Boundary Representations & Volume Graphics
Videos 3: Surfaces, Solid Modeling

William H. Hsu
Department of Computing and Information Sciences, KSU

KSOL course pages: http://bit.ly/hGvXlH / http://bit.ly/eVizrE
Public mirror web site: http://www.kddresearch.org/Courses/CIS636
Instructor home page: http://www.cis.ksu.edu/~bhsu

Readings:
Today: §10.4, 12.7, Eberly *2e* – see http://bit.ly/ieUq45, **Mesh handout**
Next class: **Flash animation handout**
Reference on curves (required for CIS 736): §11.1 – 11.6, Eberly *2e*
Videos: http://www.kddresearch.org/Courses/CIS636/Lectures/Videos/

Lecture Outline

- Reading for Last Class: §5.3 – 5.5, Eberly *2e*, CGA handout
- Reading for Today: §10.4, 12.7, Eberly *2e*, Mesh handout
- Reading for Next Class: §11.1 – 11.6 (736), Flash animation handout
- Last Time: Skinning and Morphing
  - Skins: surface meshes for faces, character models
  - Morphing: gradual transition between skins
  - GPU-based vertex tweening: texture arrays, vertex texturing, hybrid
- Today: Curves & Surfaces
  - Piecewise linear, quadratic, cubic curves and their properties
  - Interpolation: subdivision (DeCasteljau's algorithm)
  - Bicubic surfaces & bilinear interpolation
- Outside Viewing: CG Basics 10, Advanced CG 4 & 5
- Previous Videos: Morphing & Other Special Effects (SFX)
- Today's Videos: Bicubic Surfaces (NURBS), Solid Modeling

Where We Are

Lecture	Topic	Primary Source(s)
0	Course Overview	Chapter 1, Eberly 2e
1	**CG Basics: Transformation Matrices; Lab 0**	**Sections (§) 2.1, 2.2**
2	Viewing 1: Overview, Projections	§ 2.2.3 – 2.2.4, 2.8
3	Viewing 2: Viewing Transformation	§ 2.3 esp. 2.3.4; FVFH slides
4	**Lab 1a: Flash & OpenGL Basics**	**Ch. 2, 16[1], Angel *Primer***
5	Viewing 3: Graphics Pipeline	§ 2.3 esp. 2.3.7; 2.6, 2.7
6	Scan Conversion 1: Lines, Midpoint Algorithm	§ 2.5.1, 3.1; FVFH slides
7	**Viewing 4: Clipping & Culling; Lab 1b**	**§ 2.3.5, 2.4, 3.1.3**
8	Scan Conversion 2: Polygons, Clipping Intro	§ 2.4, 2.5 esp. 2.5.4, 3.1.6
9	Surface Detail 1: Illumination & Shading	§ 2.5, 2.6.1 – 2.6.2, 4.3.2, 20.2
10	**Lab 2a: Direct3D / DirectX Intro**	**§ 2.7, Direct3D handout**
11	Surface Detail 2: Textures; OpenGL Shading	§ 2.6.3, 20.3 – 20.4, *Primer*
12	Surface Detail 3: Mappings; OpenGL Textures	§ 20.5 – 20.13
13	**Surface Detail 4: Pixel/Vertex Shad.; Lab 2b**	**§ 3.1**
14	Surface Detail 5: Direct3D Shading; OGLSL	§ 3.2 – 3.4, Direct3D handout
15	Demos 1: CGA, Fun; Scene Graphs: State	§ 4.1 – 4.3, **CGA handout**
16	**Lab 3a: Shading & Transparency**	**§ 2.6, 20.1, *Primer***
17	**Animation 1: Basics, Keyframes; HW/Exam**	**§ 5.1 – 5.2**
	Exam 1 review; Hour Exam 1 (evening)	**Chapters 1 – 4, 20**
18	**Scene Graphs: Rendering; Lab 3b: Shader**	**§ 4.4 – 4.7**
19	**Demos 2: SFX: Skinning, Morphing**	**§ 5.3 – 5.5, CGA handout**
20	Demos 3: Surfaces; B-reps/Volume Graphics	§ 10.4, 12.7, **Mesh handout**

Lightly-shaded entries denote the due date of a written problem set; heavily-shaded entries, that of a machine problem (programming assignment); blue-shaded entries, that of a paper review; and the green-shaded entry, that of the term project.

Green, **blue** and **red** letters denote **exam review**, **exam**, and **exam solution review** dates.

Where We're Going

21	**Lab 4a: Animation Basics**	**Flash animation handout**
22	Animation 2: Rotations; Dynamics, Kinematics	Chapter 17, esp. §17.1 – 17.2
23	Demos 4: Modeling & Simulation; Rotations	Chapter 10[1], 13[2], §17.3 – 17.5
24	**Collisions 1: axes, OBBs, Lab 4b**	**§2.4.3, 8.1, GL handout**
25	Spatial Sorting: Binary Space Partitioning	Chapter 6, esp. §6.1
26	**Demos 5: More CGA; Picking; HW/Exam**	**Chapter 7[2]; § 8.4**
27	**Lab 5a: Interaction Handling**	**§ 8.3 – 8.4; 4.2, 5.0, 5.6, 9.1**
28	Collisions 2: Dynamic, Particle Systems	§ 9.1, particle system handout
	Exam 2 review; Hour Exam 2 (evening)	**Chapters 5 – 6, 7[2] – 8, 12, 17**
29	**Lab 5b: Particle Systems**	**Particle system handout**
30	**Animation 3: Control & IK**	**§ 5.3, CGA handout**
31	Ray Tracing 1: intersections, ray trees	Chapter 14
32	Lab 6a: Ray Tracing Basics with POV-Ray	RT handout
33	Ray Tracing 2: advanced topic survey	Chapter 15, **RT handout**
34	Visualization 1: Data (Quantities & Evidence)	Tufte handout (1)
35	**Lab 6b: More Ray Tracing**	**RT handout**
36	Visualization 2: Objects	Tufte handout (2 & 4)
37	Color Basics; Term Project Prep	**Color handout**
38	Lab 7: Fractals & Terrain Generation	**Fractals/Terrain handout**
39	**Visualization 3: Processes; Final Review 1**	Tufte handout (3)
40	**Project presentations 1; Final Review 2**	**–**
41	Project presentations 2	–
	Final Exam	**Ch. 1 – 8, 10 – 15, 17, 20**

Lightly-shaded entries denote the due date of a written problem set; heavily-shaded entries, that of a machine problem (programming assignment); blue-shaded entries, that of a paper review; and the green-shaded entry, that of the term project.

Lab exercises are always due on the day before the next lab.

Green, **blue** and **red** letters denote **exam review**, **exam**, and **exam solution review** dates.

Review [1]: Morphing Targets

- **Vertex Tweening**
  - Two key meshes are blended
  - Varying by time
- **Morph Targets**
  - Represent by relative vectors
    - From base mesh
    - To target meshes
  - Geometry: mesh represents model
  - Samples: corresponding images
- **Applications**
  - Image morphing (see videos)
  - Lip syncing (work of Elon Gasper)

http://youtu.be/Vi5PlrZpG40

Adapted from "Morphing and Animation" © 2007 G. J. Katz, University of Pennsylvania
Lecture 12, CIS 565 (formerly 665): *GPU Programming and Architecture*, http://bit.ly/eFkXmk

Review [2]: Morph Target Animation & Lip Sync

- From Base Mesh to Multiple Targets
- Effects: Facial Animation with Muscle Deformation

- Lip Sync
  - Problem: matching mouth movements to speech waveform
  - Early work: Elon Gasper & Bright Star – http://bit.ly/g4sKBL
  - Used in Sierra's *Alphabet Blocks* (1992) – http://bit.ly/hSKCE3

Adapted from "Morphing and Animation" © 2007 G. J. Katz, University of Pennsylvania
Lecture 12, CIS 565 (formerly 665): *GPU Programming and Architecture*, http://bit.ly/eFkXmk

Review [3]: GPU Animation Method 1

- Hold Vertex Data in Texture Arrays
- Manipulate Data in Pixel Shader / Fragment Shader
- Re-output to Texture Arrays
- Pass Output as Input to Vertex Shader (NB: Usually Other Way Around!)

Adapted from "Morphing and Animation" © 2007 G. J. Katz, University of Pennsylvania
Lecture 12, CIS 565 (formerly 665): *GPU Programming and Architecture*, http://bit.ly/eFkXmk

Review [4]: Pros & Cons of GPU Method 1

- **Advantages**
  - **Keeps vertex, geometry processing units' workload at minimum (Why is this good?)**
  - **Good for copy operations, vertex tweening**
- **Disadvantages**
  - **Per-vertex data has to be accessed through texture lookups**
  - **Number of constant registers is less in pixel shader (224) than vertex shader (256)**
  - **Can not divide modification process into several pieces because only single quad is drawn**
  - **Therefore: constant registers must hold all bone matrices and morph target weights for entire object**

Adapted from "Morphing and Animation" © 2007 G. J. Katz, University of Pennsylvania
Lecture 12, CIS 565 (formerly 665): *GPU Programming and Architecture*, http://bit.ly/eFkXmk

Review [5]: GPU Animation Method 2

- **Apply Modifications in Vertex Shader, Do Nothing in Pixel Shader**
  - **Destination pixel is specified explicitly as vertex shader input**
  - **Still writing all vertices to texture**
- **Advantage: Can Easily Segment Modification Groups**
- **Disadvantage: Speed Issues Make This Method Impractical**

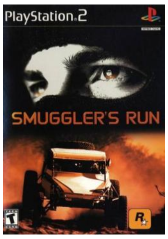
Adapted from "Morphing and Animation" © 2007 G. J. Katz, University of Pennsylvania
Lecture 12, CIS 565 (formerly 665): *GPU Programming and Architecture*, http://bit.ly/eFkXmk

Review [6]: Hybrid CPU/GPU System

- **Use Hybrid CPU/GPU Approach to Get Real Speed Advantage**
  1. **Let CPU compute final vertex attributes used during rendering frames *n*, *n* + *k***
  2. **Let GPU compute vertex tweening at frames greater than *n*, smaller than *n* + *k***
  3. **Phase shift animations between characters so processors do not have peak loads**
- **Advantages**
  - **Vertex tweening supported on almost all hardware**
  - **Modification algorithms performed on CPU, so no restrictions**

Adapted from "Morphing and Animation" © 2007 G. J. Katz, University of Pennsylvania
Lecture 12, CIS 565 (formerly 665): *GPU Programming and Architecture*, http://bit.ly/eFkXmk

Acknowledgements: Curves & Surfaces



Steve Rotenberg

Visiting Lecturer
Graphics Lab
University of California – San Diego
CEO/Chief Scientist, PixelActive
http://graphics.ucsd.edu

Barry McCaul

Lecturer
School of Computing
Dublin City University
http://www.computing.dcu.ie/~bmccaul/

Ken Hawick

Professor
Institute of Information and Mathematical Sciences (IIMS)
Massey University – Albany
http://www.massey.ac.nz/~kahawick/

MASSEY UNIVERSITY
TE KUNENGA KI PŪREHUROA

Acknowledgements: Splines

Jim Foley

Professor, College of Computing & Stephen Fleming Chair in Telecommunications

Georgia Institute of Technology

James D. Foley
Georgia Tech
http://bit.ly/ajYf2Q

Andy van Dam

T. J. Watson University Professor of Technology and Education & Professor of Computer Science

Brown University

Andries van Dam
Brown University
http://www.cs.brown.edu/~avd/

Steve Feiner

Professor of Computer Science & Director, Computer Graphics and User Interfaces Laboratory

Columbia University

Steven K. Feiner
Columbia University
http://www.cs.columbia.edu/~feiner/

John F. Hughes

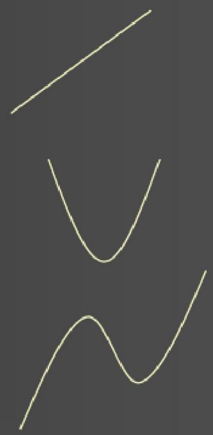
Associate Professor of Computer Science

Brown University

John F. Hughes
Brown University
http://www.cs.brown.edu/~jfh/

Polynomial Functions

- Linear: $f(t) = at + b$
- Quadratic: $f(t) = at^2 + bt + c$
- Cubic: $f(t) = at^3 + bt^2 + ct + d$



Adapted from slides ♥ 2003 – 2006 S. Rotenberg, UCSD
CSE167: Computer Graphics, Fall 2006, http://bit.ly/hXxAIP

Vector Polynomials (Curves)

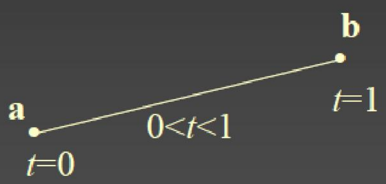
- Linear: $\mathbf{f}(t) = \mathbf{a}t + \mathbf{b}$
- Quadratic: $\mathbf{f}(t) = \mathbf{a}t^2 + \mathbf{b}t + \mathbf{c}$
- Cubic: $\mathbf{f}(t) = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d}$

We usually define the curve for $0 \leq t \leq 1$

Adapted from slides ♥ 2003 – 2006 S. Rotenberg, UCSD
CSE167: Computer Graphics, Fall 2006, http://bit.ly/hXxAIP

Linear Interpolation

- Linear interpolation (Lerp) is a common technique for generating a new value that is somewhere in between two other values
- A ‘value’ could be a number, vector, color, or even something more complex like an entire 3D object…
- Consider interpolating between two points **a** and **b** by some parameter *t*



$$Lerp(t, \mathbf{a}, \mathbf{b}) = (1 - t)\mathbf{a} + t\mathbf{b}$$

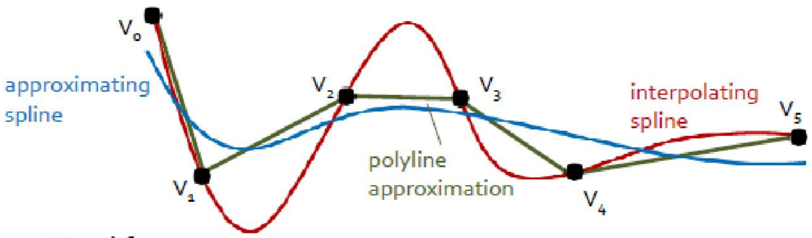
Adapted from slides ♥ 2003 – 2006 S. Rotenberg, UCSD
CSE167: Computer Graphics, Fall 2006, http://bit.ly/hXxAIP

Splines [1]: Representing General Curves

- We can represent any polyline with vertices and edges. What about curves?
  - Don't want to store curves as raster graphics (aliasing, not scalable, memory intensive). We need a more efficient mathematical representation
  - Store control points in a list, find some way of smoothly interpolating between them
- Piecewise Linear Approximation
  - Not smooth, looks awful without many control points
- Trigonometric functions
  - Difficult to manipulate and control, computationally expensive to compute
- Higher order polynomials
  - Relatively cheap to compute, only slightly more difficult to operate on than polylines

Splines [2]: Spline Types & Uses

- Polynomial interpolation is typically used. Splines are second or third order parametric curves governed by control points or control vectors
- Used early on in automobile and aircraft industry to achieve smoothness – even small differences can make a big difference in efficiency and look



Splines still exist outside of computers. They're now called flexible curves.

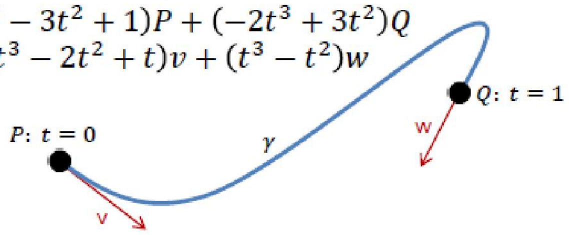
- Used for:
  - Representing smooth shapes in 2D as outlines or in 3D using "patches" parameterized with two variables: *s* and *t* (see slide 12)
  - Animation paths for "tweening" between keyframes
  - Approximating "expensive" functions (polynomials are cheaper than log, sin, cos ...)

Splines [3]: Hermite Curves

- Polylines are linear (1st order polynomial) interpolations between points
  - Given points P and Q, line between the two is given by the parametric equation:
    $$x(t) = (1 - \mathrm{t})\mathrm{P} + \mathrm{tQ}, \qquad 0 \le \mathrm{t} \le 1$$
  - $(1 - \mathrm{t})$ and t are called weighting functions of P and Q
- Splines are higher order polynomial interpolations between points
  - Like linear interpolation but with higher order weighting functions allowing better approximations/smoother curves
- One representation - Hermite curves (interpolating spline):
  - Determined by two control points P and Q, an initial tangent vector v and a final tangent vector w.

$$\gamma(t) = (2t^3 - 3t^2 + 1)P + (-2t^3 + 3t^2)Q$$
$$+ (t^3 - 2t^2 + t)v + (t^3 - t^2)w$$

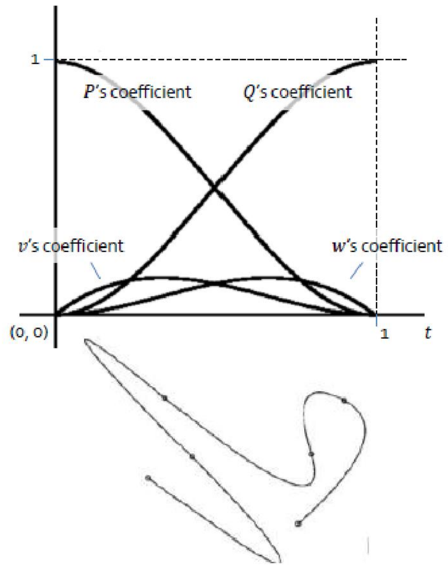
  - Satisfies:
    - $\gamma(0) = P$
    - $\gamma(1) = Q$
    - $\gamma'(0) = v$
    - $\gamma'(1) = w$



Splines [4]: Hermite Weighting Explained

- Polynomial splines have more complex weighting functions than lines
  - Coefficients for P and Q are now 3rd degree polynomials
- At $t = 0$:
  - Coefficient of P is 1, all others 0
  - Derivative of coefficient of v is 1, derivative of all others is 0
- At $t = 1$:
  - Coefficient of Q is 1, all others 0
  - Derivative of coefficient of w is 1, derivative of all others is 0
- Can be chained together to make more complex curves

Polynomial weighting functions in Hermite curve equation



Splines [5]: Bézier Curves

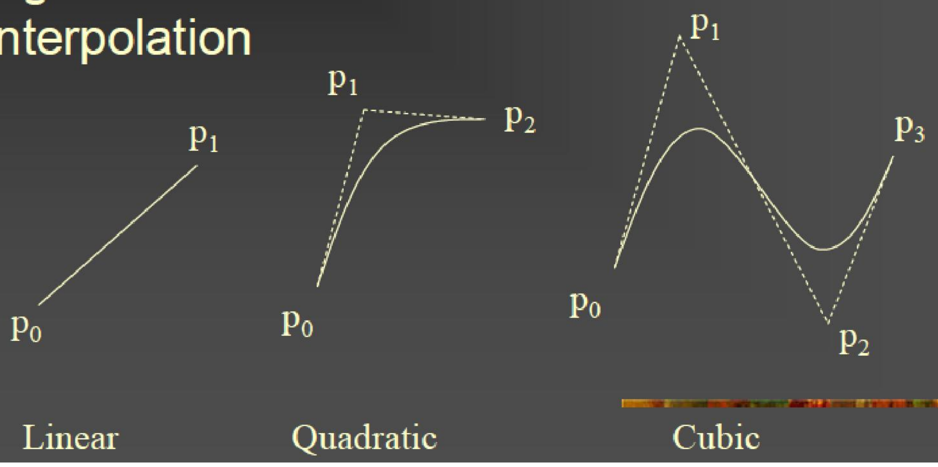
- Bezier representation is similar to Hermite
  - 4 points instead of 2 points and 2 vectors $(P_1 \dots P_4)$
  - Initial position P_1, tangent vector is $P_2 - P_1$
  - Final position P_4 tangent vector is $P_4 - P_3$
  - This representation allows a spline to be stored as a list of vertices with some global parameters that describe the smoothness and continuity

- Bezier splines are widely used (Adobe, Microsoft) for font definition

Brown Exploratory (Spalter & Bielawa): http://bit.ly/fva1il

Bézier Curves [1]: Piecewise Cubic Curves

- Bezier curves can be thought of as a higher order extension of linear interpolation



Adapted from slides ♥ 2003 – 2006 S. Rotenberg, UCSD
CSE167: Computer Graphics, Fall 2006, http://bit.ly/hXxAIP

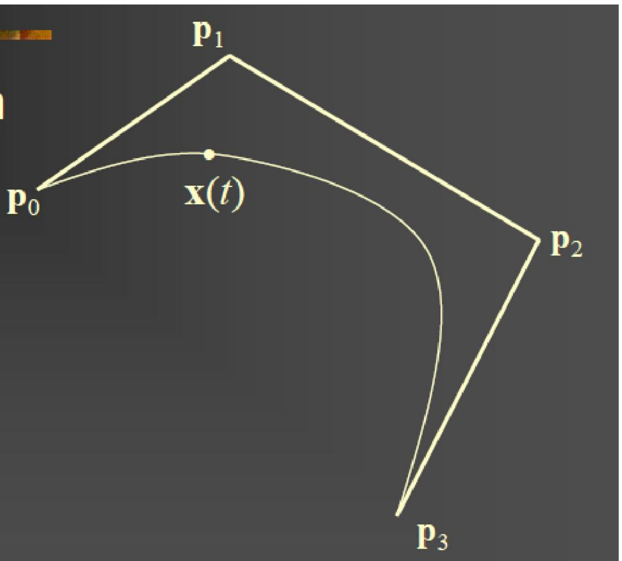
Bézier Curves [2]: Formulation

- There are lots of ways to formulate Bezier curves mathematically. Some of these include:
  - de Castlejau (recursive linear interpolations)
  - Bernstein polynomials (functions that define the influence of each control point as a function of t)
  - Cubic equations (general cubic equation of t)
  - Matrix form
- We will briefly examine each of these

Adapted from slides ♥ 2003 – 2006 S. Rotenberg, UCSD
CSE167: Computer Graphics, Fall 2006, http://bit.ly/hXxAIP

Bézier Curves [3]: Interpolation Problem Defined

- Find the point **x** on the curve as a function of parameter *t*:

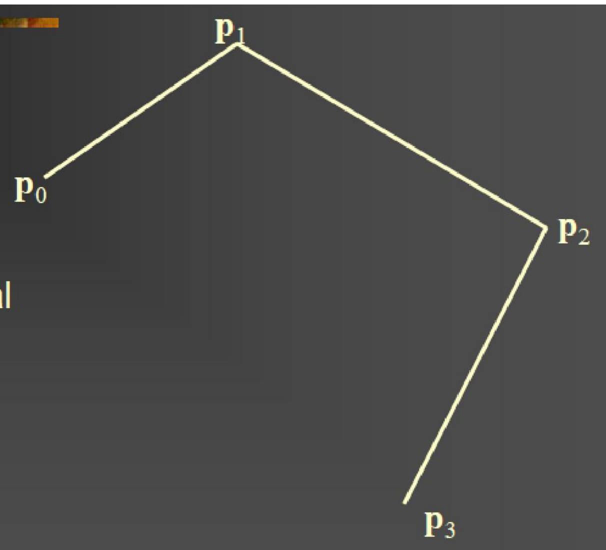


Adapted from slides ♥ 2003 – 2006 S. Rotenberg, UCSD
CSE167: Computer Graphics, Fall 2006, http://bit.ly/hXxAIP

De Casteljau's Algorithm [1]: Idea

- The de Casteljau algorithm describes the curve as a recursive series of linear interpolations
- This form is useful for providing an intuitive understanding of the geometry involved, but it is not the most efficient form

De Casteljau's Algorithm [2]: Initialization

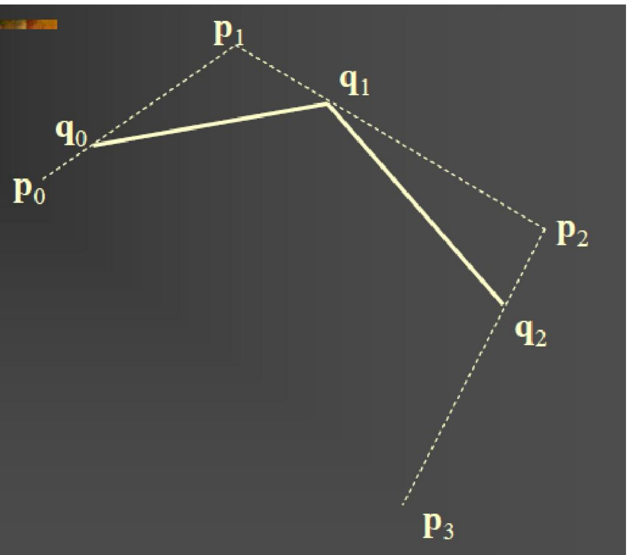


- We start with our original set of points
- In the case of a cubic Bezier curve, we start with four points

Adapted from slides ♥ 2003 – 2006 S. Rotenberg, UCSD
CSE167: Computer Graphics, Fall 2006, http://bit.ly/hXxAIP

De Casteljau's Algorithm [3]: Lerp Step 1

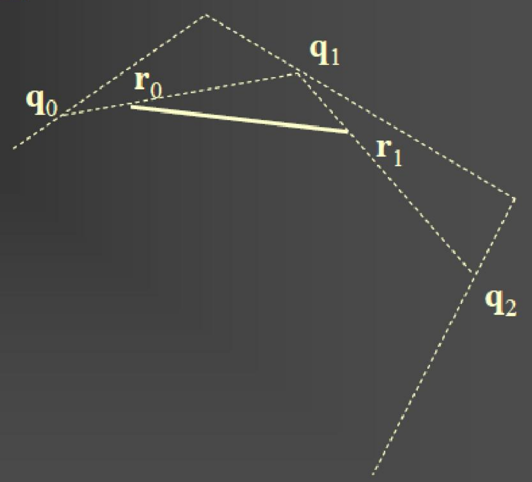
$$\mathbf{q}_0 = Lerp(t, \mathbf{p}_0, \mathbf{p}_1)$$
$$\mathbf{q}_1 = Lerp(t, \mathbf{p}_1, \mathbf{p}_2)$$
$$\mathbf{q}_2 = Lerp(t, \mathbf{p}_2, \mathbf{p}_3)$$



Adapted from slides ♥ 2003 – 2006 S. Rotenberg, UCSD
CSE167: Computer Graphics, Fall 2006, http://bit.ly/hXxAIP

De Casteljau's Algorithm [4]: Lerp Step 2

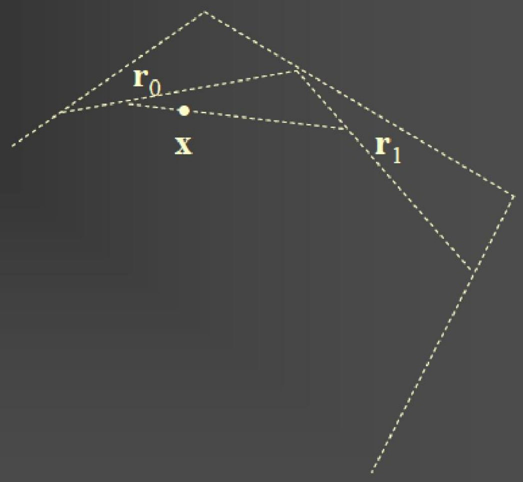
$$\mathbf{r}_0 = Lerp(t, \mathbf{q}_0, \mathbf{q}_1)$$
$$\mathbf{r}_1 = Lerp(t, \mathbf{q}_1, \mathbf{q}_2)$$



Adapted from slides ♥ 2003 – 2006 S. Rotenberg, UCSD
CSE167: Computer Graphics, Fall 2006, http://bit.ly/hXxAIP

De Casteljau's Algorithm [5]: Lerp Step 3

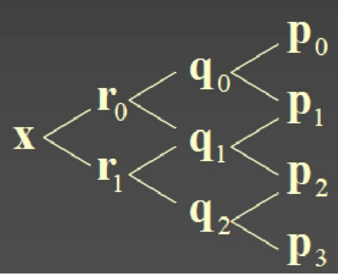
$$\mathbf{x} = Lerp(t, \mathbf{r}_0, \mathbf{r}_1)$$



Adapted from slides ♥ 2003 – 2006 S. Rotenberg, UCSD
CSE167: Computer Graphics, Fall 2006, http://bit.ly/hXxAIP

De Casteljau's Algorithm [6]: Recursive Linear Interpolation

$$\mathbf{x} = Lerp(t, \mathbf{r}_0, \mathbf{r}_1) \quad \begin{matrix} \mathbf{r}_0 = Lerp(t, \mathbf{q}_0, \mathbf{q}_1) \\ \mathbf{r}_1 = Lerp(t, \mathbf{q}_1, \mathbf{q}_2) \end{matrix} \quad \begin{matrix} \mathbf{q}_0 = Lerp(t, \mathbf{p}_0, \mathbf{p}_1) \\ \mathbf{q}_1 = Lerp(t, \mathbf{p}_1, \mathbf{p}_2) \\ \mathbf{q}_2 = Lerp(t, \mathbf{p}_2, \mathbf{p}_3) \end{matrix} \quad \begin{matrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{matrix}$$



Adapted from slides ♥ 2003 – 2006 S. Rotenberg, UCSD
CSE167: Computer Graphics, Fall 2006, http://bit.ly/hXxAIP

Bernstein Polynomials [1]: Coefficients of Control Points

$$\mathbf{x} = (1-t)\big((1-t)((1-t)\mathbf{p}_0 + t\mathbf{p}_1) + t((1-t)\mathbf{p}_1 + t\mathbf{p}_2)\big) + t\big((1-t)((1-t)\mathbf{p}_1 + t\mathbf{p}_2) + t((1-t)\mathbf{p}_2 + t\mathbf{p}_3)\big)$$

$$\mathbf{x} = (1-t)^3\mathbf{p}_0 + 3(1-t)^2 t\mathbf{p}_1 + 3(1-t)t^2\mathbf{p}_2 + t^3\mathbf{p}_3$$

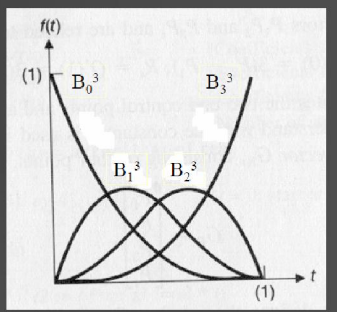
$$\mathbf{x} = \left(-t^3 + 3t^2 - 3t + 1\right)\mathbf{p}_0 + \left(3t^3 - 6t^2 + 3t\right)\mathbf{p}_1 + \left(-3t^3 + 3t^2\right)\mathbf{p}_2 + \left(t^3\right)\mathbf{p}_3$$

Adapted from slides ♥ 2003 – 2006 S. Rotenberg, UCSD
CSE167: Computer Graphics, Fall 2006, http://bit.ly/hXxAIP

Bernstein Polynomials [2]: Piecewise Cubic Basis

$$\mathbf{x} = \left(-t^3 + 3t^2 - 3t + 1\right)\mathbf{p}_0 + \left(3t^3 - 6t^2 + 3t\right)\mathbf{p}_1 + \left(-3t^3 + 3t^2\right)\mathbf{p}_2 + \left(t^3\right)\mathbf{p}_3$$

$$\mathbf{x} = B_0^3(t)\mathbf{p}_0 + B_1^3(t)\mathbf{p}_1 + B_2^3(t)\mathbf{p}_2 + B_3^3(t)\mathbf{p}_3$$



$$B_0^3(t) = -t^3 + 3t^2 - 3t + 1$$

$$B_1^3(t) = 3t^3 - 6t^2 + 3t$$

$$B_2^3(t) = -3t^3 + 3t^2$$

$$B_3^3(t) = t^3$$

Adapted from slides ♥ 2003 – 2006 S. Rotenberg, UCSD
CSE167: Computer Graphics, Fall 2006, http://bit.ly/hXxAIP

Bernstein Polynomials [3]: Binomial Form of Basis Functions

$$B_0^3(t) = -t^3 + 3t^2 - 3t + 1$$
$$B_1^3(t) = 3t^3 - 6t^2 + 3t$$
$$B_2^3(t) = -3t^3 + 3t^2$$
$$B_3^3(t) = t^3$$

$$B_0^2(t) = t^2 - 2t + 1$$
$$B_1^2(t) = -2t^2 + 2t$$
$$B_2^2(t) = t^2$$

$$B_0^1(t) = -t + 1$$
$$B_1^1(t) = t$$

$$B_i^n(t) = \binom{n}{i}(1-t)^{n-i}(t)^i \qquad \binom{n}{i} = \frac{n!}{i!(n-i)!}$$

$$\sum B_i^n(t) = 1$$

Adapted from slides ♥ 2003 – 2006 S. Rotenberg, UCSD
CSE167: Computer Graphics, Fall 2006, http://bit.ly/hXxAIP

Bernstein Polynomials [4]: Cubic Matrix Form

$$\mathbf{x} = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d}$$

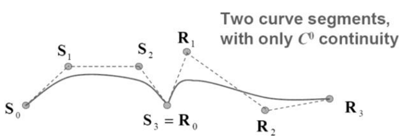
$$\mathbf{a} = \left(-\mathbf{p}_0 + 3\mathbf{p}_1 - 3\mathbf{p}_2 + \mathbf{p}_3\right)$$
$$\mathbf{b} = \left(3\mathbf{p}_0 - 6\mathbf{p}_1 + 3\mathbf{p}_2\right)$$
$$\mathbf{c} = \left(-3\mathbf{p}_0 + 3\mathbf{p}_1\right)$$
$$\mathbf{d} = \left(\mathbf{p}_0\right)$$

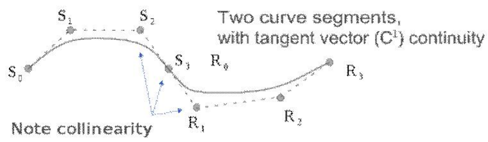
$$\mathbf{x} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} \qquad \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix}$$

Adapted from slides ♥ 2003 – 2006 S. Rotenberg, UCSD
CSE167: Computer Graphics, Fall 2006, http://bit.ly/hXxAIP

Geometric (G^i) vs. Mathematical (C^i) Continuity

- <u>Geometric Continuity</u>: G^i
  - Guarantees that direction of i^{th} derivative equal
  - G^0: curves touch at join point
  - G^1: curves also share common tangent direction at join point
  - G^2: curves also share common center of curvature at join point
- <u>Mathematical Continuity</u>: C^i
  - Guarantees that direction, magnitude of i^{th} derivative equal
  - $C^0 \equiv G^0$: curves touch at join point
  - C^1: curves share common tangent direction / magnitude at join point
  - C^2: curves share common second derivative at join point

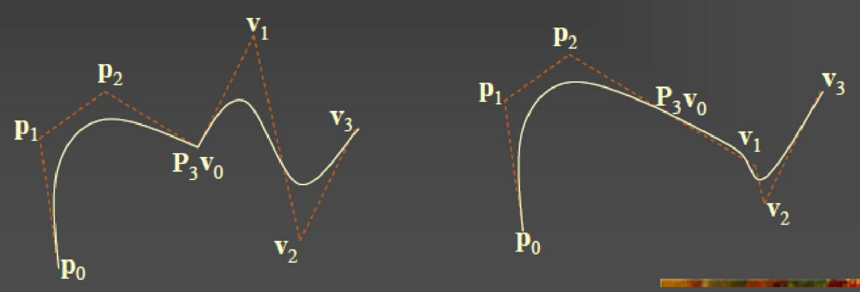




http://bit.ly/hQwnY2

Connecting Bézier Curves: C^i Continuity

- A simple way to make larger curves is to connect up Bezier curves
- Consider two Bezier curves defined by $\mathbf{p}_0 \ldots \mathbf{p}_3$ and $\mathbf{v}_0 \ldots \mathbf{v}_3$
- If $\mathbf{p}_3 = \mathbf{v}_0$, then they will have C^0 continuity
- If $(\mathbf{p}_3 - \mathbf{p}_2) = (\mathbf{v}_1 - \mathbf{v}_0)$, then they will have C^1 continuity
- C^2 continuity is more difficult…



C^0 continuity $\quad$ C^1 continuity

Adapted from slides ♥ 2003 – 2006 S. Rotenberg, UCSD
CSE167: Computer Graphics, Fall 2006, http://bit.ly/hXxAIP

Building 3-D Primitives

- Made out of 2D and 1D primitives

- Triangles are commonly used
- Many triangles used for a single object is a triangular mesh.

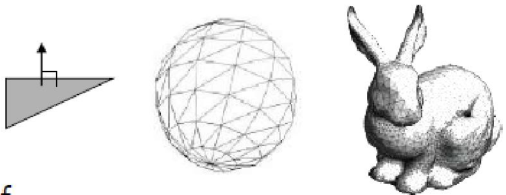
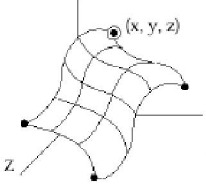
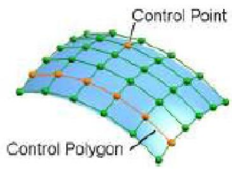


Image credit (Stanford Bunny): http://bit.ly/fDSxn9

- Splines used to describe boundaries of "patches" – these can be "sewn together" to represent curved surfaces
  - $x(s,t) = (1 - s)^3 * (1 - t)^3 * P_{1,1} + (1 - s)^3 * 3t(1 - t)^2 * P_{1,2} + \ ...$



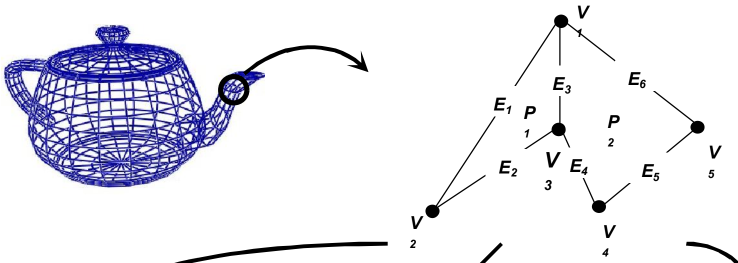


Surface Modeling: Utah Teapot

- Many real-world objects: inherently smooth
  - Therefore need infinitely many points to model them
  - Not feasible for a computer with finite storage
- More often we merely approximate objects with
  - Pieces of planes
  - Spheres
  - Other shapes that are easy to describe mathematically
- Two most common representations for 3-D surfaces
  - Polygon mesh surfaces
  - Parametric surfaces
- Will also discuss parametric curves
  - 2-D, embedded in 3-D
  - Think of parametric surfaces as generalization of curves

Adapted from slides ♥ 2006 B. McCaul, Dublin City University
CA433 Computer Graphics I, http://bit.ly/ghw08y

Polygon Meshes [1]: Vertex, Edge, Polygon Tables



VERTEX TABLE
V_1: x_1, y_1, z_1
V_2: x_2, y_2, z_2
V_3: x_3, y_3, z_3
V_4: x_4, y_4, z_4
V_5: x_5, y_5, z_5

EDGE TABLE
E_1: V_1, V_2
E_2: V_2, V_3
E_3: V_3, V_1
E_4: V_3, V_4
E_5: V_4, V_5
E_6: V_5, V_1

POLYGON TABLE
P_1: V_1, V_2, V_3
P_2: V_1, V_3, V_4, V_5

OR

POLYGON TABLE
P_1: E_1, E_2, E_3
P_2: E_3, E_4, E_5, E_6

Adapted from slides ♥ 2006 B. McCaul, Dublin City University
CA433 Computer Graphics I, http://bit.ly/ghw08y

Polygon Meshes [2]: "Eliminating" Edge Table

- The geometry can be stored as three tables: a vertex table, an edge table, and a polygon table. Each entry in the vertex table is a list of coordinates defining that point. Each entry in the edge table consists of a pointer to each endpoint of that edge. And the entries in the polygon table define a polygon by providing pointers to the edges that make up the polygon.

- We can eliminate the edge table by letting the polygon table reference the vertices directly, but we can run into problems, such as drawing some edges twice, because we don't realise that we have visited the same set of points before, in a different polygon. We could go even further and eliminate the vertex table by listing all the coordinates explicitly in the polygon table, but this wastes space because the same points appear in the polygon table several times.

Adapted from slides ♥ 2006 B. McCaul, Dublin City University
CA433 Computer Graphics I, http://bit.ly/ghw08y

Polygon Meshes [3]: Representation

1. **Explicit way: just list 3D vertices of each polygon in a certain order. Problems are, firstly it represents same vertex many times and secondly, no explicit representation of shared edges and vertices**

$$P = ((x_1, y_1, z_1), (x_2, y_2, z_2), \ldots, (x_n, y_n, z_n))$$

2. **Pointer to a vertex list: store all vertices once into a numbered list, and represent each polygon by its vertices. It saves space (vertex only listed once) but still has no explicit representation of shared edges and vertices**

$$P = (1, 3, 4, 5)$$

3. **Explicit edges: list all edges that belong to a polygon, and for each edge list the vertices that define it along with the polygons of which it is a member.**

$$E = (V_1, V_2, P_1)$$

Adapted from slides ♥ 2006 B. McCaul, Dublin City University
CA433 Computer Graphics I, http://bit.ly/ghw08y

Types of Curves [1]: Explicit & Implicit

1. **Explicit**

   In Cartesian plane, explicit equation of planar curve given by

   $$y = f(x)$$

   Difficulties with this approach

   a) **impossible to get multiple values of y for single x, so curves such as circles and ellipses must be represented by multiple curve segments**

   b) **describing curves with vertical tangents: difficult, numerically unstable**

2. **Implicit**

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

   $$Ax + By + C = 0$$

   Difficulties: determining tangent continuity of two given curves – crucial in many applications

   (Circle can be defined as: $x^2 + y^2 = 1$, but what about half circle?)

Adapted from slides ♥ 2006 B. McCaul, Dublin City University
CA433 Computer Graphics I, http://bit.ly/ghw08y

Types of Curves [2]: Parametric

3. **Parametric Curves**

Cubic polynomials that define curve segment *Q*(*t*)= [*x*(*t*) *y*(*t*)]^T are of form:

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$
$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$$

Written in matrix form, system becomes

$$Q(t) = [x(t)\ y(t)] = T \cdot C$$

where

$$C = \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_y & d_x \end{bmatrix} \qquad T = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix}$$

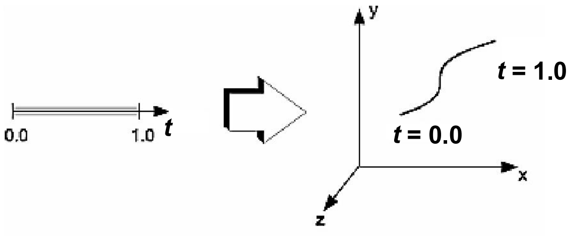
Adapted from slides ♥ 2006 B. McCaul, Dublin City University
CA433 Computer Graphics I, http://bit.ly/ghw08y

Parametric Bicubic Surfaces [1]

- **Equations that describe parametric curve depend on variable *t* not explicitly part of geometry**

$$x = f(t)$$
$$y = g(t)$$

- **By <u>sweeping</u> through *t*, in our case $0 \le t \le 1$, we can evaluate equations and determine *x*, *y* values for points on curve**



Parameter space **Object space**

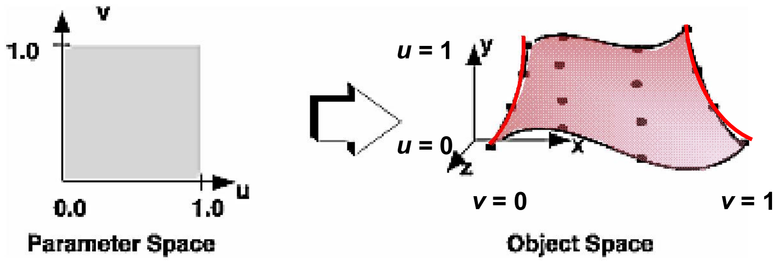
Adapted from slides ♥ 2006 B. McCaul, Dublin City University
CA433 Computer Graphics I, http://bit.ly/ghw08y

Review [8]: Parametric Bicubic Surfaces

- **Parametric Bicubic Surface: Generalization of Parametric Cubic Curve**

$$P(u, v) = [x(u, v), y(u, v), z(u, v)] \quad 0 \le u \le 1 \quad 0 \le v \le 1$$

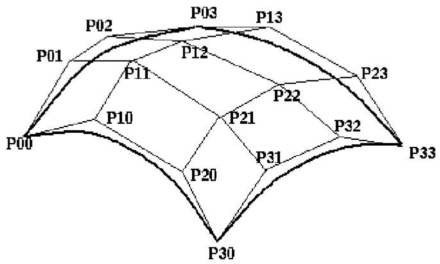
- **From Curves to Surfaces**
  - Let one parameter (say *v*) be held at constant value
  - Above will represent a curve
  - Surface generated by sweeping all points on boundary curve, *e.g.*, *P*(*u, 0*), through cubic trajectories, defined using *v*, to boundary curve *P*(*u,1*)



Adapted from slides ♥ 2006 B. McCaul, Dublin City University
CA433 Computer Graphics I, http://bit.ly/ghw08y

Bézier Surface Patch

The representation of the bicubic surface patch can be illustrated by considering the Bézier Surface Patch. The edge *P*(*0, v*) of a Bezier patch is defined by giving four control points P_{00}, P_{01}, P_{02} and P_{03}. Similarly the opposite edge *P*(*1, v*) can be represented by a Bezier curve with four control points. The surface patch is generated by sweeping the curve *P*(*0, v*) through a cubic trajectory in the parameter *u* to *P*(*1, v*). To define this trajectory we need four control points, hence the Bezier surface patch requires a mesh of 4×4 control points as illustrated below.



Adapted from slides ♥ 2006 B. McCaul, Dublin City University
CA433 Computer Graphics I, http://bit.ly/ghw08y

Surfaces – Simple Extension

- Easy to generalize from cubic curves to bicubic surfaces
- Surfaces defined by parametric equations of two variables, *s* and *t*
- *i.e.*, surface is approximated by series of crossing parametric cubic curves
- Result is polygon mesh
- Decreasing step size in *s* and *t* will give
  - mesh of small near-planar quadrilateral patches
  - more accuracy

$$0 \le s \le 1 \quad and \quad 0 \le t \le 1$$

Adapted from slide ♥ 2007 - 2008 K. Hawick, Massey University
159-235 Graphics and Graphical Programming, http://bit.ly/gmY8R8

MASSEY UNIVERSITY
TE KUNENGA KI PŪREHUROA

Control of Surface Shape

- **Control is now 2-D array of control points**
- **Two parameter surface function, forming tensor product with blending functions, is:**

$$X(s,t) = \sum_{ij} f_i(s) f_j(t) q_{ij}$$

$$\text{similarly} \quad \text{for } Y(s,t) \text{ and } Z(s,t)$$

- **Use appropriate blending functions for Bézier and B-Spline surface functions**
- **Convex Hull property preserved since bicubic is still weighted sum (1)**

Example : Bézier Surface

- **Matrix formulation as follows**

$$x(s,t) = s^T.M_B.q_x.M_B^T.t$$

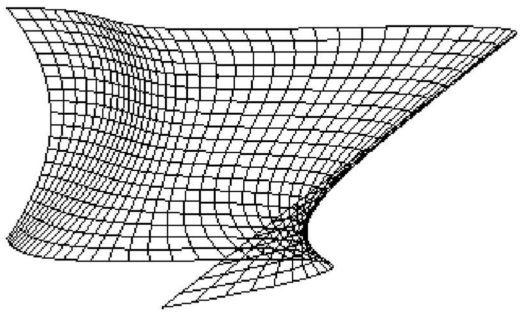
$$q_x \; is \; 4\times4 \; array \; of \; x \; coords$$

$$y(s,t) = s^T.M_B.q_y.M_B^T.t$$

$$q_y \; is \; 4\times4 \; array \; of \; y \; coords$$

$$z(s,t) = s^T.M_B.q_z.M_B^T.t$$

$$q_z \; is \; 4\times4 \; array \; of \; z \; coords$$



- **Substitute suitable values for *s*, *t* (20 in above example)**

B-Spline Surfaces

- **Break surface into 4-sided patches choosing suitable values for *s* and *t***
- **Points on any external edges must be multiple knots of multiplicity *k***
- **Lot more work than Bézier**
- **There are other types of spline systems and NURBS modelling packages are available to make the work much easier**
- **Use polygon packages for display, hidden-surface removal and rendering (Bézier too)**

Continuity of Bicubic Patches

- **Hermite and Bézier patches**
  - C^0 continuity by sharing 4 control points between patches
  - C^1 continuity when both sets of control points either side of the edge are collinear with the edge
- **B-Spline patch**
  - C^2 continuity between patches

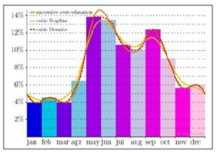
Display (Rendering) of Bicubic Patches

- Can calculate surface normals to bicubic surfaces by vector cross product of their 2 tangent vectors
- Normal is expensive to compute
  - Formulation of normal is a biquintic (two-variable, fifth-degree) polynomial
- Display
  - Can use brute-force method – very expensive!
  - Forward differencing method very attractive

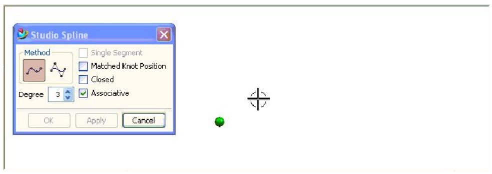
Non-Uniform Rational B-Splines & NURBS Surfaces

- **B-Splines**

© 2009 Wikipedia, *B-spline*

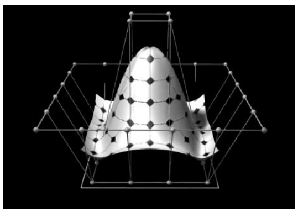


- **NURBS**



© 2007 Wikipedia, *Non-uniform rational B-spline*

- **NURBS Surface**



© 2010 Wikipedia, *Non-uniform rational B-spline*

Curves & Surfaces: Summary

- Curves
  - Bézier: easier to scan convert (DeCasteljau)
  - Hermite: easier to control via GUI (tangent)
- Bicubic patches
  - Bilinear interpolation
  - Control patch *aka* Coons patch

- Acknowledgments - thanks to Eric McKenzie, Edinburgh, from whose Graphics Course some of these slides were adapted.

Sinbad: Legend of the Seven Seas
♥ 2003 Dreamworks SKG
Trailer: http://youtu.be/1KCX0pFPRwk
Eris scene: http://youtu.be/w1r8_vByXW4
2003 *Wired* article: http://bit.ly/gm85UU

Adapted from slide ♥ 2007 - 2008 K. Hawick, Massey University
159-235 Graphics and Graphical Programming, http://bit.ly/gmY8R8

MASSEY UNIVERSITY
TE KUNENGA KI PŪREHUROA

Further Reading

- Foley *et al.*: *Computer Graphics: Principles and Practice*
  - 2nd edition in C (1991), http://amzn.to/hFNqNC
  - Chapter 11: Representing Curves and Surfaces
- Approaches: Classical (OpenGL v1 & 2) vs. New (OpenGL v3 & 4)
  - Classical: `evalCoord` (http://bit.ly/e8olZj), `evalMesh` (http://bit.ly/gGkt8Z)
  - New: `Map{1|2}{f|d};` Chapter 5, compatibility profile (http://bit.ly/gkbVyE)
- *OpenGL 1.1 Specification*
  - All versions: http://www.opengl.org/documentation/specs/
  - Chapter 5: Evaluators, http://bit.ly/gMVzAO
- Red Book (*OpenGL Programming Guide*)
  - v1.1: http://glprogramming.com/red/ (current edition: 7th)
  - Chapter 12: Evaluators and NURBS, http://bit.ly/hZ1tpb
- Blue Book (*OpenGL Reference Manual*)
  - 1992 edition: http://glprogramming.com/blue/
  - See `evalCoord` (http://bit.ly/eADQLM), `evalMesh` (http://bit.ly/f7Juog)

Adapted from slides ♥ 2006 B. McCaul, Dublin City University
CA433 Computer Graphics I, http://bit.ly/ghw08y

Summary

- Reading for Last Class: §5.3 – 5.5, Eberly *2e*, CGA handout
- Reading for Today: §10.4, 12.7, Eberly *2e*, Mesh handout
- Reading for Next Class: §11.1 – 11.6 (736), Flash animation handout
- Last Time: Brief Survey of Skinning and Morphing
  - GPU-based vertex tweening: texture arrays, vertex texturing, hybrid
  - Agent simulation using GPU-based finite state machines
- Today: Curves & Surfaces
  - Piecewise linear, quadratic, cubic curves and their properties
  - Interpolation: subdivision (DeCasteljau's algorithm)
  - Bicubic surfaces & bilinear interpolation
- Outside Viewing
  - CIS 536 & 636 students: watch Basic CG lecture 10 on VSD
  - CIS 736 students: watch Advanced CG lectures 4 & 5 on CGA, IK
- Previous Videos: Morphing & Other Special Effects (SFX)
- Today's Videos: Bicubic Surfaces (NURBS), Solid Modeling

Terminology

- Skins – Surface Meshes for Faces, Character Models
- Morphing – gradual transition between images or meshes
  - Vertex tweening – texture arrays, vertex texturing, or hybrid method
  - GPU computing – offload some tasks to GPU
- Piecewise Polynomial Curves *aka* Splines
  - Piecewise linear, piecewise quadratic, piecewise cubic
  - Types of splines: Bézier, Hermite, B-splines, NURBS
  - DeCasteljau's algorithm: recursive linear interpolation (subdivision)
  - Control points: vertices of control polygon, determine spline shape
  - Bernstein polynomials: weight of each control point as function of t
- Continuity: Geometric (G^i), Mathematical (C^i)
- Bicubic Surfaces
  - Controlled by control patch (Coons patch), defining 3-D surface
  - Bilinear interpolation – sweep spline along another spline path
  - NURBS surface – bicubic surface based on NURBS curves