Lecture 13

Object representation:
Quadrics
Splines
Bézier patches

(Chapter 8.5-)

Information Coding / Computer Graphics, ISY, LiTH

3D object representation

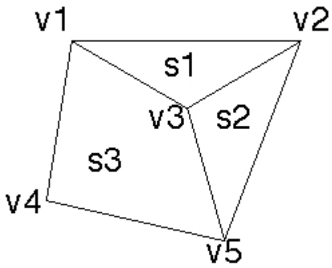
In order of importance:

- Polygonal
- Bi-cubic parametric patches
- Procedural representation, fractals
- Constructive solid geometry
- Implicit representation by quadrics

Also: Volumes, point-based methods...

Polygonal representations

Dominant in real-time graphics
Less suited for off-line rendering



Vertex table

v1 = x1, y1, z1
v2 = x2, y2, z2
v3 = x3, y3, z3
v4 = x4, y4, z4
v5 = x5, y5, z5

Surface table

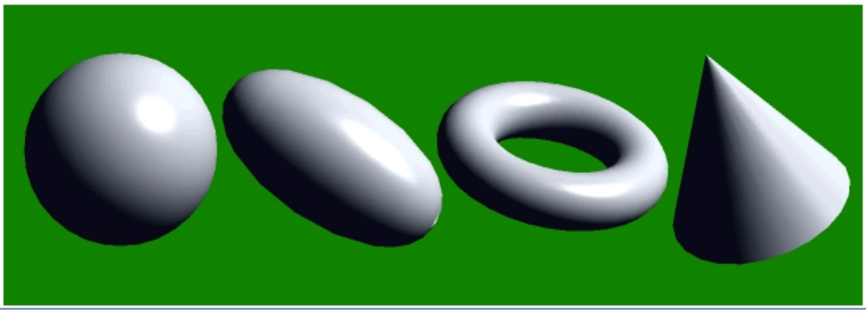
s1 = v1, v2, v3
s2 = v2, v5, v3
s3 = v1, v3, v5, v4

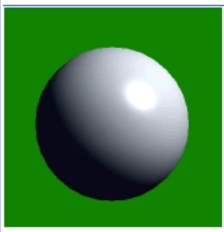
Information Coding / Computer Graphics, ISY, LiTH

Implicit representations: Quadric surfaces

Surfaces represented by second-degree polynomials

Sphere Ellipsoid Torus Cone





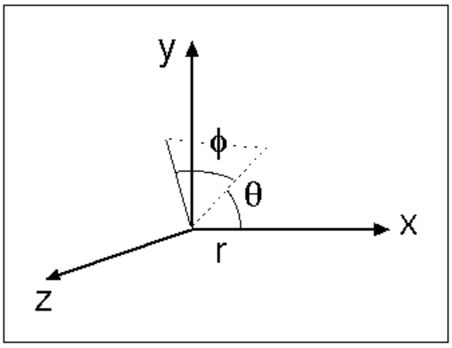
Sphere

Equation:

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

Parametric:

$$x = r \cos\phi \cos\theta$$
$$y = r \cos\phi \sin\theta$$
$$z = r \sin\phi$$



$$-\pi/2 \le \phi \le \pi/2$$
$$-\pi \le \theta \le \pi$$

Ellipsoid

Equation:

$$x^2/r_x^2 + y^2/r_y^2 + z^2/r_z^2 = 1$$

Parametric:

$$x = r_x \cos\phi \cos\theta$$
$$y = r_y \cos\phi \sin\theta$$
$$z = r_z \sin\phi$$

$$-\pi/2 \le \phi \le \pi/2$$
$$-\pi \le \theta \le \pi$$

Torus

Rotate a circle around an axis

Equation:

$$(r - \mathrm{sqrt}(x^2/r_x^2 + y^2/r_y^2))^2 + z^2/r_z^2 = 1$$

Parametric:

$$x = r_x\,(r + \cos\phi)\cos\theta$$
$$y = r_y\,(r + \cos\phi)\sin\theta$$
$$z = r_z \sin\phi$$

$-\pi \le \phi \le \pi$
$-\pi \le \theta \le \pi$

Quadric surfaces

Limited possibilities. Slightly more freedom can be achieved with “superquadrics”

Many quadric surfaces are hard to rotate freely

Rendering packages replace them with meshes (polygons or curved surfaces)

Are quadrics outdated?

Superquadrics:

Example: Superellipse

$$x = r_x \cos^s\theta$$
$$y = r_y \sin^s\theta$$

s = 0.1

$$(x/r_x)^{2/s} + (y/r_y)^{2/s} = 1$$

s = 5.0

s = 0.5 s = 1.0 s = 1.5 s = 2.0 s = 2.5

Quadric surfaces in OpenGL

Calls that approximate quadric surfaces by polygons

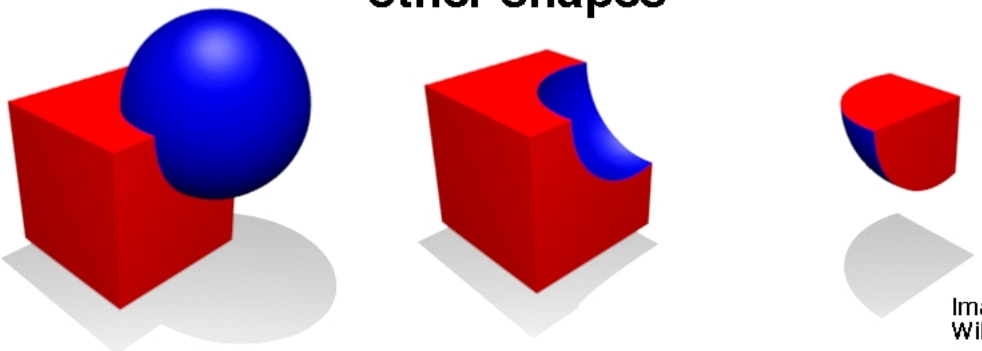
glutWireSphere/glutSolidSphere

glutWireCone/glutSolidCone

glutWireTorus/glutSolidTorus

Constructive Solid Geometry

Define shapes by Boolean operations on other shapes



Images from Wikipedia

Union (a or b)

Difference (a and not b)

Intersection (a and b)

Constructive Solid Geometry

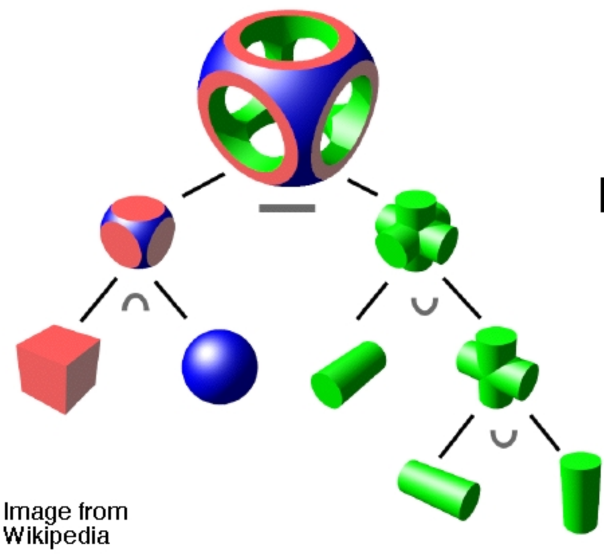


Image from Wikipedia

Good shapes in - somewhat useful.

Limited shapes in - limited results.

Splines

Originally a drafting tool to create a smooth curve

In compute graphics: a curve built from sections, each described by 3rd degree polynomial.

Very common in non-real-time graphics, both 2D and 3D!

Useful also for real-time.

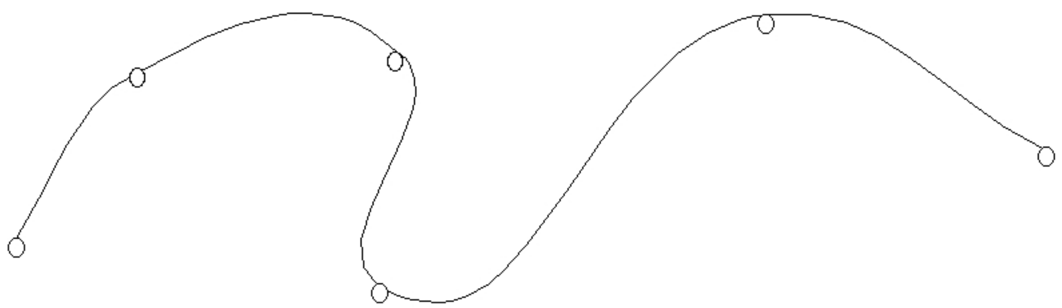
Information Coding / Computer Graphics, ISY, LiTH

Applications of splines

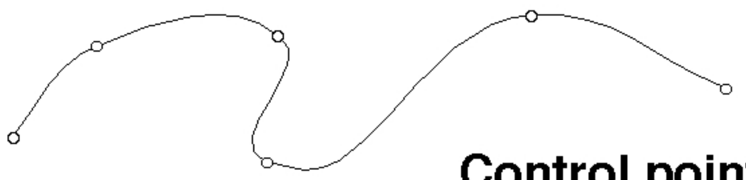
- Designing smooth curves (common in 2D illustrations)
- Modelling smooth surfaces
- Representating of smooth surfaces (converted to polygons in real-time)
- Animation paths

Control points

A spline is specified by a set of control points.

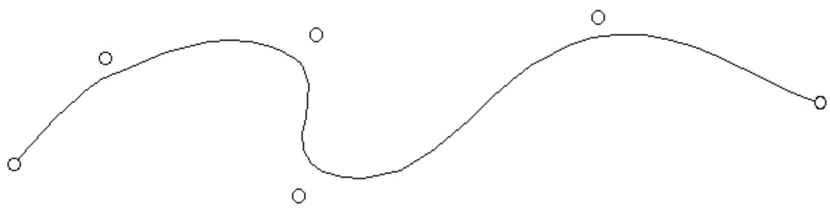


Interpolation spline



Control points on the curve.

Approximation spline



Control points not on the curve.

Parametric representation

$$x = x(u)$$

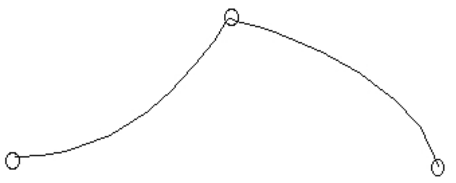
$$y = y(u) \qquad u_1 \leq u \leq u_2$$

$$z = z(u)$$

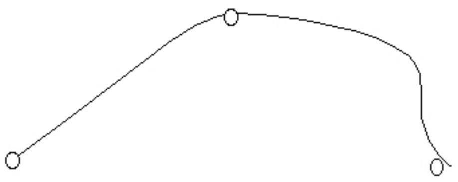
A set of functions for each coordinate

Natural splines

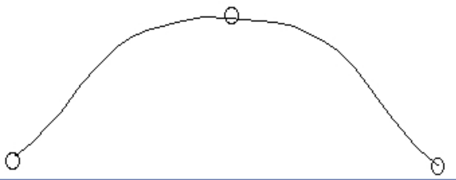
Parametric continuity



C^0 = continuous position
= the curves meet



C^1 = continuous direction
= the curves meet at same angle



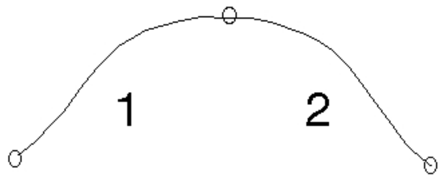
C^2 = continuous curvature
= the curves meet at same bend

Specification of splines by functions

$$x_1(u) = a_{x1}u^3 + b_{x1}u^2 + c_{x1}u + d_{x1}$$

$$y_1(u) = a_{y1}u^3 + b_{y1}u^2 + c_{y1}u + d_{y1}$$

$$z_1(u) = a_{z1}u^3 + b_{z1}u^2 + c_{z1}u + d_{z1}$$



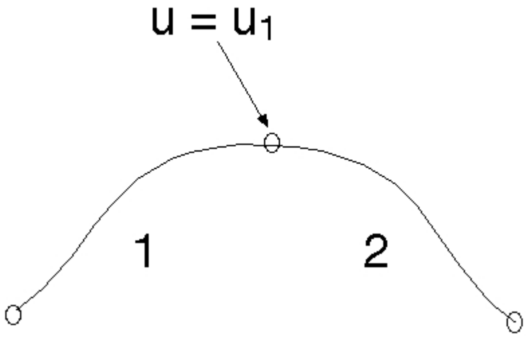
$$x_2(u) = a_{x2}u^3 + b_{x2}u^2 + c_{x2}u + d_{x2}$$

$$y_2(u) = a_{y2}u^3 + b_{y2}u^2 + c_{y2}u + d_{y2}$$

$$z_2(u) = a_{z2}u^3 + b_{z2}u^2 + c_{z2}u + d_{z2}$$

Information Coding / Computer Graphics, ISY, LiTH

Parametric continuity



C^0:

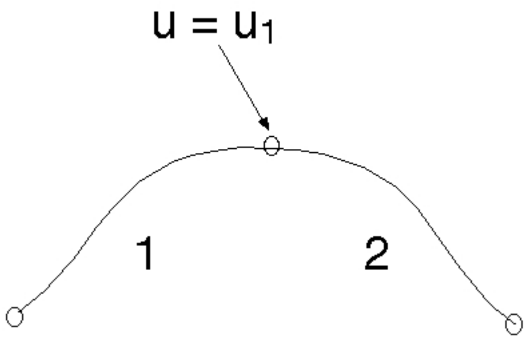
$$x_1(u_1) = x_2(u_1)$$
$$y_1(u_1) = y_2(u_1)$$
$$z_1(u_1) = z_2(u_1)$$

C^1:

$$x'_1(u_1) = x'_2(u_1)$$
$$y'_1(u_1) = y'_2(u_1)$$
$$z'_1(u_1) = z'_2(u_1)$$

C1: 6 equations per vertex, 12 coefficients per section

Geometric continuity



G^0:

$$x_1(u_1) = x_2(u_1)$$
$$y_1(u_1) = y_2(u_1)$$
$$z_1(u_1) = z_2(u_1)$$

G^1:

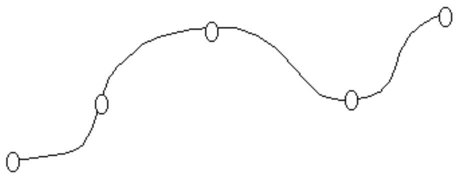
$$x'_1(u_1) = k*x'_2(u_1)$$
$$y'_1(u_1) = k*y'_2(u_1)$$
$$z'_1(u_1) = k*z'_2(u_1)'$$

for some k

Essentially one less constraint

Natural cubic splines

C^2 continuity
Solve the entire equation system!



$$x_1(u) = a_{x1}u^3 + b_{x1}u^2 + c_{x1}u + d_{x1}$$

n sections
n+1 control points

$$x'_i(u_i) = x'_{i+1}(u_i)$$
$$x''_i(u_i) = x''_{i+1}(u_i)$$
$$x_i(u_i) = x_i$$
$$x_{i+1}(u_i) = x_i$$

4n coefficients
(n-1)*4 boundary conditions
+ 2 for the end points – 2 free parameters

Natural splines

Drawbacks:

Complex equation system!
Minor problem.

Moving one point changes all sections.
Major problem!

Blending functions

Rewrite parametric form to a set of polynomials, one polynomial for each control point

Approximation splines

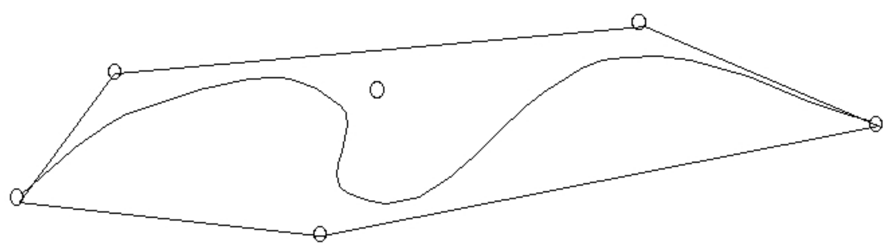
Use a set of blending functions to blend together control points to points on the curve

Bézier curves
B-splines
NURBS

Information Coding / Computer Graphics, ISY, LiTH

Common demand on approximations splines:

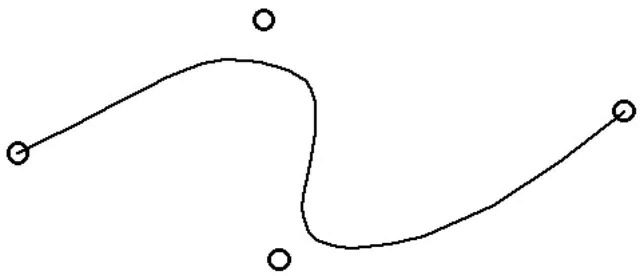
Stay within the convex hull of the control points!



Convex hull = minimal convex polygon enclosing a specified set of points

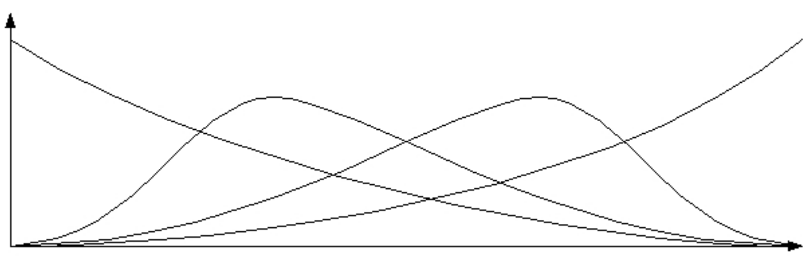
Bézier curves

Typically uses 4 control points per section



Bézier curves

The 4 points are blended together using 4 blending functions



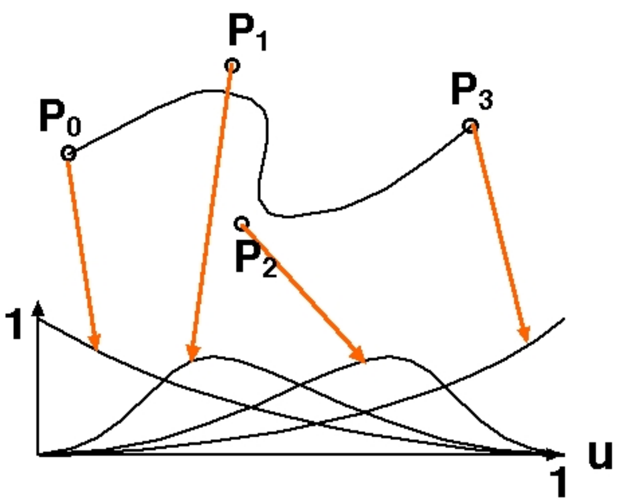
Bézier curves

Blending functions:
Bernstein polynomials

$BEZ_{0,3} = (1-u)^3$
$BEZ_{1,3} = 3u(1-u)^2u$
$BEZ_{2,3} = 3(1-u)u^2$
$BEZ_{3,3} = u^3$

The sum is 1 for any u

Information Coding / Computer Graphics, ISY, LiTH



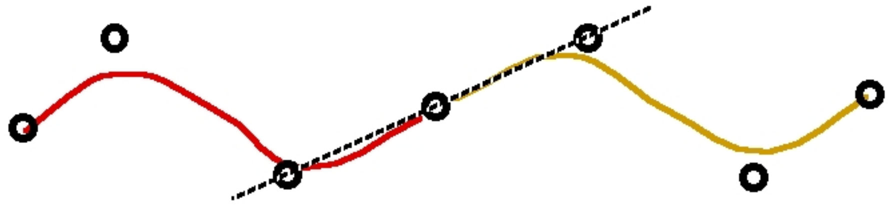
$BEZ_{0,3} = (1-u)^3$
$BEZ_{1,3} = 3u(1-u)^2u$
$BEZ_{2,3} = 3(1-u)u^2$
$BEZ_{3,3} = u^3$

$$P(u) = P_0*(1-u)^3 + P_1*3u(1-u)^2 + P_2*3(1-u)u^2 + P_3*u^3$$
$$= \sum_{i=0}^{3} P_i * BEZ_{i,3}(u)$$

Fitting together sections

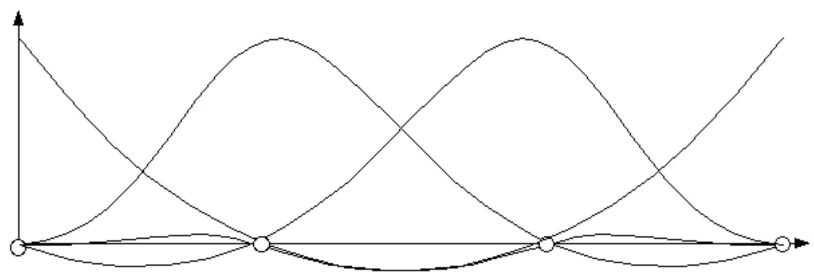
G_0 continuity: just fit the points

G_1 continuity: Make sure the tangents are equal along the edge.
Simple method: Put 3 points in a line

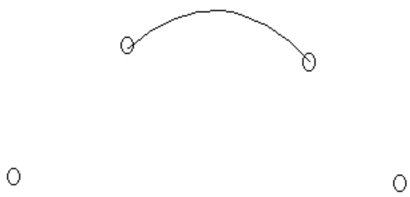


Blending functions for interpolation spline

The points are *blended* together using blending functions



Cardinal splines
Catmull-Rom splines



Interpolation spline

Specified *only* by control points

Calculated from 4 closest control points

A tension parameter t can adjust the shape

t = 0 => Catmull-Rom

Catmull-Rom splines,
Matrix form

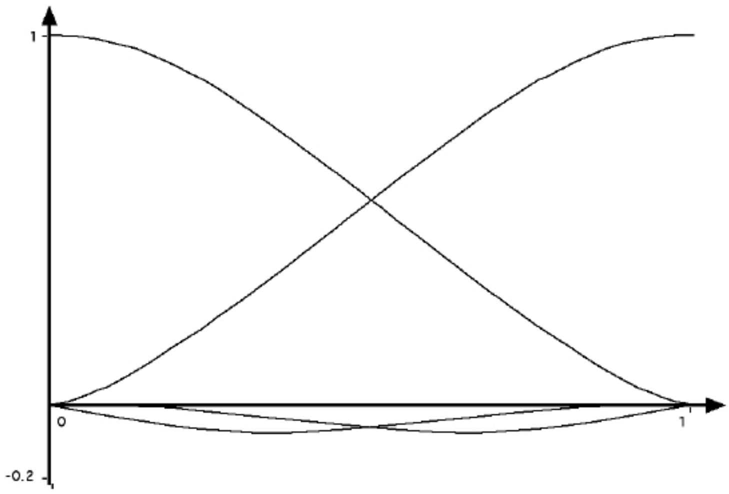
$$P(u) = [\, u^3 \; u^2 \; u \; 1 \,] \begin{bmatrix} -1/2 & 3/2 & -3/2 & 1/2 \\ 1 & -5/2 & 2 & -1/2 \\ -1/2 & 0 & 1/2 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_{k-1} \\ p_k \\ p_{k+1} \\ p_{k+2} \end{bmatrix}$$

$$\begin{aligned} P(u) = \; & p_{k-1}\,(-u^3/2 + u^2 - u/2) + \\ & p_k\,(3u^3/2 - 5u^2/2 + 1) + \\ & p_{k+1}\,(-3u^3/2 + 2u^2 + u/2) + \\ & p_{k+2}\,(u^3/2 - u^2/2) \end{aligned}$$

$$= p_{k-1}{*}CAR_0(u) + p_k\,{*}CAR_1(u) + p_{k+1}{*}CAR_2(u) + p_{k+2}{*}CAR_3(u)$$

Catmull-Rom splines,

Blending functions



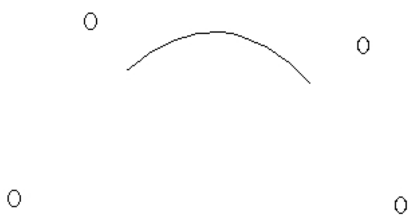
Information Coding / Computer Graphics, ISY, LiTH

B-splines

B = basis function

Uniform cubic B-spline

Approximating spline similar to Catmull-Rom; all control points have the same role (unlike Bézier).



NURBs/NURBS

Non-Uniform Rational B-spline.

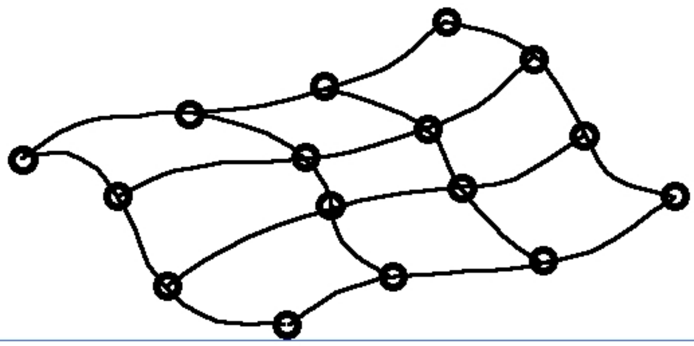
Popular in CAD programs.

Can exactly represent all quadric curves.

Bézier surfaces

A surface is built from a set of Bézier patches

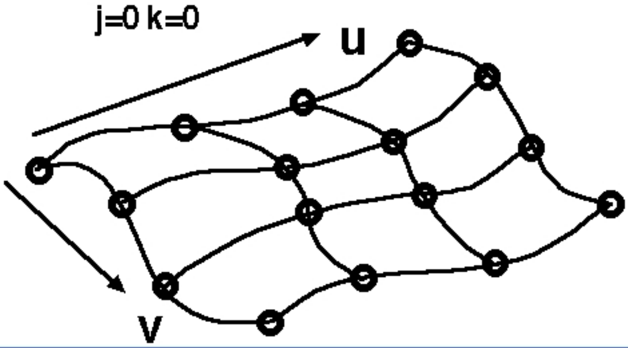
A Bézier patch consists of 16 control points in a 4x4 grid



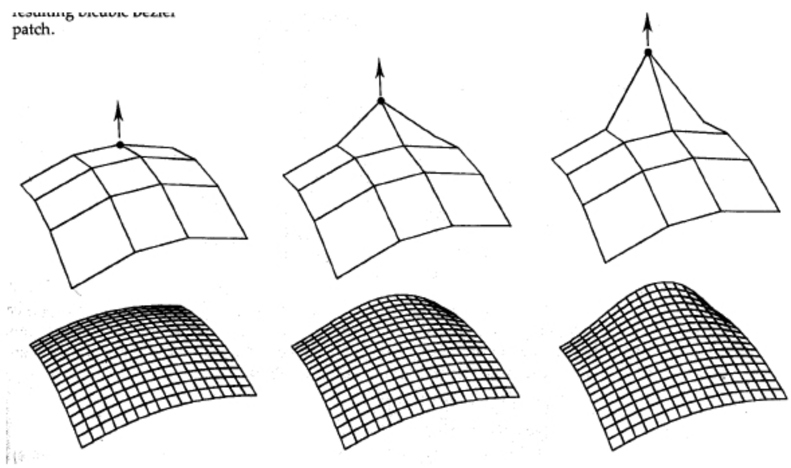
Bézier surfaces

Blending of the 16 control points as a 2-dimensional sum

$$P(u,v) = \sum_{j=0}^{3} \sum_{k=0}^{3} p_{j,k}\ BEZ_{j,3}(v)\ BEZ_{k,3}(u)$$



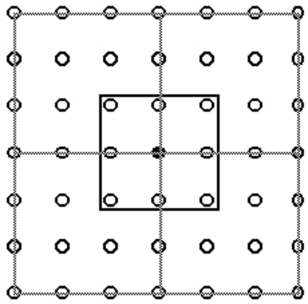
Bézier surface example



Fitting together patches

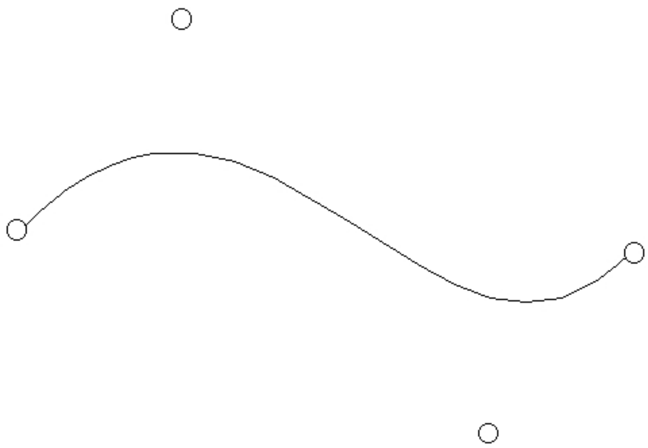
Fit in both u and v direction

Make a 3x3 “joystick” at each corner



Evaluators

Built-in functions for drawing Bezier curves. It calculates the curve with the desired density, and creates line segments.



Ingemar
Ragnemalm
ingis@isy.liu.se

Evaluators

Configure with glMap

Evaluate element by element with glEvalCoord

or all at once with glEvalMesh

```
glMap1f(GL_MAP1_VERTEX_3, u0, u1, 3, 4, &data2[0][0]);
glEnable(GL_MAP1_VERTEX_3);
glBegin(GL_LINE_STRIP);
for (int i = 0; i <= 20; i++)
  glEvalCoord1f(u0 + i*(u1-u0)/20);
glEnd();
```

Control points

Evaluation, specifies vertices

Evaluators

glEvalMesh practical if uniforms steps are desired:

```
glMap1f(GL_MAP1_VERTEX_3, u0, u1, 3, 4, &data2[0][0]);
glEnable(GL_MAP1_VERTEX_3);
glMapGrid1f(20, 0, 1);
glEvalMesh1(GL_LINE, 0, 20);
```

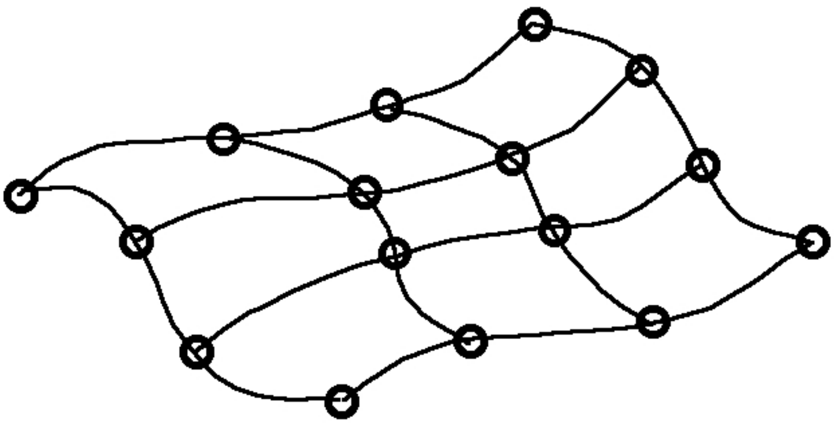
Which interval?

The entire loop in one call (two, actually)

Evaluators

Same thing in 2D!

```
glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4, 0, 1, 12, 4, &data2d[0][0][0]);
glEnable(GL_MAP2_VERTEX_3);
glMapGrid2f(20, 0, 1, 20, 0, 1);
glEvalMesh2(GL_FILL, 0, 20, 0, 20);
```



Evaluators

Obviously useful for modelling smooth shapes.

Easy to implement level-of-detail

Good for e.g. modelling cloth.

Can give performance advantages since the calculations can be carried out by the GPU

Evaluating polynomials

Important problem for efficient spline calculations.

1) Horner's Rule

2) Forward-difference calculations