6.837 Introduction to Computer Graphics

Curves and Surfaces
Cubic Bezier splines

- \[ P(t) = (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2(1-t) P_3 + t^3 P_4 \]
Bernstein polynomials

For cubic:
- \( B_1(t) = (1-t)^3 \)
- \( B_2(t) = 3t(1-t)^2 \)
- \( B_3(t) = 3t^2(1-t) \)
- \( B_4(t) = t^3 \)

- (careful with indices, many authors start at 0)

- But defined for any degree
General spline formulation

\[ Q(t) = GBT(t) = \text{Geometry } G \cdot \text{Spline Basis } B \cdot \text{Power Basis } T(t) \]

- Geometry: control points coordinates assembled into a matrix \((P_1, P_2, \ldots, P_{n+1})\)
- Power basis: the monomials \(T(t^n, t^{n-1}, \ldots t^2, t, 1)\)
- Bezier:

\[
P(t) = \begin{pmatrix}
P_1,x & P_2,x & P_3,x & P_4,x \\
P_1,y & P_2,y & P_3,y & P_4,y
\end{pmatrix} \begin{pmatrix}
-1 & 3 & -3 & 1 \\
3 & -6 & 3 & 0 \\
-3 & 3 & 0 & 0 \\
1 & 0 & 0 & 0
\end{pmatrix} \begin{pmatrix}
t^3 \\
t^2 \\
t \\
1
\end{pmatrix}
\]
General spline formulation

\[ Q(t) = GBT(t) = \text{Geometry } G \cdot \text{Spline Basis } B \cdot \text{Power Basis } T(t) \]

- Geometry: control points coordinates assembled into a matrix \((P_1, P_2, \ldots, P_{n+1})\)
- Spline basis: defines the type of spline
  - Bernstein for Bezier
- Power basis: the monomials \(T(t^n, t^{n-1}, \ldots t^2, t, 1)\)
- Advantage of general formulation
  - Compact expression
  - Easy to convert between types of splines
Cubic BSplines: basis

\[ Q(t) = \frac{(1-t)^3}{6} P_{i-3} + \frac{3t^3 - 6t^2 + 4}{6} P_{i-2} + \frac{-3t^3 + 3t^2 + 3t + 1}{6} P_{i-1} + \frac{t^3}{6} P_i \]

\[ Q(t) = \text{GBT}(t) \]

\[ B_{B-Spline} = \frac{1}{6} \begin{pmatrix}
-1 & 3 & -3 & 1 \\
3 & -6 & 0 & 4 \\
-3 & 3 & 3 & 1 \\
1 & 0 & 0 & 0
\end{pmatrix} \]
Bézier is not the same as BSpline

• Relationship to the control points is different
Converting between Bézier & BSpline

original control points as Bézier

new BSpline control points to match Bézier

new Bézier control points to match BSpline

original control points as BSpline
Converting between Bézier & BSpline

• Using the basis functions:

\[ B_{\text{Bezier}} = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \]

\[ B_{\text{BSpline}} = \frac{1}{6} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 0 & 4 \\ -3 & 3 & 3 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \]

\[ Q(t) = GBT(t) = \text{Geometry } G \cdot \text{Spline Basis } B \cdot \text{Power Basis } T(t) \]
Questions?
Representing surfaces

- Triangle meshes
  - 3D version of polylines
- Tensor Splines
  - 3D version of splines
- Subdivision surfaces
- Implicit surfaces
  - \( f(x,y,z)=0 \)
- Procedural
  - e.g. surfaces of revolution, generalized cylinder
- From volume data (medical images, etc.)
Conversion

• Often need to convert
  – Modeling is often easier with higher-level primitives (e.g. splines, surfaces of revolution)
  – rendering likes triangles

• Tessellation is the act of turning a smooth surface into facets.
Triangle meshes

- What you’ve used so far in Asst 0
- Triangle represented by 3 vertices (3 triplets of coordinates)
- Can be grouped in indexed face sets where vertices are shared
  - First list of triplet of coordinates for vertices
  - Then list of tuples of indices for facets
- Pro: simple, can be rendered directly
- Cons: not smooth, needs many triangles to approximate smooth surfaces
Questions?
Tensor spline patches

- Parametric surface $P(u,v)$ is a cubic polynomial of two variables $u$ & $v$
- Defined by $4 \times 4 = 16$ control points $P_{1,1}$, $P_{1,2}$,..., $P_{4,4}$
- Interpolates 4 corners, approximates others
Tensor spline patches

- Defined by $4 \times 4 = 16$ control points $P_{1,1}, P_{1,2}, ..., P_{4,4}$
- Basis are product of two Bernstein polynomials: $B_1(u)B_1(v); B_1(u)B_2(v); ... B_4(u)B_4(v)$
Tensor spline patches

• Pros:
  – Smooth
  – Defined by reasonably small set of points (knots)

• Cons
  – Harder to render (usually converted to triangles)
  – Tricky to ensure continuity at patch boundary

• Extensions
  – Rational splines: splines in homogeneous coordinates
  – NURBS: Non-Uniform Rational B-Splines
    • More crazy formula (ratio of polynomials, non-uniform location of control points)
Utah teapot: tensor Bezier splines

• Designed by Martin Newell
Questions?
Subdivision surfaces

- Start with polygonal mesh
- Subdivide into larger number of polygons & smooth
- The limit surface is smooth
Illustration: subdivision curve
Corner Cutting
Corner Cutting
Corner Cutting
Corner Cutting
Corner Cutting
Corner Cutting
Corner Cutting

A control point

The limit curve

The control polygon

Overview

Slide by Adi Levin
Subdivision curves and surfaces

- Idea: cut corners to smooth
- Add points and compute weighted average of neighbors
- Same for surfaces
  - Special case for irregular vertices
    - vertex with more or less than 6 neighbors in a triangle mesh
Subdivision curves and surfaces

• Advantages
  – Arbitrary topology
  – Smooth at boundaries
  – Level of detail, scalable
  – Simple representation
  – Numerical stability, well-behaved meshes
  – Code simplicity

• Little disadvantage:
  – Procedural definition
  – Not parametric, not implicit
  – Tricky at special vertices

Warren et al.
Questions?
Implicit surfaces

- Implicit definition: \( f(x,y,z) = 0 \)
  e.g. for a sphere: \( x^2 + y^2 + z^2 = R^2 \)
- Often defined as metaballs with seed points
- \( f(x,y,z) = f_1(x,y,z) + f_2(x,y,z) + \ldots \)
  – where \( f_i \) depends on distance to a seed point \( P_i \)

![Diagram of metaballs with seed points](image)
Interpretation

- Isosurface of a higher-dimensional function
Implicit surfaces

• Pros:
  – Can handle weird topology for animation
  – Easy to do sketchy modeling
  – Some data comes this way (medical & scientific data)

• Cons:
  – Does not allow us to easily generate a point on the surface
Questions?
Specialized Procedural Definitions

- **Surfaces of revolution**
  - Given 2D profile

- **Generalized cylinders**
  - Given 2D profile and 3D curve
Surface of revolution

- 2D curve $q(u)$ provides one dimension
  - Note: works also with 3D curve
- Rotation $R(v)$ provides 2nd dimension

$s(u,v) = R(v)q(u)$

where $R$ is a matrix, $q$ a vector, and $s$ is a point on the surface.
General Sweep Surfaces

- Trace out surface by moving a profile curve along a trajectory.
  - profile curve $q(u)$ provides one dimension
  - trajectory $c(u)$ provides the other
- Surface of revolution can be seen as a special case where trajectory is a circle

$$s(u,v) = M(c(v))q(u)$$

where $M$ is a matrix that depends on the trajectory $c$
General Sweep Surfaces

- How do we get $M$?
  - Location is easy, given by $c(v)$
  - What about orientation?

- Orientation options:
  - Align profile curve with an axis.
  - Align profile curve with frame that “follows” the curve

$$s(u,v) = M(c(v))q(u)$$

where $M$ is a matrix that depends on the trajectory $c$
Questions
Differential properties of curves

• Motivation
  – Define orientation for swept surfaces
  – Compute velocity for animation
  – Compute normal for surfaces
  – Analyze smoothness
Velocity

- First derivative wrt $t$
- Can you compute this for Bezier curves?

\[ P(t) = (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2(1-t) P_3 + t^3 P_4 \]

\[ P'(t) = -3(1-t)^2 P_1 + [3(1-t)^2 - 6t(1-t)] P_2 + [6t(1-t) - 3t^2] P_3 + 3t^2 P_4 \]

Sanity check: $t=0$; $t=1$
The tangent to the curve $P(t)$ can be defined as $T(t) = \frac{P'(t)}{||P'(t)||}$
- normalized velocity

This provides us with one orientation for swept surfaces
Questions?
Curvature

• Derivative of tangent
  – $K(t) = T'(t)$
  – Constant for a circle
  – Zero for a straight line

• Always orthogonal to tangent
  – Because tangent is unit-length
Normal

• Normalized curvature: $\frac{T'(t)}{||T'(t)||}$
Frenet frame

• Recall high-school physics?
• Frame defined by 1st (tangent), 2nd (curvature) and 3rd (torsion) derivatives of a 3D curve
• Looks like a good idea for swept surfaces...
Problem at inflection

- Normal flips!
- Bad to define a smooth swept surface
Questions?
Constructing smooth frames

- Tangent is assumed reliable
- Build triplet of vectors
  - include tangent
  - orthonormal
  - coherent over the curve
- Idea:
  - use cross product to create orthogonal vectors
  - exploit discretization of curve
  - use previous frame to bootstrap orientation
Problem

- We seek $N_i$, $B_i$, $T_i$ basis vectors for frame $i$
- We know them for the previous frame $i-1$
- We know $T_i$
Algorithm

• “Pretend” \( B_i \) is the same as \( B_{i-1} \)
  – This will ensure continuity
  – Note, though, that \( B_{i-1} \) is usually not orthogonal to \( T_i \)

• Then we know \( T_i \) and “\( B_i \)”, we can deduce \( N_i \):
  – \( N_i = \text{normalize}(B_{i-1} \times T_i) \)

• Now we need to fix “\( B_i \)”:
  just use \( N_i \) and \( T_i \)
  – \( B_i = \text{normalize}(T_i \times N_i) \)

Sanity check?
Standard graphics trick

- When you need to get an orthonormal frame
- Start with some reasonable guess
- Do enough cross products/normalizations to make sure everything is orthonormal.
Other use of the trick

• When defining a camera
  – provide viewing direction
  – Need to define up vector

• Very tedious to provide up vector orthogonal to viewing direction
  – usually provide it approximately, then do cross products
Questions?