Shape is an element of art

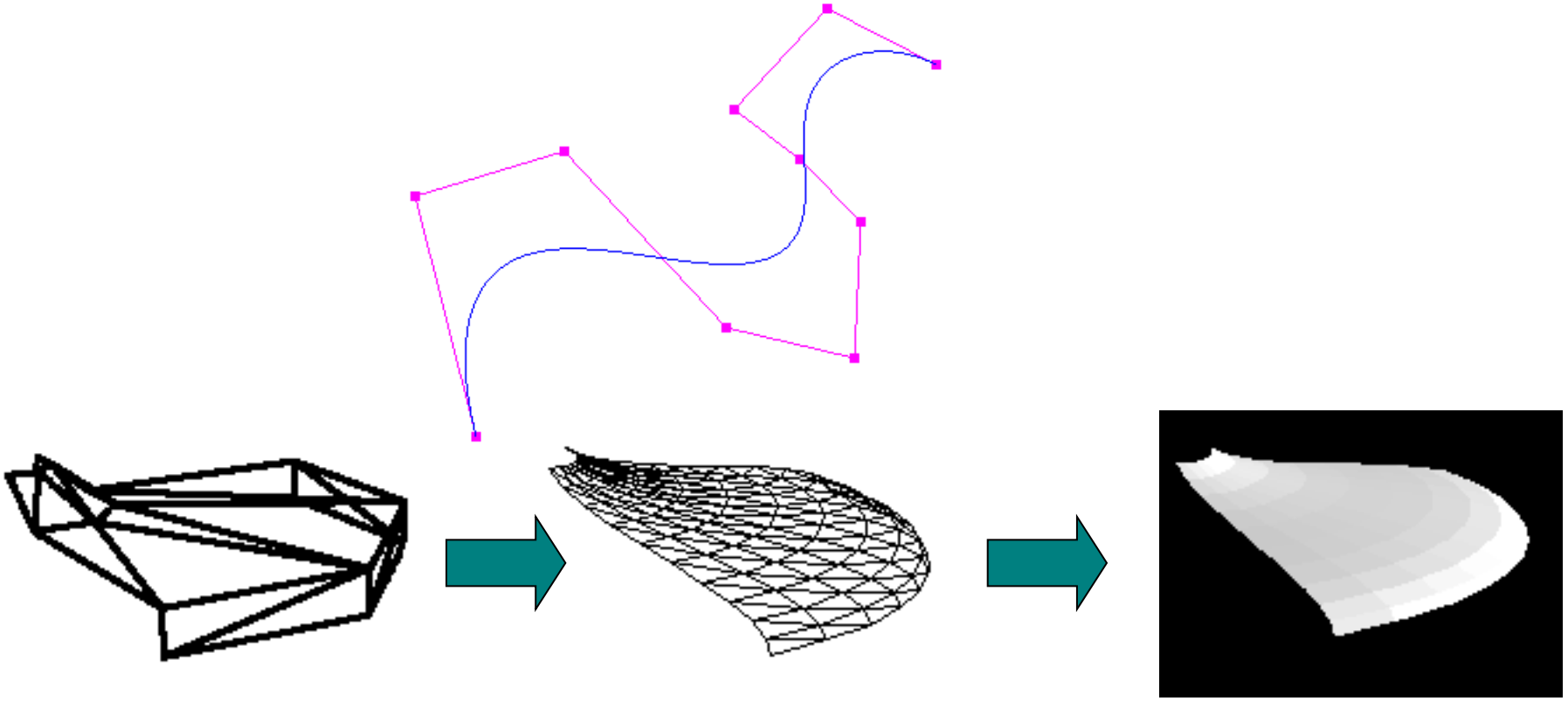
10 Curves and Surfaces

Department of Computer Science and Software Engineering
University of Canterbury, New Zealand.

R. Mukundan (mukundan@canterbury.ac.nz)

Motivation

We often require methods for procedurally generating complex surface geometries using only a coarse definition of their shapes specified by a small set of points (2D) or polygons (3D).



Curves and Surfaces

- Key idea:

  We can combine a set of points using parametric functions to generate a smooth and continuous curve.

  Given a set of *n* points $P_0, P_1, \ldots P_{n-1}$, and parametric functions $f_0(t), f_1(t), \ldots f_{n-1}(t)$, we can generate *any* number of points along a curve given by

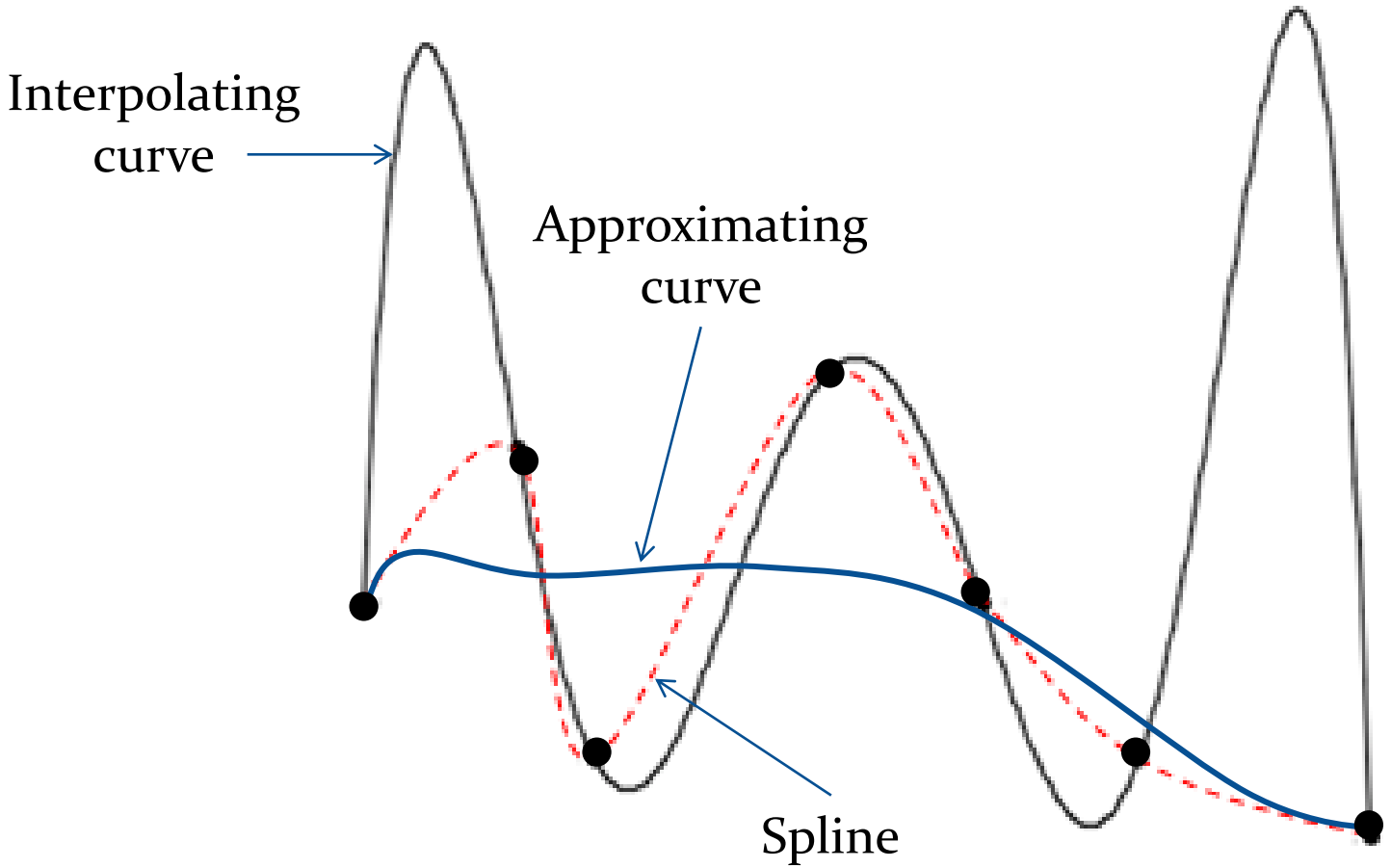
  $$P(t) = P_0 f_0(t) + P_1 f_1(t) + \ldots + P_{n-1} f_{n-1}(t)$$

- The points P_i are called "control points" and the functions "blending functions".
- Main requirement: The generated curve should be continuous and closely following the shape defined by the control points.

Curves and Surfaces

- There are three types of modelling curves:
  - Interpolating curve: A single polynomial curve that pass through *every* control point.
  - Approximating curve: A polynomial curve that pass through only few of the control points.
  - Splines: Piecewise polynomial curves that have a specified degree of smoothness at the connecting points (knots)
- Applications:
  - Surface design (Bezier Surfaces, B-Splines, NURBS)
  - Keyframe animation, Interactive drawing (Catmull-Rom splines)
  - Vertex blending (Hermite polynomials)

Curves and Surfaces

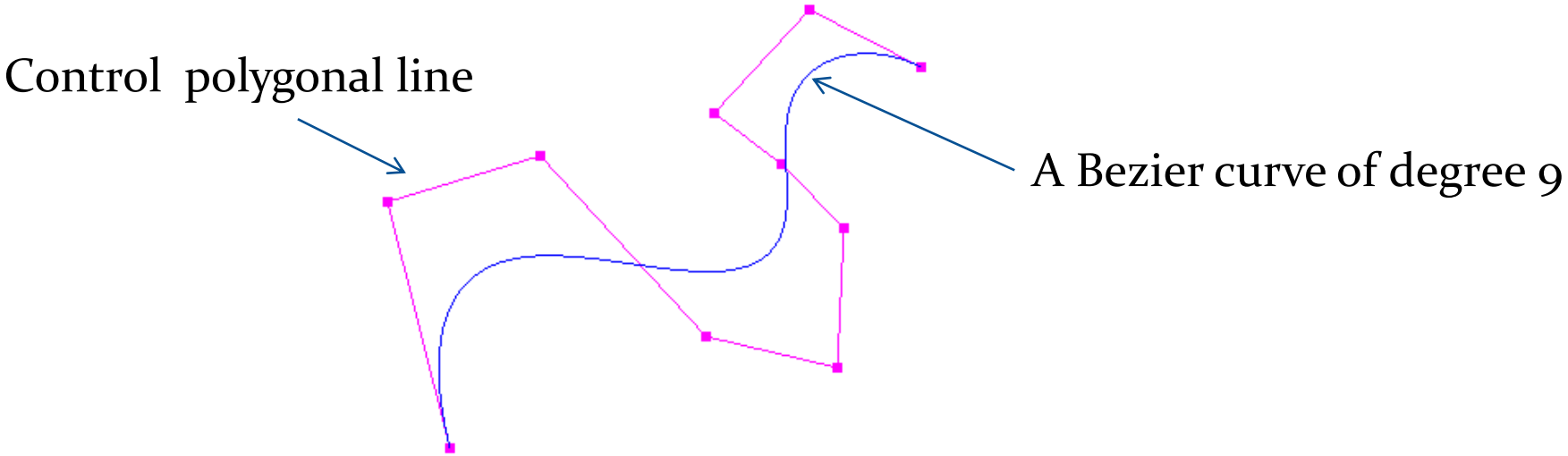


Interpolating Curves

- Polynomial interpolation theorem: Given n points (x_i, y_i), $i = 1..n$ on the xy-plane, where all x_i's are distinct, there exists a unique polynomial $f(x)$ of degree $n-1$ such that $f(x_i) = y_i$.
- Interpolation curves are generated using a combination of Lagrange polynomials of degree $n-1$.
- Polynomial interpolation curves are not very useful as they tend to have large overshoots and undesirable oscillations. They cannot be used for representing the shape defined by the control points.
- They also become numerically unstable when n becomes large.

Approximating Curves

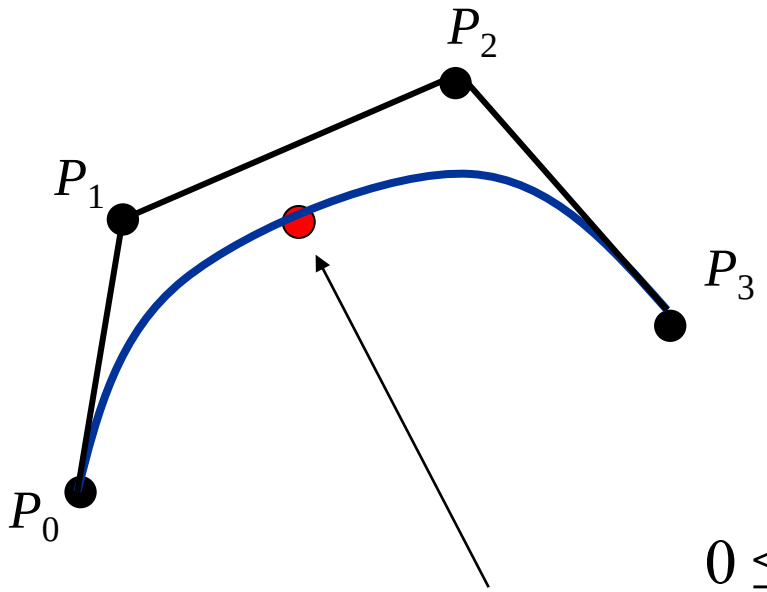
- Hermite curves and Bezier curves are examples of approximating curves.
- Bezier curves are defined using Bernstein polynomials as blending functions.
- Bezier curves are guaranteed to pass through only the first and the last control points.



Cubic Bezier Curve

- A cubic Bezier curve is defined using 4 control points.
- The blending functions are Bernstein polynomials of degree 3:

$$f_0^{(3)}(t) = (1-t)^3$$
$$f_1^{(3)}(t) = 3(1-t)^2 t$$
$$f_2^{(3)}(t) = 3(1-t)t^2,$$
$$f_3^{(3)}(t) = t^3.$$



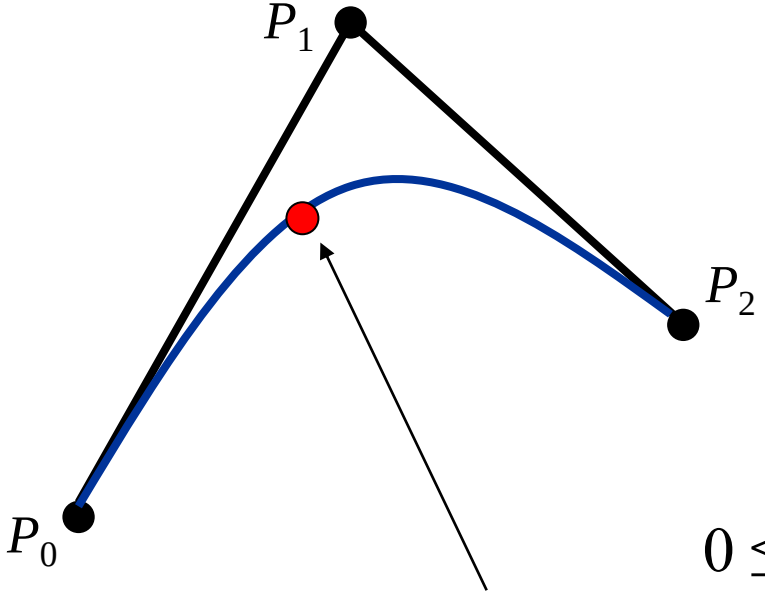
$$0 \le t \le 1.$$

$$x = (1-t)^3 x_0 + 3(1-t)^2 t\, x_1 + 3(1-t)t^2 x_2 + t^3 x_3$$
$$y = (1-t)^3 y_0 + 3(1-t)^2 t\, y_1 + 3(1-t)t^2 y_2 + t^3 y_3$$
$$z = (1-t)^3 z_0 + 3(1-t)^2 t\, z_1 + 3(1-t)t^2 z_2 + t^3 z_3$$

Quadratic Bezier Curve

- A quadratic Bezier curve is defined using 3 control points.
- The blending functions are Bernstein polynomials of degree 2:

$$f_0^{(2)}(t) = (1-t)^2$$
$$f_1^{(2)}(t) = 2(1-t)t$$
$$f_2^{(2)}(t) = t^2 ,$$



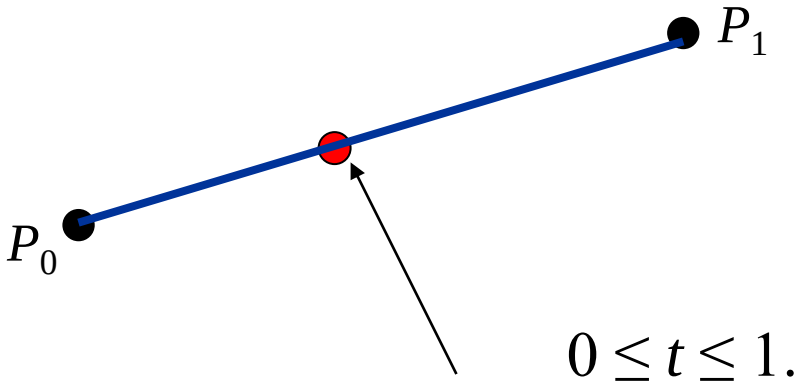
$$x = (1-t)^2 x_0 + 2(1-t)t\, x_1 + t^2 x_2$$
$$y = (1-t)^2 y_0 + 2(1-t)t\, y_1 + t^2 y_2$$
$$z = (1-t)^2 z_0 + 2(1-t)t\, z_1 + t^2 z_2$$

Linear Bezier Curve

- A linear Bezier curve is just a straight line defined using 2 control points.
- The blending functions are Bernstein polynomials of degree 1:

$f_0^{(1)}(t) = (1-t)$

$f_1^{(1)}(t) = t$



$$x = (1-t)\,x_0 + t\,x_1$$
$$y = (1-t)\,y_0 + t\,y_1$$
$$z = (1-t)\,z_0 + t\,z_1$$

Nth degree Bezier Curve

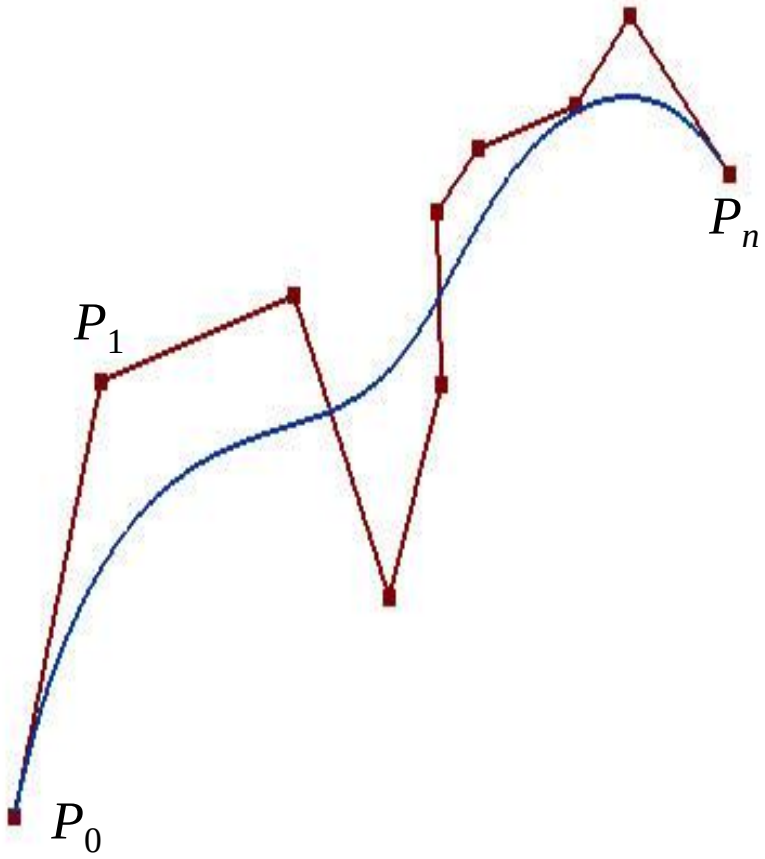
- An n^{th} degree Bezier curve is defined using n+1 control points.
- The blending functions are Bernstein polynomials of degree n.

$$f_i^{(n)}(t) = \binom{n}{i}(1-t)^{n-i}t^i$$

$$i = 0, 1, 2 \ldots n.$$

$$P(t) = \sum_{i=0}^{n} f_i^{(n)}(t) P_i$$

$$0 \le t \le 1.$$

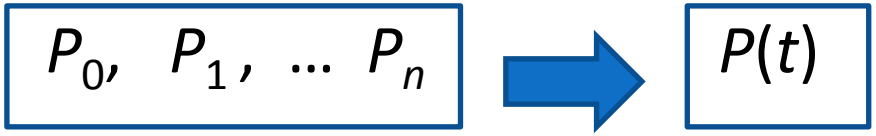


Properties of Bezier Curves

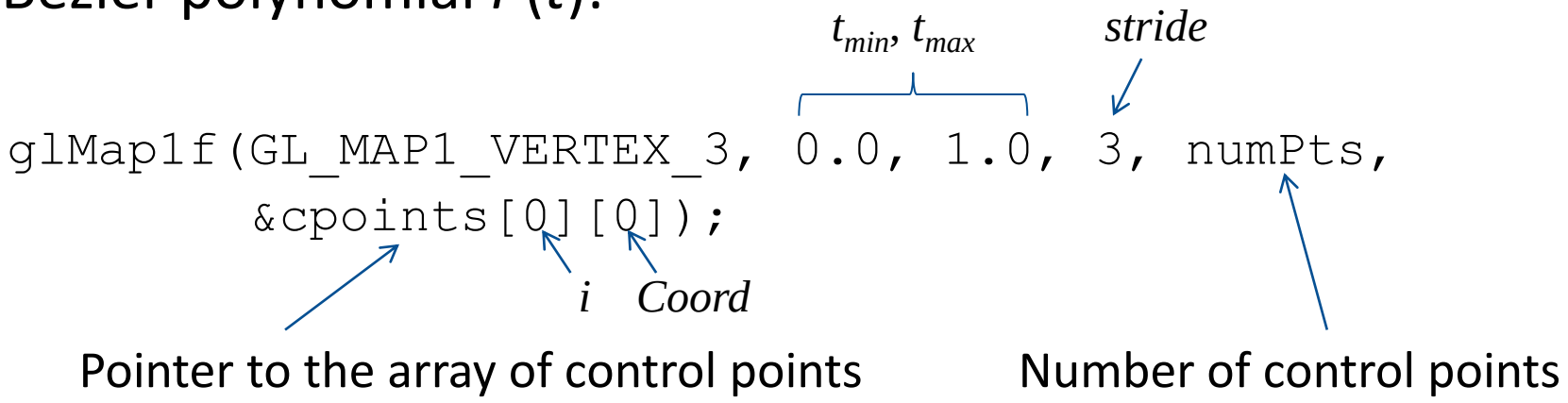
- The Bernstein polynomials form a partition of unity (sum of the polynomial values = 1), and therefore a Bezier curve is always generated using a convex combination of control points.
- The Bezier curve always lies within the convex hull of the control points.
- The Bezier curve passes through the first and the last control points, and has the corresponding edges of the control polygonal line as its tangents.
- The curve is affine invariant: An affine transformation of a Bezier curve is the same as the Bezier curve generated using the transformed control points.

OpenGL Evaluators

- We can view a Bezier equation as a mapping from a set of three-dimensional control points to a one-parameter curve.



- The above mapping is denoted by GL_MAP1_VERTEX_3 and is specified by `glEnable(GL_MAP1_VERTEX_3);`
- Next, specify the parameters required for setting up the Bezier polynomial $P(t)$:



OpenGL Evaluators

- The Bezier curve can be generated in either of the following two ways:

1. Use glEvalCoord1f(t) to get points along the curve.

```
glBegin(GL_LINE_STRIP);
  for(int i=0; i<=100; i++)    //101 pts
  {
    t = (float)i/100.0;  //t from 0 to 1
    glEvalCoord1f(t);
  }
glEnd();
```

2. Evaluate the function at all points on a uniform grid:

```
glMapGrid1f(100, 0.0, 1.0);
glEvalMesh1(GL_LINE, 0, 100);
```

In the above example, the *t* range [0, 1] is divided into 100 intervals (101 points)

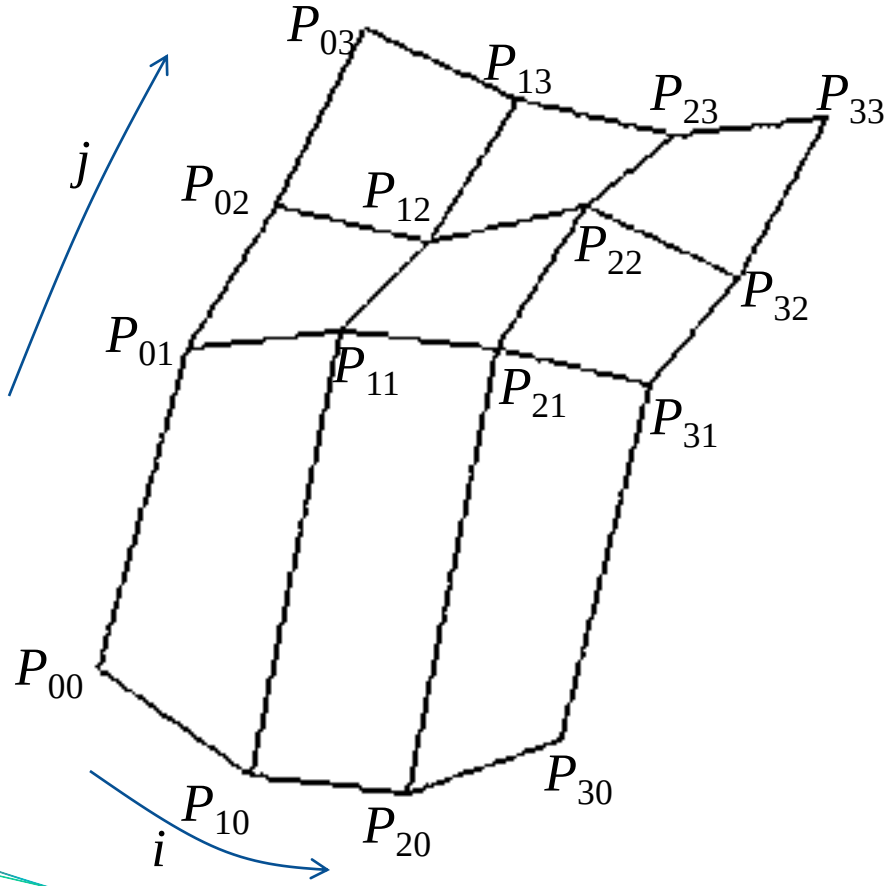
From Curves to Patches

- The method for generating one-parameter Bezier curves $P(t)$ can be extended to get two-parameter Bezier surfaces $P(u, v)$ by using a polygonal grid of control points instead of a polygonal line. Control Points: $\{P_{ij},\ i = 0,1,\ldots m,\quad j = 0,1,\ldots n\}$.
- The generated surface is called a ***Bezier surface patch***.

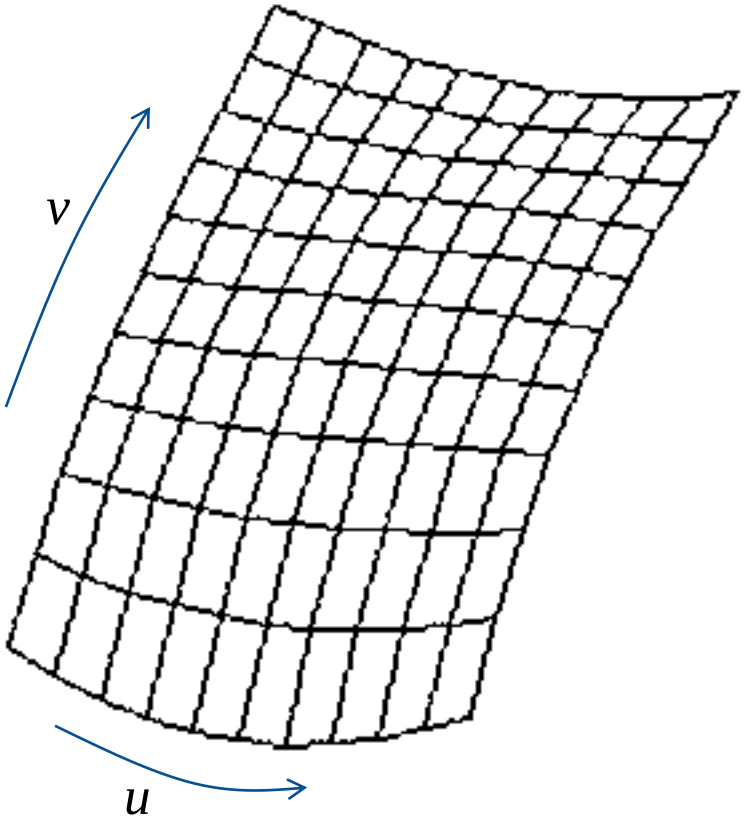
$$p(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} f_i^{(n)}(u)\, f_j^{(m)}(v) P_{ij} \qquad \begin{matrix} 0 \le u \le 1 \\ 0 \le v \le 1 \end{matrix}$$

Bezier Surface Patch

- A 4x4 grid of control points is used to generate a bi-cubic Bezier patch.



Control polygonal grid



Bi-cubic Bezier patch

Bi-cubic Bezier Surface Patch

A *Bi-cubic Bezier surface patch* is generated using two sets of Bernstein polynomials of degree 3: $B_i^3(u)$ and $B_j^3(v)$.

$$p(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} f_i^{(3)}(u)\, f_j^{(3)}(v) P_{ij} \qquad \begin{matrix} 0 \le u \le 1 \\ 0 \le v \le 1 \end{matrix}$$

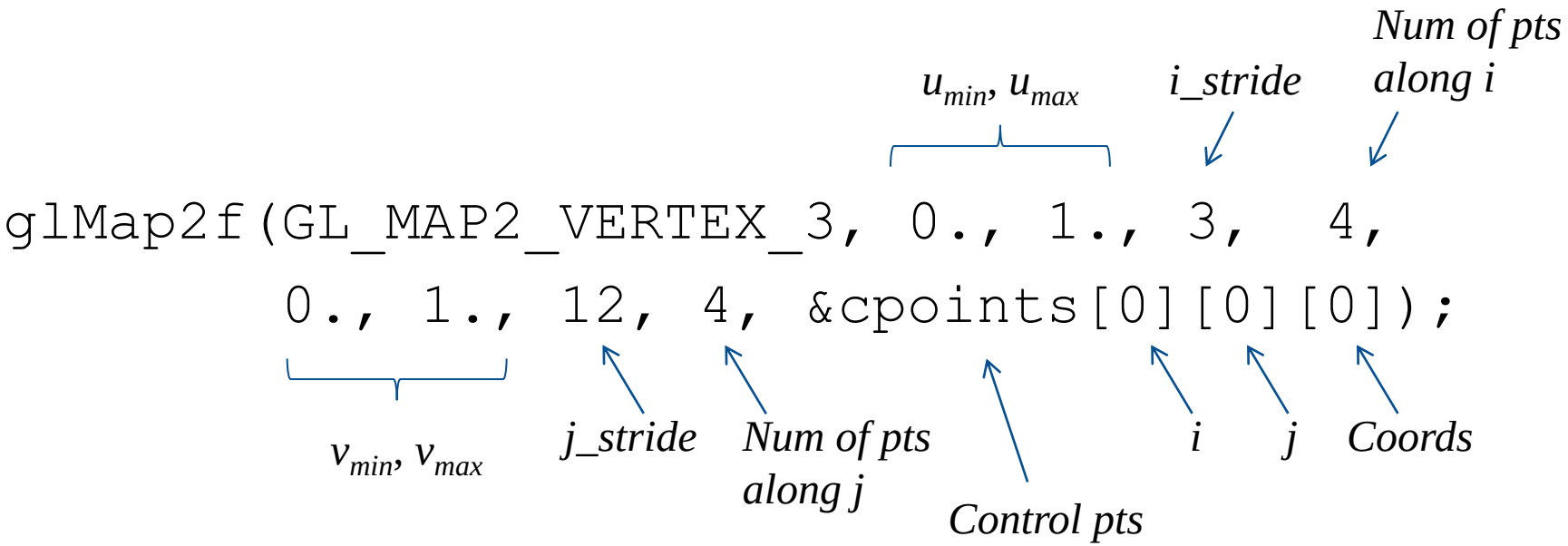
$$\begin{aligned} p(u,v) = {} & (1-u)^3 \left\{(1-v)^3 P_{00} + 3(1-v)^2 v P_{01} + 3(1-v)v^2 P_{02} + v^3 P_{03}\right\} \\ & + 3(1-u)^2 u \left\{(1-v)^3 P_{10} + 3(1-v)^2 v P_{11} + 3(1-v)v^2 P_{12} + v^3 P_{13}\right\} \\ & + 3(1-u)u^2 \left\{(1-v)^3 P_{20} + 3(1-v)^2 v P_{21} + 3(1-v)v^2 P_{22} + v^3 P_{23}\right\} \\ & + u^3 \left\{(1-v)^3 P_{30} + 3(1-v)^2 v P_{31} + 3(1-v)v^2 P_{32} + v^3 P_{33}\right\} \end{aligned}$$

Bezier Surfaces (OpenGL)

- Enable the mapping from three-dimensional control points to a two-parameter surface:

```
glEnable(GL_MAP2_VERTEX_3);
```

- Set up the Bezier polynomial $P(u, v)$: The example below is for a 4x4 grid:



Bezier Surfaces (OpenGL)

- Use any of the following methods to get the surface points:
  1. Use glEvalCoord2f(u, v) to get points along the surface:

```
glBegin(GL_LINE_LOOP);
   glEvalCoord2f(u, v);
   glEvalCoord2f(u+step, v);
   glEvalCoord2f(u+step, v+step);
   glEvalCoord2f(u, v+step);
glEnd();
```

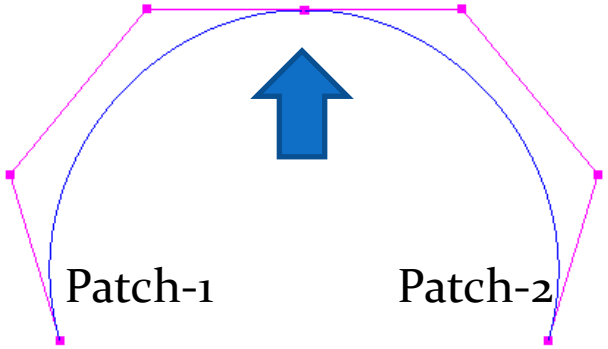
  2. Evaluate the function at all points on a uniform grid:

```
glMapGrid2f(nu, 0.0, 1.0, nv, 0.0, 1.0);
glEvalMesh2(GL_LINE, 0, nu, 0, nv);
```

nu, nv = Number of subdivisions required along u, v directions.

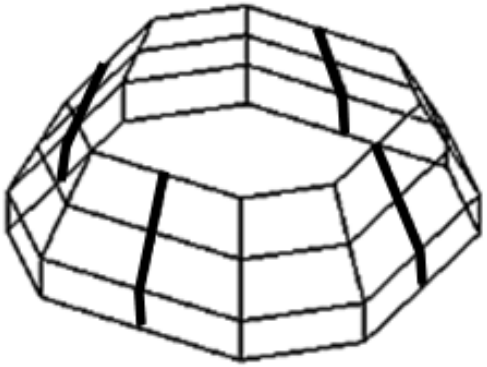
Modelling Complex Shapes

- A coarse description of a shape could be specified using several 4x4 control grids. Each grid is then used to create a Bezier patch.
- We can get a smooth surface *if* the Bezier patches do not have any discontinuity along the edges where two patches meet. This can be achieved by making sure that the corresponding polygonal edges are collinear (for curves) or the corresponding polygons of the grids are coplanar (for surfaces)

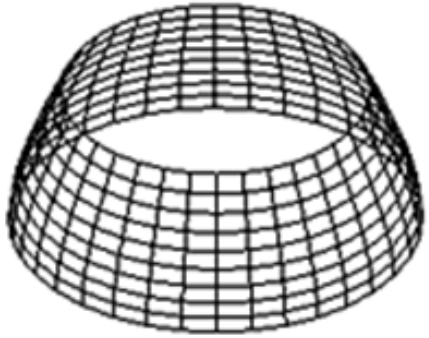


Modelling Complex Shapes

- The Utah Teapot is defined using 32 control polygonal grids each of size 4x4.
- The grids are defined such that polygons on either side of an edge where two grids meet are coplanar.
- Bi-cubic Bezier patches are generated for each grid to get the shape of the teapot.

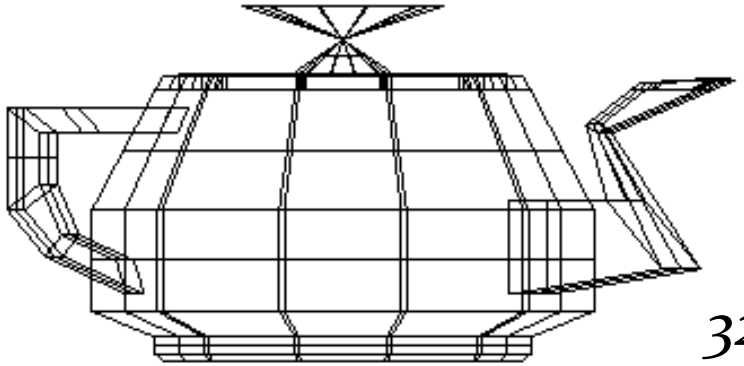


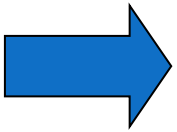
4 Control grids joined together



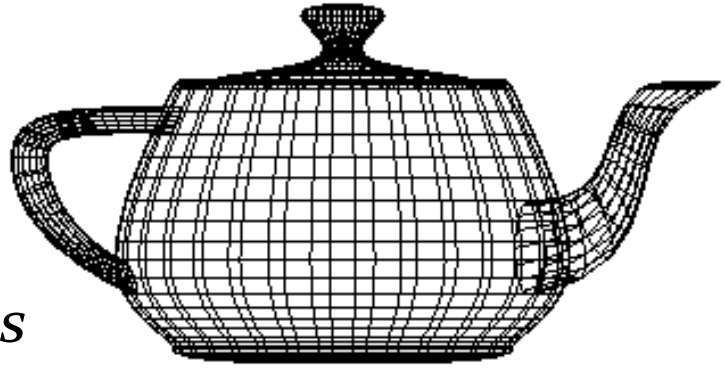
Teapot's upper body

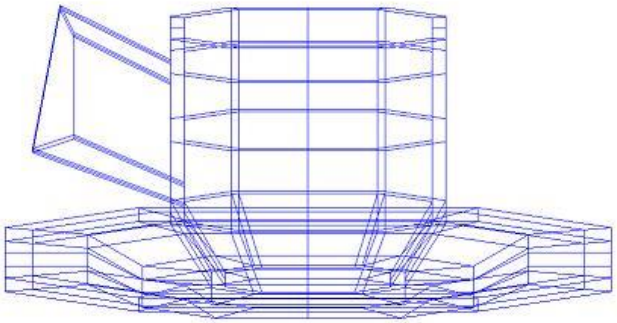
The Teapot Family

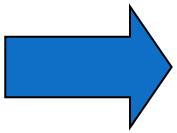




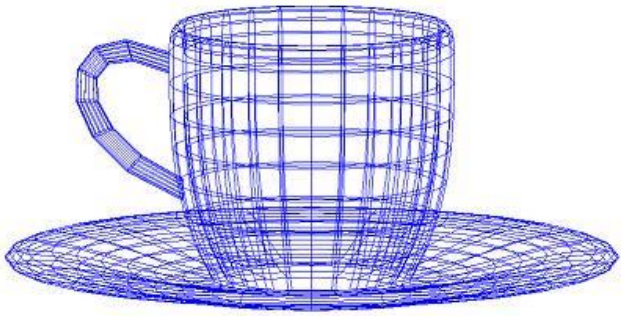
The Teapot
32 Control grids

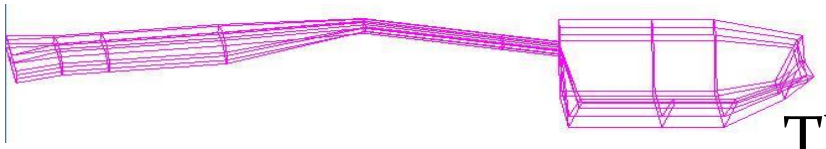


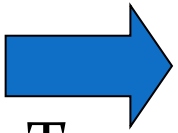




The Teacup
26 Control grids



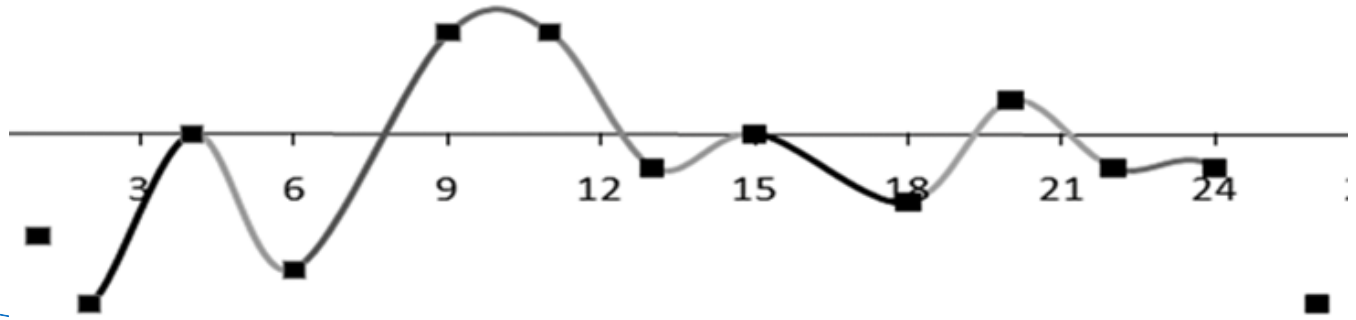




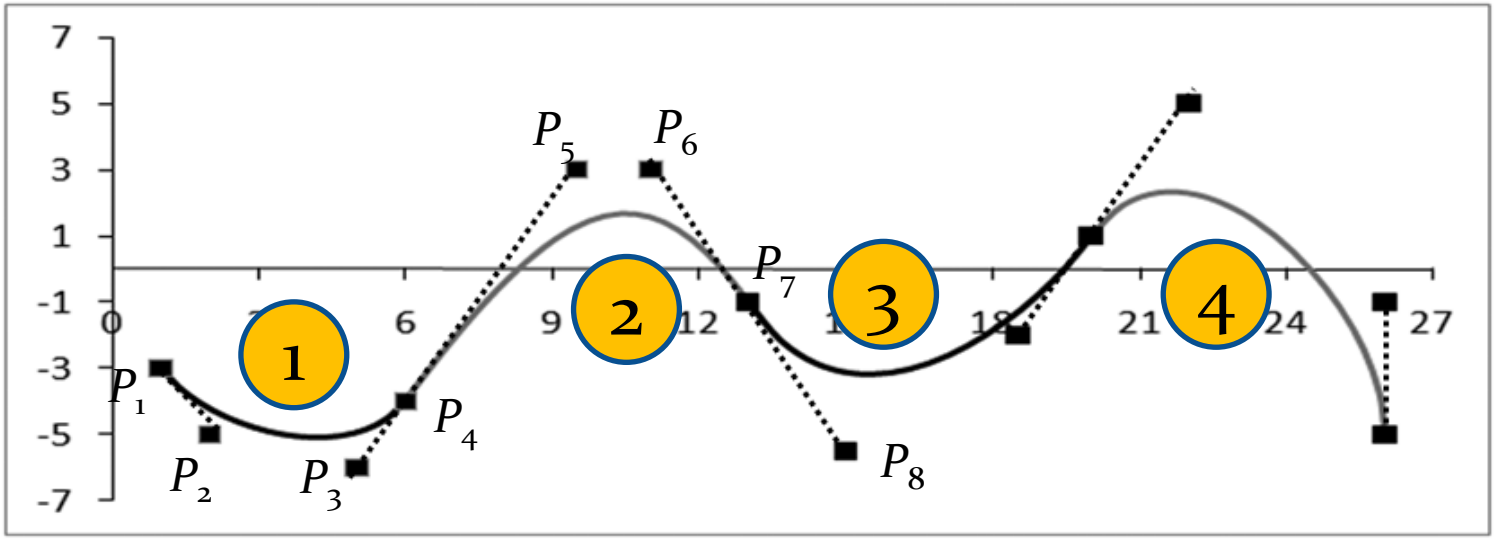
The Teaspoon
16 Control grids

Splines

- Bezier curves or surfaces of a high degree do not allow local control. Changing the position of one control point affects the whole curve.
- We therefore modelled surfaces by connecting together several Bezier patches of low degree.
- Splines are piecewise smooth curves that connect together at points called knots. At the knots, splines are required to satisfy continuity constraints.
- Splines allow fine local control of a shape.



Cubic Bezier Splines



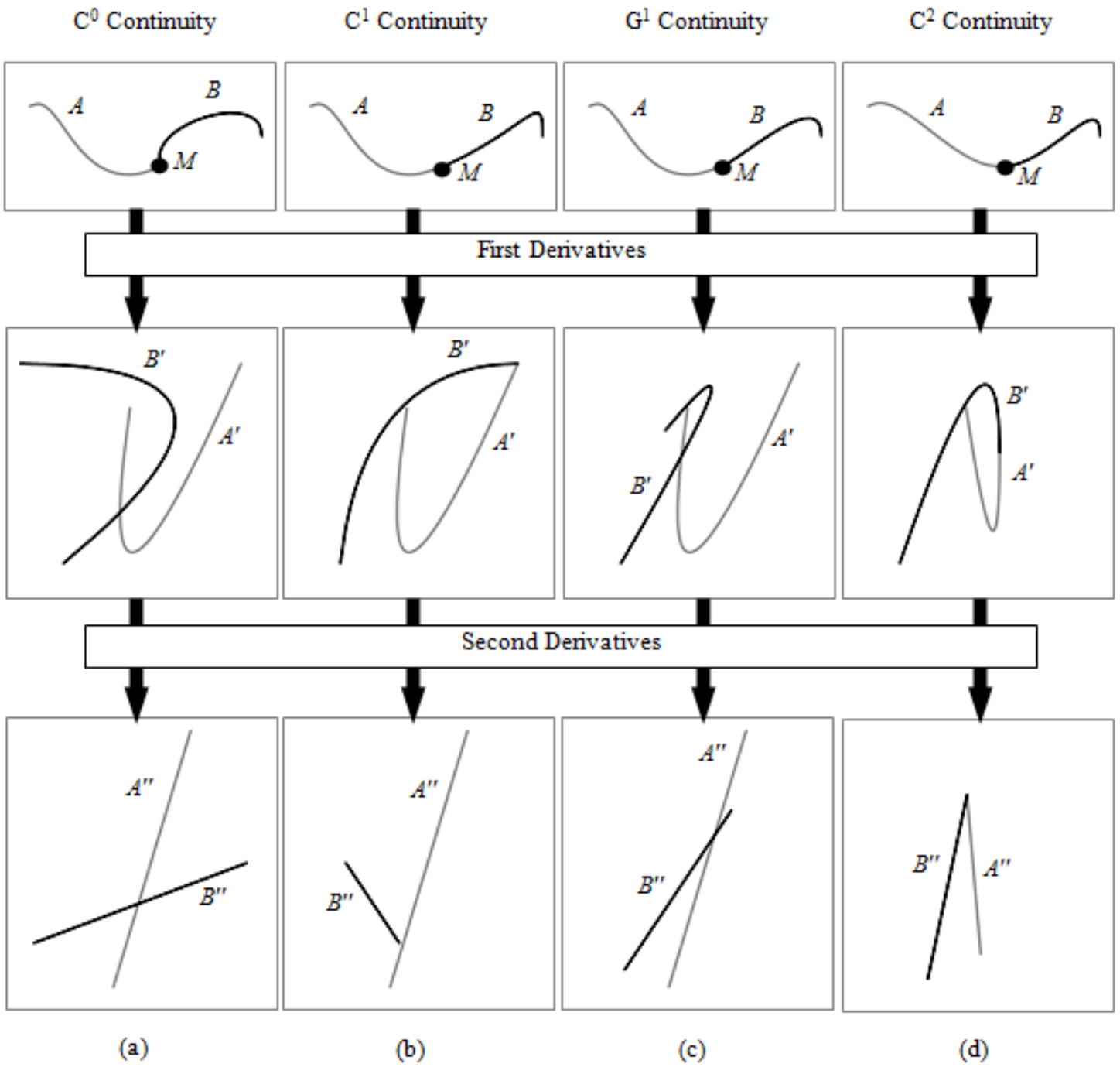
- The above spline is made up of 4 cubic Bezier curves.
- The spline passes through the points P_1, P_4, P_7, P_{10}, P_{13}
- The remaining points control the shape, and may be used to give tangential continuity.

Cubic Parametric Splines

- At the knots, the curves must have tangential continuity.
- Cubic polynomial curves have continuous first and second order derivatives.
- The Cubic Bezier Spline on the previous slide has the following limitations:
  - The user must make sure that the point P_3, P_4, P_5 etc. are collinear.
  - The parameter *t* does not vary continuously over the whole curve.
  - It is not possible to interactively reshape the curve by moving the knots alone.

Parametric Continuity

- Two parametric curves $P_A(t) = (x_A(t), y_A(t), z_A(t))$ and $P_B(t) = (x_B(t), y_B(t), z_B(t))$ are said to have C^0 continuity if they meet at a common point (at a knot M).
- If the tangents to the two curves at M also coincide, then the curves have C^1 continuity.
- The requirement to have the same tangent vector for both curves at a knot can be relaxed to the condition that the tangent vectors are just parallel, with possibly unequal magnitudes. The curves are then said to have a geometric continuity G^1 at the common point M.
- Splines must at least have G^1 continuity at every knot.

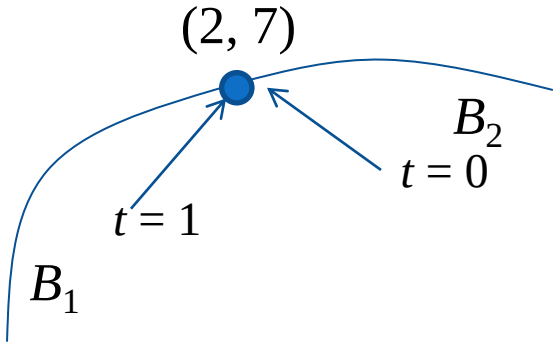

C0 Continuity
C1 Continuity
G1 Continuity
C2 Continuity
A
B
M
First Derivatives
B′
A′
Second Derivatives
A″
B″
(a)
(b)
(c)
(d)

Example-1

- Consider two parametric curves given by

$B_1(t)$: $(2t^2, -3t^2 + 10t)$

$B_2(t)$: $(3t^2 + 4t + 2, -2t^2 + 4t + 7)$



$B_1(1) = B_2(0) = (2, 7)$. The curves meet at (2, 7).

Tangent of B_1: $(4t, -6t + 10)$

Tangent of B_2: $(6t + 4, -4t + 4)$

At $t = 1$, the curve B_1 has the tangent (4, 4)

At $t = 0$, the curve B_2 has the tangent (4, 4)

The curves have the same tangent vector at the knot. The curves therefore have C^1 continuity at (2, 7).

Example-2

- Consider two parametric curves given by

$B_1(t)$: $(3t^2,\ -t^3 + 3t)$

$B_2(t)$: $(-t^3 - 3t^2 + 9t + 3,\ 3t^3 - 6t^2 + 2)$

$B_1(1) = B_2(0) = (3, 2)$. The curves meet at (3, 2).

Tangent of B_1: $(6t,\ -3t^2 + 3)$

Tangent of B_2: $(-3t^2 - 6t + 9,\ 9t^2 - 12t)$

At $t = 1$, the curve B_1 has the tangent (6, 0)

At $t = 0$, the curve B_2 has the tangent (9, 0)

The tangent vectors to the curves are parallel at the knot.

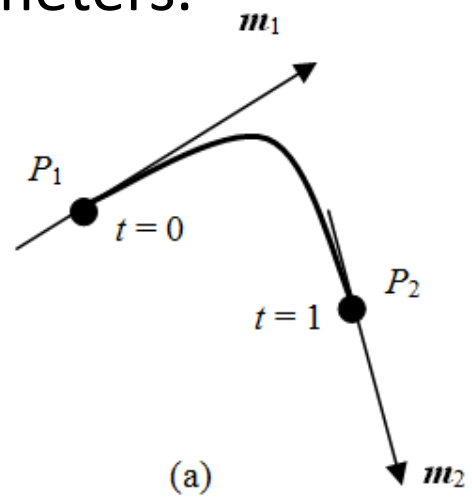
The curves therefore have G^1 continuity at (3, 2).

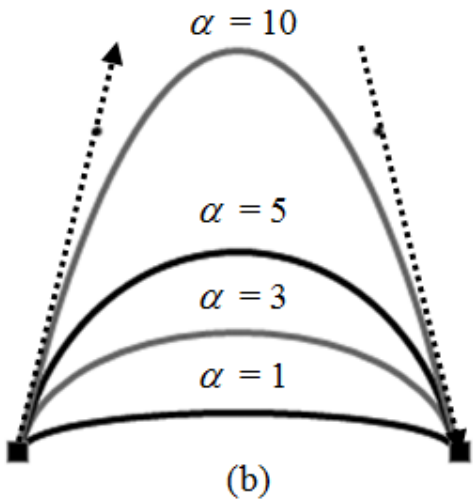
Hermite Splines

- Input: 2 points P_1, P_2, two tangent directions $\boldsymbol{m}_1$, $\boldsymbol{m}_2$, and two constants α_1, α_2.
- Equation of the cubic curve:

$$P(t) = (1 - 3t^2 + 2t^3)\, P_1 + (3t^2 - 2t^3)\, P_2 + (t - 2t^2 + t^3)\, \alpha_1\, \mathbf{m}_1 + (-t^2 + t^3)\, \alpha_2\, \mathbf{m}_2$$

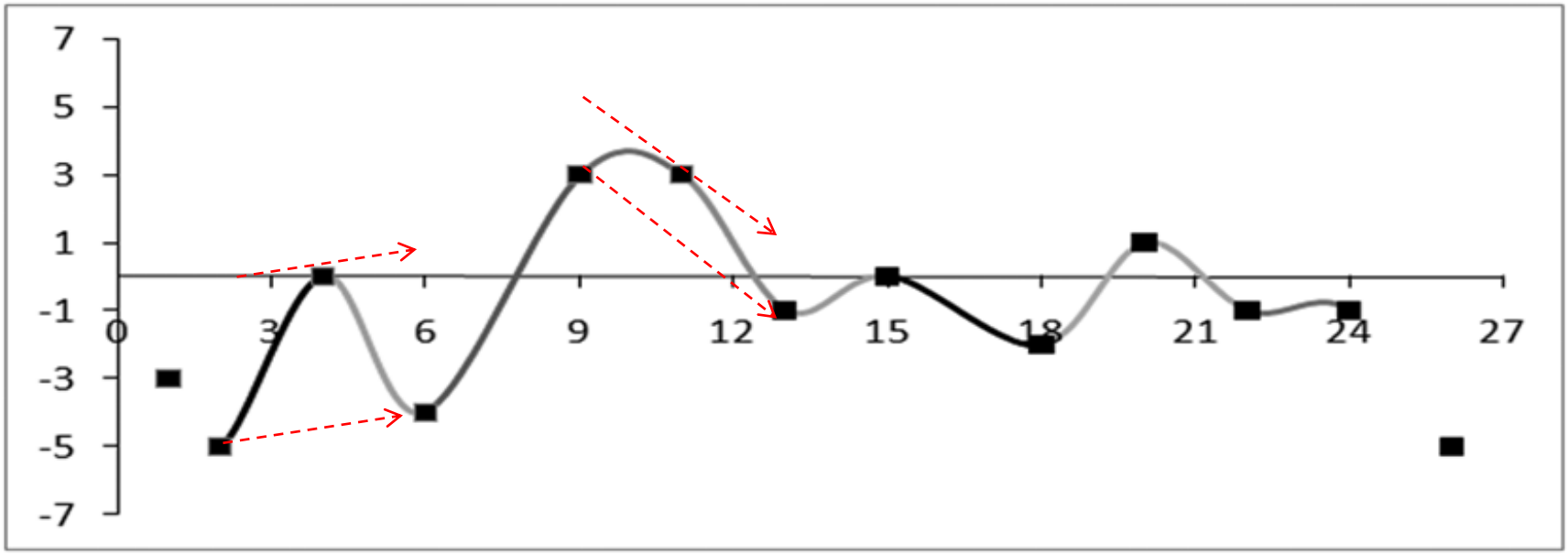
The weights α_1, α_2 of the tangent vectors are called the tension parameters.





Catmull-Rom Splines

- We can combine a series of Hermite curves to obtain a spline with tangential continuity, if we define the tangent vectors as shown below:



- The tangent vector at a point is the same as the vector from the previous point to the next point.

Catmull-Rom Splines

- The spline shown on the previous slide is known as the Catmull-Rom spline if $\alpha_1 = \alpha_2 = 0.5$.
- The primary advantages of Catmull-Rom splines over Bezier splines are:
  - The tangent directions are automatically defined based on control points.
  - The curve passes through all control points except the first and the last points.
  - The control points can be interactively modified without affecting the tangential continuity constraints.
- Splines having the above properties are called cardinal splines. They are commonly used in interactive drawing applications.