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

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

Instructor home page: http://www.cis.ksu.edu/~bhsu

Readings:
Today: §10.4, 12.7, Eberly

Next class: Flash animation handout

Reference on curves (required for CIS 736): §11.1 – 11.6, Eberly

Videos:
http://www.kddresearch.org/Courses/CIS636/Lectures/Videos/

Today: Curves & Surfaces

- Piecewise linear, quadratic, cubic surfaces and their properties
- Interpolation: subdivision (De Casteljau’s algorithm)
- Bicubic surfaces and 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

Review [1]: Morphing Targets

- Two key meshes are blended
- Varying by time
- Morph Targets
  - Represent by relative vectors
    - From base mesh
    - To target meshes
  - Geometry: mesh represents model
  - Applications: corresponding images

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
Introduction to CIS 536/636

Lecture 12, CIS 565 (formerly 665):
Adapted from “Morphing and Animation”

Pros & Cons of GPU Method 1

Advantages
- Keeps vertex, geometry processing units’ workload at minimum
- 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

Advantages
- 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

Disadvantages
- Vertex tweening supported on almost all hardware
- Modification algorithms performed on CPU, so no restrictions

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!)

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

Acknowledgements: Curves & Surfaces

Steve Rotenberg
Graphic Lat.
University of California – San Diego
http://graphic_lat.edu/

Barry McCaul
Lecturer
School of Computing
Dundalk Institute of Technology
http://www.computing.dcu.ie/~bmccaul/

Ken Hawick
Professor
Institute of Information and Mathematical Sciences (IIMS)
Massey University – Albany
http://www.computing.massey.ac.nz/kenhawick/

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

Acknowledgements: Splines

Jim Foley
Professor, College of Computing &
Stephen Fleming CNN in
Telecommunications
Georgia Institute of Technology
http://ietg.ca/folly

Andy van Dam
Chair, Computer Science &
Professor of Computer Science
Brown University
http://www.cs.brown.edu/~avad

Steve Feiner
Professor of Computer Science &
Director, Computer Graphics and User
Interfaces Laboratory
Columbia University
http://cglab.cs.columbia.edu/

John F. Hughes
Associate Professor of Computer
Science
Brown University
http://www.cs.brown.edu/~jfh/
Polynomial Functions

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

We usually define the curve for \( 0 \leq t \leq 1 \)

Vector Polynomials (Curves)

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

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 \( t \):

\[
\text{Lerp}(t, a, b) = (1-t)a + tb
\]

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 (alazing, 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
  - Really 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 in efficiency and tooling are now important.
- Used for:
  - Representing smooth shapes in 2D as outlines or in 3D using “patches” parameterized with two variables \( u \) and \( v \) (see slide 13).
  - Animation paths for “swaying” between keyframes.
  - Approximating “hermite” functions (polynomials are cheaper than beys, in con…).

Splines [3]: Hermite Curves

- Polylines are linear (or) order polynomial interpolations between points.
- Given points \( P \) and \( Q \) (see figure), there is a unique polynomial function \( f(\tau) \) that satisfies:
  \[
  f(0) = P, \quad f'(0) = \gamma \quad f(1) = Q, \quad f'(1) = \omega
  \]
- Hermite curves are higher-order polynomial interpolations between points.
- Linear interpolation but with higher order weighting functions allowing better approximation/smoothness.
- One representation - Hermite curves (interpolating splines)
  - Determined by two control points \( P \) and \( Q \), an initial tangent vector \( \gamma \) and a final tangent vector \( \omega \).
  - Satisfies:
    - \( f(0) = P \)
    - \( f'(0) = \gamma \)
    - \( f(1) = Q \)
    - \( f'(1) = \omega \)
Splines [4]: Hermite Weighting Explained

- Polynomial splines have more complex weighting functions than lines.
- Coefficients for P and Q are now 3rd degree polynomials.
- \( A \) at 0:
  - Coefficient of P is 1, all others 0.
  - Derivative of coefficient of P is 0.
- \( A \) at 1:
  - Coefficient of Q is 1, all others 0.
  - Derivative of coefficient of Q is 0.
- Can be chained together to make more complex curves.

Splines [5]: Bézier Curves

- Bézier representation is similar to Hermite.
  - 4 points instead of 5 points and 4 vectors (P0, ..., P3).
  - Initial position: P0, tangent vector is P1 - P0.
  - Final position: P3, tangent vector is P3 - P2.
- This representation allows a spline to be stored as a list of vertices with some global parameters that describe the smoothness and continuity.

Bézier Curves [1]: Piecewise Cubic Curves

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

Bézier Curves [2]: Formulation

- There are lots of ways to formulate Bézier curves mathematically. Some of these include:
  - de Casteljau (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.

Bézier Curves [3]: Interpolation Problem Defined

- Find the point \( x \) on the curve as a function of parameter \( t \).

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

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

- \( q_0 = \text{Lerp}(r_1, p_0, p_1) \)
- \( q_1 = \text{Lerp}(r_1, p_1, p_2) \)
- \( q_2 = \text{Lerp}(r_1, p_2, p_3) \)

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

- \( r_0 = \text{Lerp}(q_0, q_1) \)
- \( r_1 = \text{Lerp}(q_1, q_2) \)

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

- \( x = \text{Lerp}(r_0, r_1) \)

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

- \( x = \text{Lerp}(r_0, r_1) \)
- \( q_0 = \text{Lerp}(r_1, p_0, p_1) \)
- \( q_1 = \text{Lerp}(r_1, p_1, p_2) \)
- \( q_2 = \text{Lerp}(r_1, p_2, p_3) \)

Bernstein Polynomials [1]:
Coefficients of Control Points

- \( x = \frac{(1-t)(l_0)(1-t)(l_0+\xi_p)+\frac{(1-t)(l_0+\xi_p)/(1-t)}}{(1-t)(l_0+\xi_p)/(1-t)} + \frac{t(1-t)(l_0+\xi_p)}{(1-t)} \)
- \( x = \frac{(1-t)^3 l_0 + 3(l_0+\xi_p) + (4(1-t)^2 l_0+\xi_p) + \frac{t^3 l_0+\xi_p}}{(1-t)^3 l_0+\xi_p + t^3 l_0+\xi_p} \)
Bernstein Polynomials [2]: Piecewise Cubic Basis

\[ x = (-t^3 + 3t^2 - 3t + 1)p_0 + (3t^3 - 6t^2 + 3t)p_1 + (-3t^3 + 3t^2)p_2 + t^3p_3 \]

Bernstein Polynomials [3]: Binomial Form of Basis Functions

\[ B^3_0(t) = t^3 \]
\[ B^3_1(t) = -3t^3 + 3t^2 + 3t - t \]
\[ B^3_2(t) = 3t^3 - 4t^2 + t \]
\[ B^3_3(t) = -t^3 + 3t - 3t^3 + t \]
\[ \sum B^3_i(t) = 1 \]

Bernstein Polynomials [4]: Cubic Matrix Form

\[ x = \begin{pmatrix} 1 & t & t^2 & t^3 \end{pmatrix} \begin{pmatrix} a & b & c & d \end{pmatrix} \]

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

- **Geometric Continuity:**
  - \(G^0\): 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

- **Mathematical Continuity:**
  - \(C^0\): 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

Connecting Bézier Curves: \(C^i\) Continuity

- A simple way to make larger curves is to connect up Bézier curves
- Consider two Bézier curves defined by \(p_0\) to \(p_3\) and \(p_0\) to \(v_0\)
- If \(p_0 = p_3\), then they will have \(C^0\) continuity
- If \(p_0 \neq p_3\), then they will have \(C^1\) continuity
- \(C^1\) continuity is more difficult...

Building 3-D Primitives

- Made out of 2D and 3D primitives
- Triangles are commonly used
- Many triangles used for a single object is a triangular mesh
- Splines used to describe boundaries of “patches” — these can be “sewn together” to represent curved surfaces
  - \((x_1, y_1) = (1 - t^2)(1 - s^2), \quad x_2, \quad y_2 = (1 - s^2), \quad x_3, \quad y_3 = (1 - t^2), \quad x_4, \quad y_4 = \ldots \)
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

2. Implicit

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.

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

\[ f(x, y) = 0 \]
\[ A x + B y + C = 0 \]

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?)

Types of Curves [1]: Explicit & Implicit

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

- VERTEX TABLE
- EDGE TABLE
- POLYGON TABLE

OR

Types of Curves [2]: Parametric

1. Cubic polynomials that define curve segment \( Q(t)= \{ x(t), y(t) \} \) of form:

\[
\begin{align*}
Q(t) &= \text{coefficients} \\
&= a t^3 + b t^2 + c t + d
\end{align*}
\]

Written in matrix form, system becomes

\[
Q(t) = \begin{bmatrix}
A & B \\
C & D
\end{bmatrix}
\begin{bmatrix}
t^3 \\
t^2 \\
t \\
1
\end{bmatrix}
\]

where

\[
C = \begin{bmatrix}
a & b & c & d
\end{bmatrix}
\]

Parametric Curves

2.gon 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.

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

\[ f(x, y) = 0 \]
\[ A x + B y + C = 0 \]

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?)

Types of Curves [1]: Explicit & Implicit

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

- VERTEX TABLE
- EDGE TABLE
- POLYGON TABLE

OR

Types of Curves [2]: Parametric

1. Cubic polynomials that define curve segment \( Q(t)= \{ x(t), y(t) \} \) of form:

\[
\begin{align*}
Q(t) &= \text{coefficients} \\
&= a t^3 + b t^2 + c t + d
\end{align*}
\]

Written in matrix form, system becomes

\[
Q(t) = \begin{bmatrix}
A & B \\
C & D
\end{bmatrix}
\begin{bmatrix}
t^3 \\
t^2 \\
t \\
1
\end{bmatrix}
\]

where

\[
C = \begin{bmatrix}
a & b & c & d
\end{bmatrix}
\]
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 sweeping through \( t \), in our case \( 0 \leq t \leq 1 \), we can evaluate equations and determine \( x \), \( y \) values for points on curve

![Control of Surface Shape](https://example.com/control_of_surface_shape.png)

- Control is now 2-D array of control points
- Two parameter surface function, forming tensor product with blending functions, is:
  \[
  X(s, t) = \sum f_i(s) f_j(t) q_{ij}
  \]
- Similarly for \( Y(s, t) \) 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

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
  \[
  0 \leq s \leq 1 \quad \text{and} \quad 0 \leq t \leq 1
  \]

Example: Bézier Surface

- Matrix formulation as follows
  \[
  x(s, t) = s^3 . M_3 d_3 . M_3^T . t \\
  y(s, t) = s^3 . M_3 d_3 . M_3^T . t \\
  z(s, t) = s^3 . M_3 d_3 . M_3^T . t
  \]
- \( q \) is \( 4 \times 4 \) array of \( x \) coords
- \( q \) is \( 4 \times 4 \) array of \( y \) coords
- \( q \) is \( 4 \times 4 \) 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 Bzier
- 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 (Bzier too)

Continuity of Bicubic Patches

- Hermite and Bzier patches
  - C1 continuity when sharing 4 control points between patches
  - C2 continuity when both sets of control points either side of the edge are collinear with the edge
- B-Spline patch
  - C2 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
- NURBS

Curves & Surfaces: Summary

- Curves
  - Bzier: 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.

Further Reading

- Foley et al.: Computer Graphics: Principles and Practice
  - Chapter 11: Representing Curves and Surfaces
- Approaches: Classical (OpenGL v1 & 2) vs. New (OpenGL v3 & 4)
- OpenGL 1.1 Specification

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

- Reading for Last Class: §5.3 – 5.5, Eberly 2\textsuperscript{e}, CGA handout
- Reading for Today: §10.4, 12.7, Eberly 2\textsuperscript{e}, 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), Mathematical (C)
- 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