Overview

Part 1: Curves

Part 2: Surfaces

Przemyslaw Prusinkiewicz

Curves

- 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

Many applications in graphics

- Fonts ABC
- Animation paths
- Shape modeling
- etc…

Goals

- Some attributes we might like to have:
  - Predictable control
  - Multiple values
  - Local control
  - Versatility
  - Continuity
- We’ll satisfy these goals using:
  - Piecewise
  - Parametric
  - Polynomials

Parametric curves

A parametric curve in the plane is expressed as:

\[ x = x(u) \]
\[ y = y(u) \]

Example: a circle with radius r centered at origin:

\[ x = r \cos u \]
\[ y = r \sin u \]

In contrast, an implicit representation is:

\[ x^2 + y^2 = r^2 \]
Parametric polynomial curves

- A parametric polynomial curve is described:
  \[ x(u) = \sum_{i=0}^{n} a_i u^i \]
  \[ y(u) = \sum_{i=0}^{n} b_i u^i \]

- Advantages of polynomial curves
  - Easy to compute
  - Infinitely differentiable

Piecewise parametric polynomials

- Use different polynomial functions on different parts of the curve
  - Provides flexibility
  - How do you guarantee smoothness at “joints”? (continuity)

- In the rest of this lecture, we’ll look at:
  - Bézier curves: general class of polynomial curves
  - Splines: ways of putting these curves together

Bézier curves

- Developed independently in 1960s by
  - Bézier (at Renault)
  - deCasteljau (at Citroen)

- Curve \( Q(u) \) is defined by nested interpolation:

![Bézier curve diagram]

\( V_0, V_1, \ldots, V_n \) is control polygon

Explicit formulation

- Let’s indicate level of nesting with superscript \( j \):
- An explicit formulation of \( Q(u) \) is given by:
  \[ V_i^j = (1-u)V_i^{j-1} + uV_{i+1}^{j-1} \]

- Case \( n=2 \) (quadratic):
  \[ Q(u) = V_2^1 \]
  \[ = (1-u)V_0^1 + uV_1^1 \]
  \[ = (1-u)((1-u)V_0^0 + uV_1^0) + u((1-u)V_1^0 + uV_2^0) \]
  \[ = (1-u)V_0^0 + 2u(1-u)V_1^0 + u^2V_2^0 \]

More properties

- General case: Bernstein polynomials
  \[ Q(u) = \sum_{i=0}^{n} \binom{n}{i} u^i (1-u)^{n-i} \]

- Degree: polynomial of degree \( n \)

- Tangents:
  \[ Q'(0) = n(V_1^0 - V_0^0) \]
  \[ Q'(1) = n(V_n^0 - V_{n-1}^0) \]
Cubic curves

- From now on, let’s talk about cubic curves (n=3)
- In CAGD, higher-order curves are often used
- In graphics, piecewise cubic curves will do
  - Specified by points and tangents
  - Allows specification of a curve in space
- All these ideas generalize to higher-order curves

Matrix form

Bézier curves may be described in matrix form:

\[ Q(u) = \sum_{i=0}^{3} p_i \binom{n}{i} u^i (1-u)^{n-i} \]

\[ = (1-u)^3 V_0 + 3u(1-u)^2 V_1 + 3u^2(1-u) V_2 + u^3 V_3 \]

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

Display

Q: How would you draw it using line segments?
A: Recursive subdivision!

Flatness

Q: How do you test for flatness?
A: Compare the length of the control polygon to the length of the segment between endpoints

Splines

- For more complex curves, piece together Béziers
- We want continuity across joints:
  - Positional (C0) continuity
  - Derivative (C1) continuity
- Q: How would you satisfy continuity constraints?
- Q: Why not just use higher-order Bézier curves?
A: Splines have several of advantages:
  - Numerically more stable
  - Easier to compute
  - Fewer bumps and wiggles
**Catmull-Rom splines**

- **Properties**
  - Interpolate control points
  - Have C₀ and C¹ continuity
- **Derivation**
  - Start with joints to interpolate
  - Build cubic Bézier between each joint
  - Endpoints of Bézier curves are obvious
- **What should we do for the other Bézier control points?**

**Matrix formulation**

Convert from Catmull-Rom CP's to Bezier CP's:

\[
\begin{pmatrix}
B_0 \\
B_1 \\
B_2 \\
B_3
\end{pmatrix} = \begin{pmatrix}
0 & 6 & 0 & 0 \\
0 & -1 & 6 & 1 \\
6 & 0 & 1 & 6 \\
0 & 0 & 6 & 0
\end{pmatrix} \begin{pmatrix}
V_0 \\
V_1 \\
V_2 \\
V_3
\end{pmatrix}
\]

*Exercise:* Derive this matrix.
(Hint: in this case, \(\tau\) is not 1/2.)

**B-splines**

- We still want local control
- Now we want C² continuity
- Give up interpolation
- It turns out we get convex hull property
- **Constraints:**
  - Three continuity conditions at each joint
  - Position of two curves same
  - Derivative of two curves same
  - Second derivatives same
  - Local control
  - Each joint affected by 4 CPs

**Matrix formulation for B-splines**

Grind through some messy math to get:

\[
Q(u) = [u^3 \quad u^2 \quad u] \begin{pmatrix}
-1 & 3 & -3 & 1 \\
3 & -6 & 3 & 0 \\
-3 & 0 & 3 & 0 \\
1 & 4 & 1 & 0
\end{pmatrix} \begin{pmatrix}
V_0 \\
V_1 \\
V_2 \\
V_3
\end{pmatrix}
\]
Curved Surfaces

• Motivation
  ◦ Exact boundary representation for some objects
  ◦ More concise representation than polygonal mesh

Curved Surfaces

• What makes a good surface representation?
  ◦ Accurate
  ◦ Concise
  ◦ Intuitive specification
  ◦ Local support
  ◦ Affine invariant
  ◦ Arbitrary topology
  ◦ Guaranteed continuity
  ◦ Natural parameterization
  ◦ Efficient display
  ◦ Efficient intersections

Curved Surface Representations

• Polygonal meshes
• Subdivision surfaces
• Parametric surfaces
• Implicit surfaces

Curved Surface Representations

• Polygonal meshes
• Subdivision surfaces
• Parametric surfaces
• Implicit surfaces

Parametric Surfaces

• Boundary defined by parametric functions:
  ◦ \( x = f_1(u,v) \)
  ◦ \( y = f_2(u,v) \)
  ◦ \( z = f_3(u,v) \)

• Example: ellipsoid
  \[
  x = r_c \cos \phi \cos \theta \\
  y = r_c \cos \phi \sin \theta \\
  z = r_c \sin \phi
  \]

Surface of revolution

• Idea: take a curve and rotate it about an axis
**Swept surface**

Idea: sweep one curve along path of another curve

---

**Parametric Surfaces**

Advantage: easy to enumerate points on surface.

Disadvantage: need piecewise-parametric surface to describe complex shape.

---

**Piecewise Parametric Surfaces**

Surface is partitioned into parametric patches:

Same ideas as parametric splines!

---

**Parametric Patches**

• Each patch is defined by blending control points

Same ideas as parametric curves!

---

**Parametric Patches**

• Point $Q(u,v)$ on the patch is the tensor product of parametric curves defined by the control points

---

**Parametric Bicubic Patches**

Point $Q(u,v)$ on any patch is defined by combining control points with polynomial blending functions:

$$Q(u,v) = UM \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \\ P_{41} & P_{42} & P_{43} & P_{44} \end{bmatrix} M^TV^T$$

Where $M$ is a matrix describing the blending functions for a parametric cubic curve (e.g., Bezier, B-spline, etc.)
B-Spline Patches

\[ Q(u, v) = U M_{\text{B-spline}} \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \\ P_{41} & P_{42} & P_{43} & P_{44} \end{bmatrix} M_{\text{B-spline}}^T V \]

\[ M_{\text{B-spline}} = \begin{bmatrix} -\frac{1}{6} & \frac{1}{2} & \frac{1}{2} & \frac{1}{6} \\ \frac{1}{2} & -1 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{6} & \frac{1}{2} & \frac{1}{2} & \frac{1}{6} \end{bmatrix} \]

Watt Figure 6.28

Bezier Patches

\[ Q(u, v) = U M_{\text{Bezier}} \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \\ P_{41} & P_{42} & P_{43} & P_{44} \end{bmatrix} M_{\text{Bezier}}^T V \]

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

Watt Figure 6.29a

Bezier Patches

- Properties:
  - Interpolates four corner points
  - Convex hull
  - Local control

Watt Figure 6.22

Bezier Surfaces

- Continuity constraints are similar to the constraints Bezier splines

Watt Figure 6.26a

Bezier Surfaces

- \( C^0 \) continuity requires aligning boundary curves

Watt Figure 6.26b

Bezier Surfaces

- \( C^1 \) continuity requires aligning boundary curves and derivatives (a reason to prefer subdiv. surf.)

Watt Figure 6.26b
Drawing Bezier Surfaces

• Simple approach is to loop through uniformly spaced increments of \( u \) and \( v \)

```c
void DrawSurface(void)
{
    for (int i = 0; i < imax; i++) {
        float u = umin + i * ustep;
        for (int j = 0; j < jmax; j++) {
            float v = vmin + j * vstep;
            DrawQuadrilateral(...);
        }
    }
}
```

Watt Figure 6.32

• Better approach is to use adaptive subdivision:

```c
void DrawSurface(surface)
{
    if (Flat(surface, epsilon)) {
        DrawQuadrilateral(surface);
    }
    else {
        SubdivideSurface(surface, ...);
        DrawSurface(surfaceLL);
        DrawSurface(surfaceLR);
        DrawSurface(surfaceRL);
        DrawSurface(surfaceRR);
    }
}
```

Uniform subdivision

Watt Figure 6.32

Adaptive subdivision

Drawing Bezier Surfaces

• One problem with adaptive subdivision is avoiding cracks at boundaries between patches at different subdivision levels

Avoid these cracks by adding extra vertices and triangulating quadrilaterals whose neighbors are subdivided to a finer level.

Watt Figure 6.33

Parametric Surfaces

• Advantages:
  o Easy to enumerate points on surface
  o Possible to describe complex shapes

• Disadvantages:
  o Control mesh must be quadrilaterals
  o Continuity constraints difficult to maintain
  o Hard to find intersections

Blender (www.blender.nl)