Parametric Curves

Adam Finkelstein & Tim Weyrich
Princeton University
COS 426, Spring 2008

1

3D Object Representations

- Points
  - Range image
  - Point cloud
- Surfaces
  - Polygonal mesh
  - Subdivision
  - Parametric
  - Implicit
- Solids
  - Voxels
  - BSP tree
  - CSG
  - Sweep
- High-level structures
  - Scene graph
  - Application specific

2

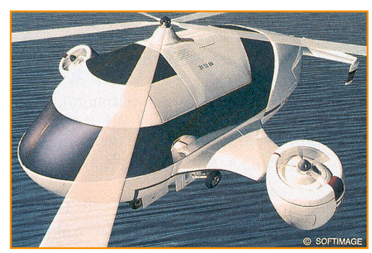
3D Object Representations

- Points
  - Range image
  - Point cloud
- Surfaces
  - Polygonal mesh
  - Subdivision
  - Parametric
  - Implicit
- Solids
  - Voxels
  - BSP tree
  - CSG
  - Sweep
- High-level structures
  - Scene graph
  - Application specific

3

Parametric Surfaces

- Applications
  - Design of smooth surfaces in cars, ships, etc.



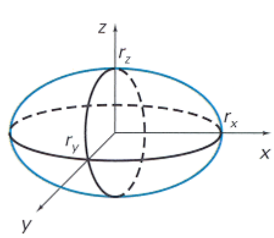
H&B Figure 10.46

4

Parametric Surfaces

- Boundary defined by parametric functions:
  - $x = f_x(u,v)$
  - $y = f_y(u,v)$
  - $z = f_z(u,v)$
- Example: ellipsoid

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

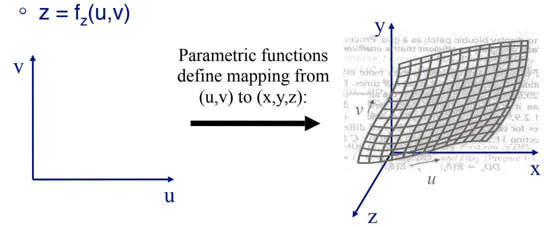


H&B Figure 10.10

5

Parametric Surfaces

- Boundary defined by parametric functions:
  - $x = f_x(u,v)$
  - $y = f_y(u,v)$
  - $z = f_z(u,v)$



FvDFH Figure 11.42

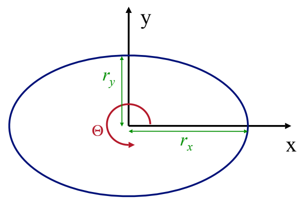
6

Parametric Curves

- Boundary defined by parametric functions:
  - $x = f_x(u)$
  - $y = f_y(u)$

- Example: ellipse

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



H&B Figure 10.10

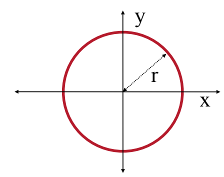
Implicit curves

An implicit curve in the plane is expressed as:

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

Example: a circle with radius r centered at origin:

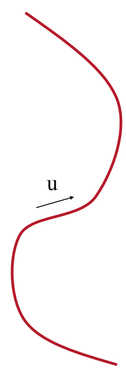
$$x^2 + y^2 - r^2 = 0$$



Parametric curves

How can we define arbitrary curves?

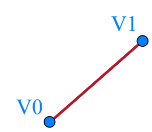
$$x = f_x(u)$$
$$y = f_y(u)$$



Parametric curves

How can we define arbitrary curves?

$$x = f_x(u)$$
$$y = f_y(u)$$



Use functions that “blend” control points

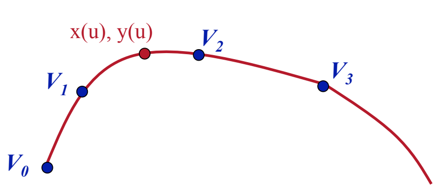
$$x = f_x(u) = V0_x*(1 - u) + V1_x*u$$
$$y = f_y(u) = V0_y*(1 - u) + V1_y*u$$

Parametric curves

More generally:

$$x(u) = \sum_{i=0}^{n} B_i(u) * Vi_x$$

$$y(u) = \sum_{i=0}^{n} B_i(u) * Vi_y$$



Parametric curves

What B(u) functions should we use?

$$x(u) = \sum_{i=0}^{n} B_i(u) * Vi_x$$

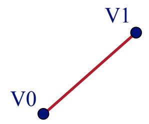
$$y(u) = \sum_{i=0}^{n} B_i(u) * Vi_y$$

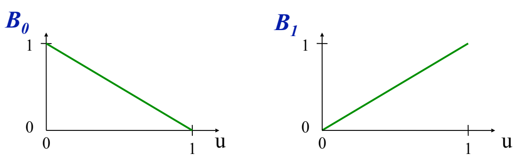
Parametric curves

What B(u) functions should we use?

$$x(u) = \sum_{i=0}^{n} B_i(u) * Vi_x$$

$$y(u) = \sum_{i=0}^{n} B_i(u) * Vi_y$$



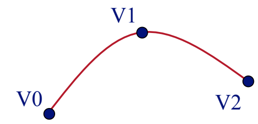


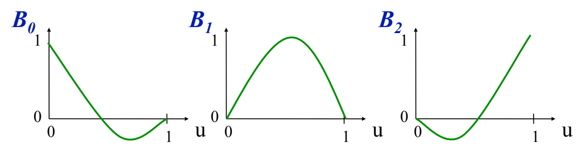
Parametric curves

What B(u) functions should we use?

$$x(u) = \sum_{i=0}^{n} B_i(u) * Vi_x$$

$$y(u) = \sum_{i=0}^{n} B_i(u) * Vi_y$$



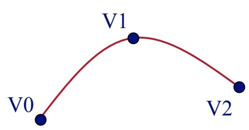


Parametric Polynomial Curves

- Blending functions are polynomials:

$$x(u) = \sum_{i=0}^{n} B_i(u) * Vi_x$$

$$y(u) = \sum_{i=0}^{n} B_i(u) * Vi_y$$



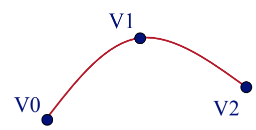
- Advantages of polynomials
  - Easy to compute
  - Infinitely continuous
  - Easy to derive curve properties

$$B_i(u) = \sum_{j=0}^{m} a_j u^j$$

Parametric Polynomial Curves

- Blending functions are polynomials:

$$Q(u) = \sum_{i=0}^{n} B_i(u) * V_i$$

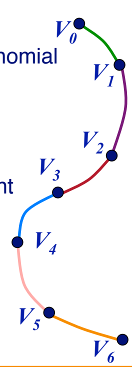


- Advantages of polynomials
  - Easy to compute
  - Infinitely continuous
  - Easy to derive curve properties

$$B_i(u) = \sum_{j=0}^{m} a_j u^j$$

Piecewise Parametric Polynomial Curves

- Splines:
  - Split curve into segments
  - Each segment defined by low-order polynomial blending subset of control vertices
- Motivation:
  - Provides control & efficiency
  - Same blending function for every segment
  - Prove properties from blending functions
- Challenges
  - How choose blending functions?
  - How determine properties?



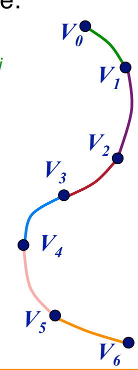
Goals

- Some properties we might like to have:
  - Local control
  - Interpolation
  - Continuity

$$B_i(u) = \sum_{j=0}^{m} a_j u^j$$

Blending functions determine properties

Properties determine blending functions



Splines

- Mathematical way to express curves
- Motivated by "loftsman's spline"
  - Long, narrow strip of wood/plastic
  - Used to fit curves through specified data points
  - Shaped by lead weights called "ducks"
  - Gives curves that are "smooth" or "fair"
- Have been used to design:
  - Automobiles
  - Ship hulls
  - Aircraft fuselage/wing

Cubic Splines

- Splines covered in this lecture
  - Cubic B-Spline
  - Cubic Bezier
- There are many others

Cubic Splines

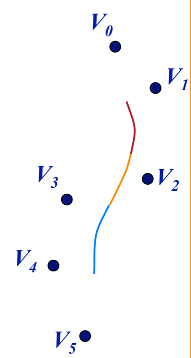
- Splines covered in this lecture
  - ➢ Cubic B-Spline
  - Cubic Bezier
- There are many others

Properties determine blending functions

Blending functions determine properties

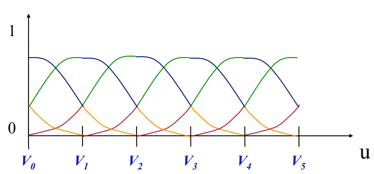
Cubic B-Splines

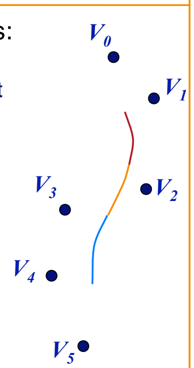
- Properties:
  - Local control
  - C^2 continuity
  - Approximating



Cubic B-Spline Blending Functions

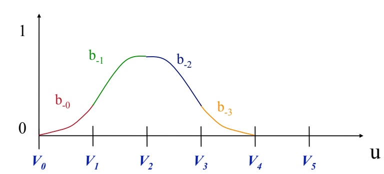
- Properties imply blending functions:
  - Cubic polynomials
  - Four control vertices affect each point
  - C^2 continuity

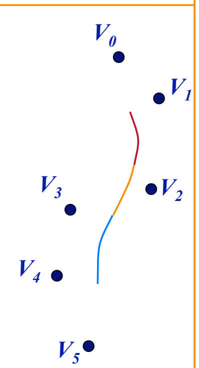




Cubic B-Spline Blending Functions

- How derive blending functions?
  - Cubic polynomials
  - Local control
  - C^2 continuity

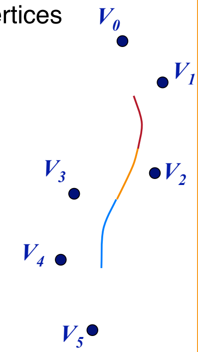




Cubic B-Spline Blending Functions

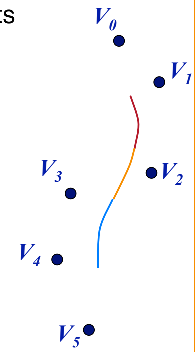
- Four cubic polynomials for four vertices
  - 16 variables (degrees of freedom)
  - Variables are a_i, b_i, c_i, d_i for four blending functions

$$b_{-0}(u) = a_0u^3 + b_0u^2 + c_0u^1 + d_0$$
$$b_{-1}(u) = a_1u^3 + b_1u^2 + c_1u^1 + d_1$$
$$b_{-2}(u) = a_2u^3 + b_2u^2 + c_2u^1 + d_2$$
$$b_{-3}(u) = a_3u^3 + b_3u^2 + c_3u^1 + d_3$$



Cubic B-Spline Blending Functions

- C2 continuity implies 15 constraints
  - Position of two curves same
  - Derivative of two curves same
  - Second derivatives same



Cubic B-Spline Blending Functions

Fifteen continuity constraints:

$0 = b_{-0}(0)$	$0 = b_{-0}'(0)$	$0 = b_{-0}''(0)$
$b_{-0}(1) = b_{-1}(0)$	$b_{-0}'(1) = b_{-1}'(0)$	$b_{-0}''(1) = b_{-1}''(0)$
$b_{-1}(1) = b_{-2}(0)$	$b_{-1}'(1) = b_{-2}'(0)$	$b_{-1}''(1) = b_{-2}''(0)$
$b_{-2}(1) = b_{-3}(0)$	$b_{-2}'(1) = b_{-3}'(0)$	$b_{-2}''(1) = b_{-3}''(0)$
$b_{-3}(1) = 0$	$b_{-3}'(1) = 0$	$b_{-3}''(1) = 0$

One more convenient constraint:

$$b_{-0}(0) + b_{-1}(0) + b_{-2}(0) + b_{-3}(0) = 1$$

Cubic B-Spline Blending Functions

- Solving the system of equations yields:

$$b_{-3}(u) = -\tfrac{1}{6}u^3 + \tfrac{1}{2}u^2 - \tfrac{1}{2}u + \tfrac{1}{6}$$
$$b_{-2}(u) = \tfrac{1}{2}u^3 - u^2 + \tfrac{2}{3}$$
$$b_{-1}(u) = -\tfrac{1}{2}u^3 + \tfrac{1}{2}u^2 + \tfrac{1}{2}u + \tfrac{1}{6}$$
$$b_{-0}(u) = \tfrac{1}{6}u^3$$

Cubic B-Spline Blending Functions

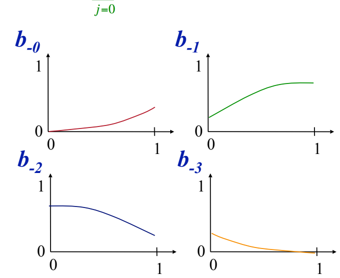
- In matrix form:

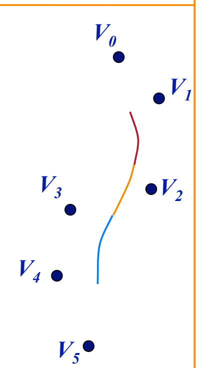
$$Q(u) = \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} \frac{1}{6} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{pmatrix} \begin{pmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \end{pmatrix}$$

Cubic B-Spline Blending Functions

In plot form:

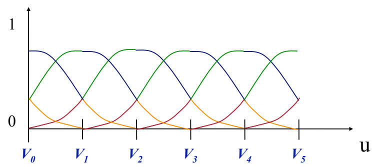
$$B_i(u) = \sum_{j=0}^{m} a_j u^j$$

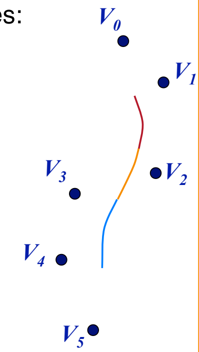




Cubic B-Spline Blending Functions

- Blending functions imply properties:
  - Local control
  - Approximating
  - C^2 continuity
  - Convex hull





Cubic Splines

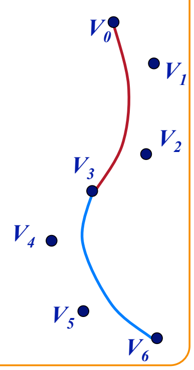
- Splines covered in this lecture
  - Cubic B-Spline
  - ➢ Cubic Bezier
- There are many others

Properties determine blending functions

Blending functions determine properties

Cubic Bezier

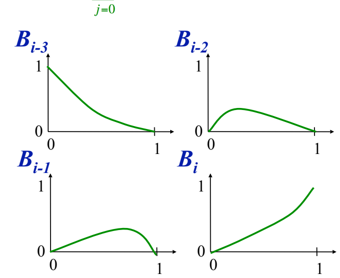
- Developed around 1960 by both
  - Bezier (Renault)
  - deCasteljau (Citroen)
- Properties:
  - Local control
  - C^1 continuity
  - Interpolating (every third)

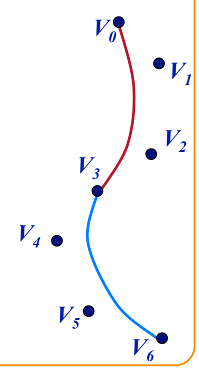


Cubic Bezier curves

Blending functions:

$$B_i(u) = \sum_{j=0}^{m} a_j u^j$$





Cubic Bezier Curves

Bézier curves in matrix form:

$$Q(u) = \sum_{i=0}^{n} V_i \binom{n}{i} u^i (1-u)^{n-i}$$

$$= (1-u)^3 V_0 + 3u(1-u)^2 V_1 + 3u^2(1-u) V_2 + u^3 V_3$$

$$= \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \end{pmatrix}$$

$\mathbf{M}_{Bezier}$

Basic properties of Bézier curves

- Endpoint interpolation:

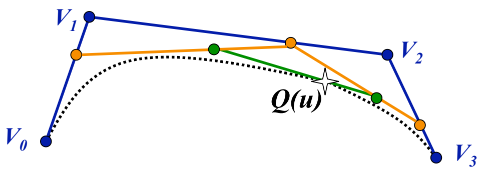
$$Q(0) = V_0$$
$$Q(1) = V_n$$

- Convex hull:
  - Curve is contained within convex hull of control polygon
- Symmetry

$$Q(u) \text{ defined by } \{V_0, \ldots, V_n\} \equiv Q(1-u) \text{ defined by } \{V_n, \ldots, V_0\}$$

Bézier curves

- Curve ***Q(u)*** can also be defined by nested interpolation:



V_i's are control points

$\{V_0, V_1, \ldots, V_n\}$ is control polygon

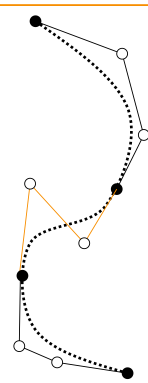
Bezier Curve Display

Pseudocode for displaying Bézier curves:

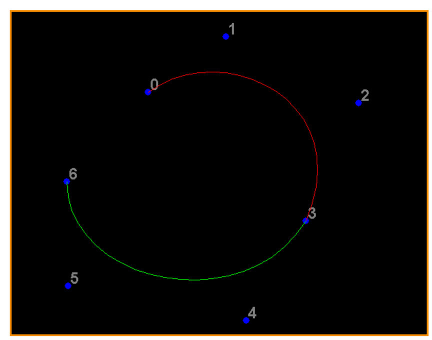
procedure Display($\{V_i\}$):
  if $\{V_i\}$ flat within ε
  then
    output line segment V_0V_n
  else
    subdivide to produce $\{L_i\}$ and $\{R_i\}$
    Display($\{L_i\}$)
    Display($\{R_i\}$)
  end if
end procedure

Bezier Splines

- For more complex curves, piece together Bézier curves
- Solve for "interior" control vertices
  - Positional (C^0) continuity
  - Derivative (C^1) continuity



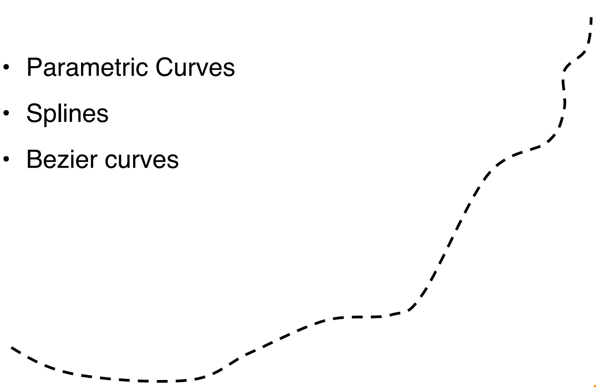
Bezier Splines



Patrick Min

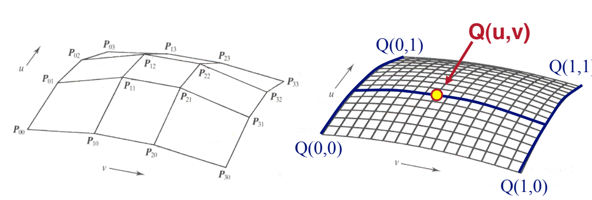
Summary

- Parametric Curves
- Splines
- Bezier curves



What's next?

- Use curves to create parameterized surfaces



Watt Figure 6.21