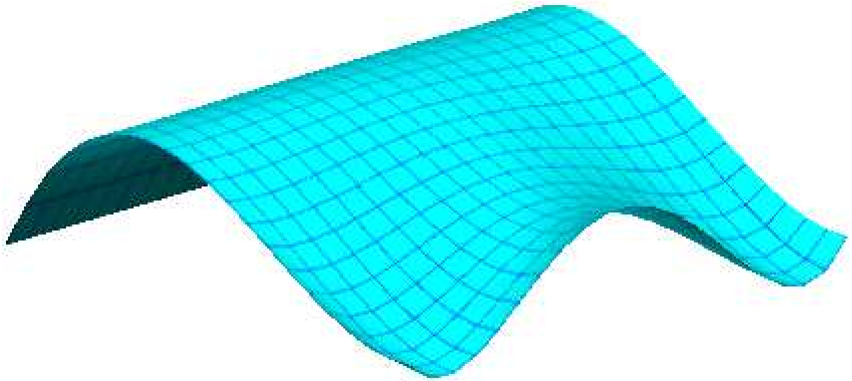
$$C(u) = \sum_{i=1}^{k} \frac{N_{i,n} w_i}{\sum_{j=1}^{k} N_{j,n} w_j} \mathbf{P}_i = \frac{\sum_{i=1}^{k} N_{i,n} w_i \mathbf{P}_i}{\sum_{i=1}^{k} N_{i,n} w_i}$$

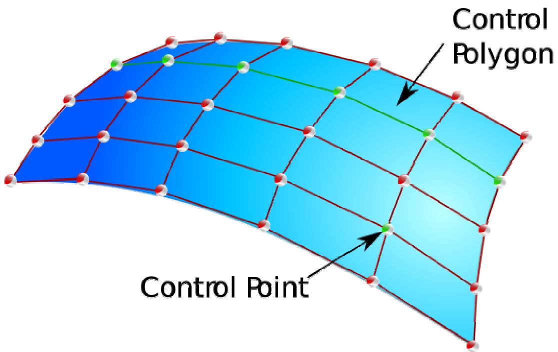
  - NURBS is commonly used in computer-aided design (CAD), manufacturing (CAM), and engineering (CAE)
  - part of numerous industry wide standards, such as IGES, STEP, ACIS, and PHIGS

Free-form surface triangulation

- **How to create meshes for free-form surfaces?**
  - create mesh in u-v parameter space

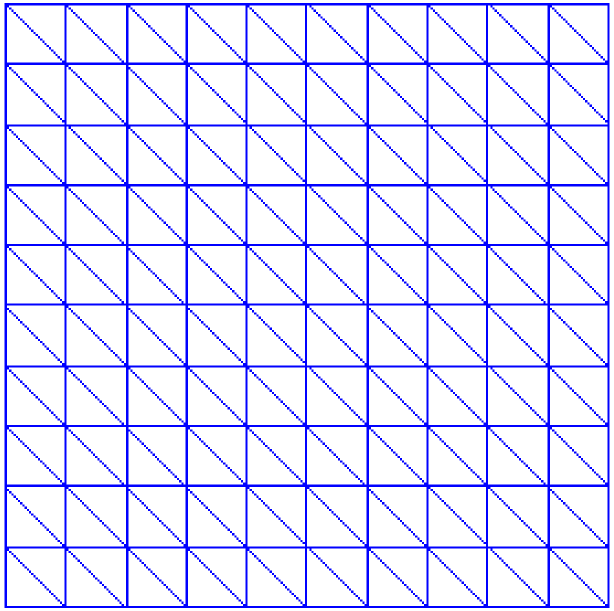


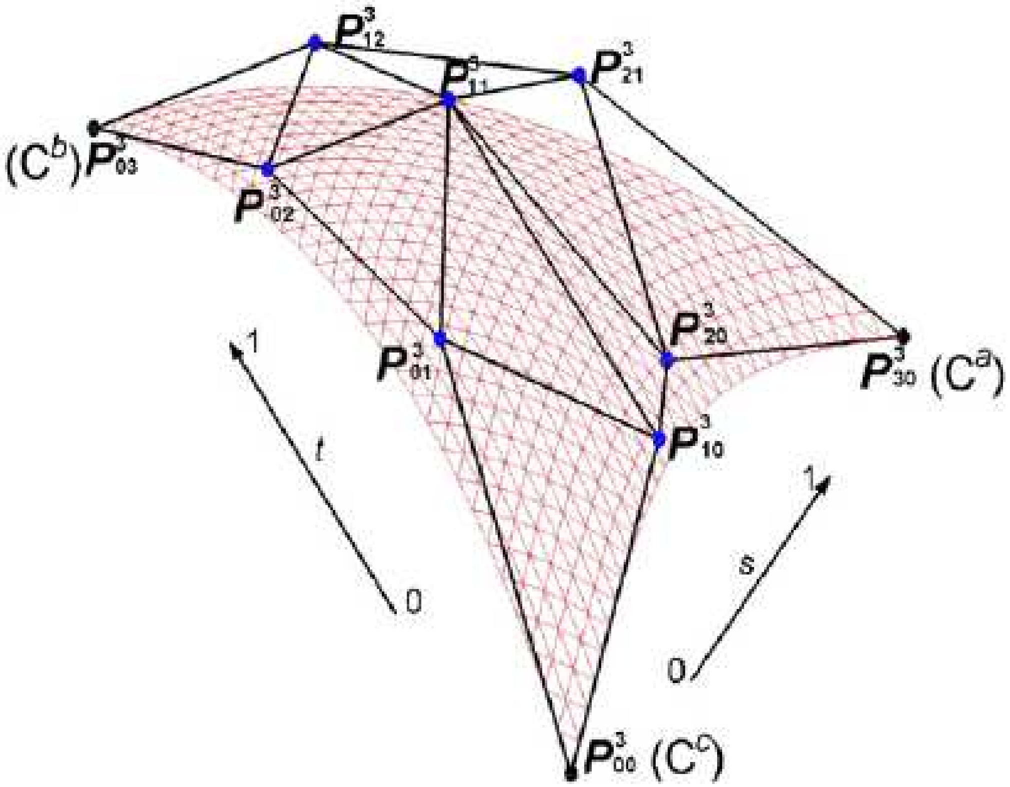




Free-form surface triangulation

- **Triangulation in parameter space**





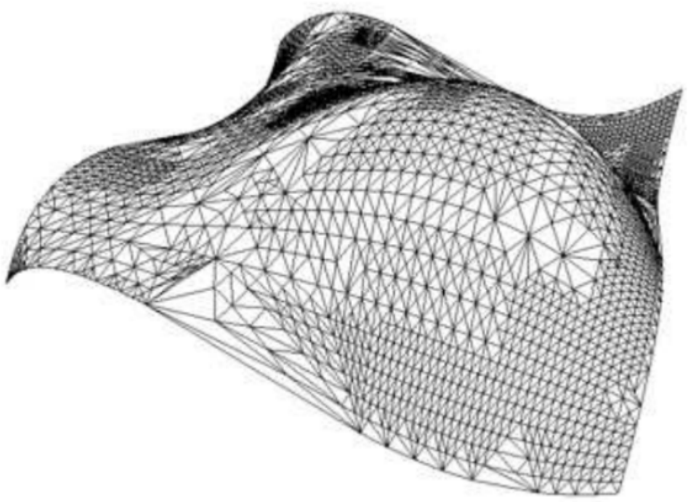
Free-form surface triangulation

- **Problem with uniform meshing in parameter space**
  - large deformation will distort triangles

- **Adaptive triangulation according to some criteria**
  - boundary, surface deformation (curvature)
  - criteria estimated from the control mesh

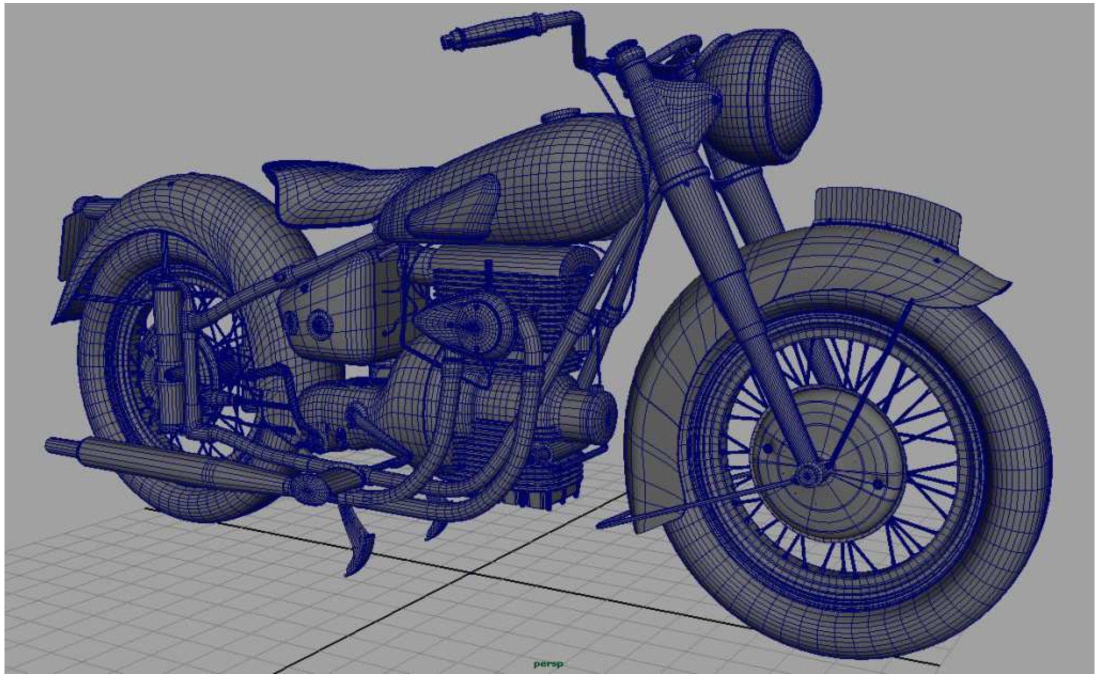
Free-form surface triangulation

- **Construct triangle meshes with storage consistent to OpenGL**
  - Vertex position/normal array + index array
  - Render with OpenGL vertex array



Free-form surface modeling

- **Design by control points**



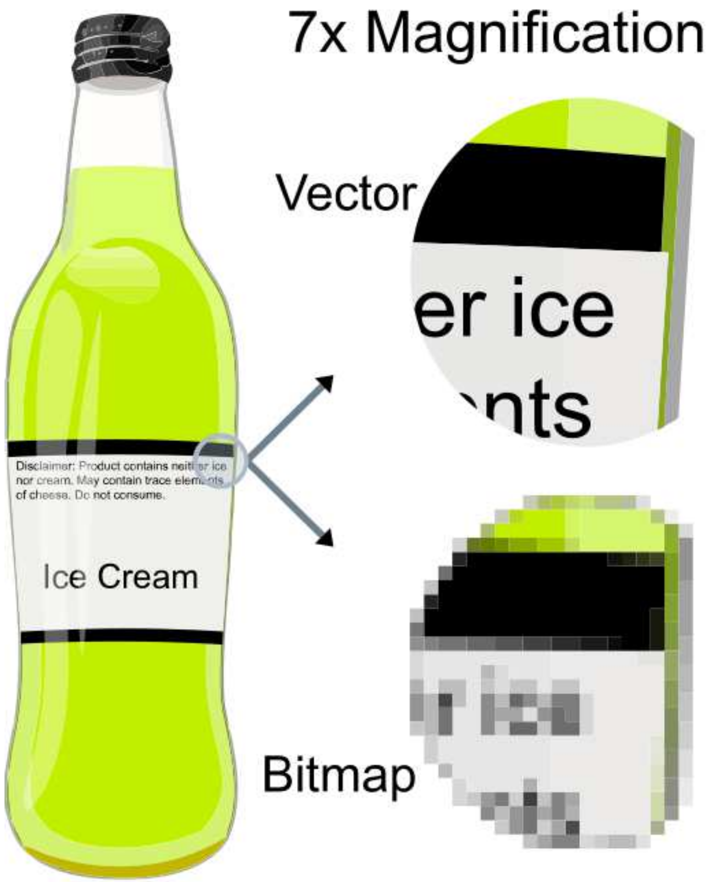
3. Vector graphics

Vector graphics

- **Vector graphics is the use of polygons to represent images in computer graphics**
  - based on vectors, which lead through locations called control points or nodes
  - ideal for printing
  - unlimited zoom-in and zoom-out without aliasing

Vector graphics

- **Benefit of vector graphics**
  - compact representation
  - aliasing-free display (rasterization)



Vector graphics

- **More examples**
  - filling based on free-form surfaces

Image vectorization

- **Create vector representation of a natural input image**

Image vectorization

- **Diffusion curves**
  - create continuous curves to represent image edges
  - the content of the image can be filled by Poisson equation solver

(a) (b) (c)

Next lecture: Geometric modeling 2