Review of Last Thursday

• Texture mapping; Aliasing caused by sampling

\[
\begin{align*}
(0,0) & \quad (0,1) \\
(1,0) & \quad (1,1)
\end{align*}
\]

• Triangles are suitable input, but not optimal for modeling
  • Explicit, implicit, and parametric functions

\[
y = f(x) \quad f(x, y) = 0 \quad f(u) = (x(u), y(u), z(u))
\]
• Parametric curves: Bézier curves

\[ f(u) = \sum_{i=0}^{n} b_{in}(u)p_i \]

\[ b_{in}(u) = \binom{n}{i} u^i (1 - u)^{n-i} \]

\[ \sum_{i=0}^{n} b_{in}(u) = 1 \]

convex combination

\[ 0 \leq b_{in}(u) \leq 1 \]

convex hull property
Review of last Thursday

- **Subdivision** – example: cubic Bézier curve

\[
p_i^k(u) = (1 - u) \cdot p_{i-1}^{k-1} + u \cdot p_i^{k-1}
\]

- 
  \[(p_0^0, p_1^1, p_2^2, p_3^3)\] Cubic curve 1
  \[(p_3^3, p_2^3, p_1^3, p_0^3)\] Cubic curve 2
Review of last Thursday

- **Derivatives**

\[ f(u) = \sum_{i=0}^{n} b_{in}(u) \cdot p_i \]

Derivatives are tangents.

Lighting computations require derivatives.
Example: Utah Teapot consists of bicubic Bézier patches.
Review of last Thursday

• Derivatives

\[
\frac{df(u)}{du} = \sum_{i=0}^{n} n(b_{i-1,n-1}(u) - b_{i,n-1}(u)) \cdot p_i
\]

\[
\frac{df(u)}{du} = \sum_{i=0}^{n-1} b_{i,n-1}(u) \cdot n(p_{i+1} - p_i)
\]

Derivative is a Bézier curve of order (n-1) with “combined” control points
Review of last Thursday

- **Joining** Bézier curves

\[ p_0, p_1, p_2, p_3 \quad \text{and} \quad q_0, q_1, q_2, q_3 \]

**C^0 continuity:**
- \( f \) continuous
- \( f' \) discontinuous

**C^1 continuity:**
- \( f \) continuous
- \( f' \) continuous

Discontinuous
Bézier Curves

• **Interpolation** with Bézier curves
  • Choose parameters for “curve points” and solve for control points
  • Example: four points = cubic Bézier curve

\[
\begin{align*}
  f(0) &= f_0 & \rightarrow & \quad p_0 = f_0 \\
  f(1) &= f_3 & \rightarrow & \quad p_3 = f_3 \\
  f(0.33) &= f_1 \\
  f(0.67) &= f_2
\end{align*}
\]

\[
\begin{align*}
  u &= 0.33 \\
  u &= 0.67 \\
  u &= 1
\end{align*}
\]
Bézier Curves

- Interpolation with Bézier curves

Interpolation conditions

\[ f(0) = f_0 \rightarrow p_0 = f_0 \]
\[ f(1) = f_3 \rightarrow p_3 = f_3 \]

Linear system

\[
\begin{pmatrix}
  f_0 \\
  f_1 \\
  f_2 \\
  f_3
\end{pmatrix}
= 
\begin{pmatrix}
  b_{03}(0) & b_{13}(0) & b_{23}(0) & b_{33}(0) \\
  b_{03}(0.33) & b_{13}(0.33) & b_{23}(0.33) & b_{33}(0.33) \\
  b_{03}(0.67) & b_{13}(0.67) & b_{23}(0.67) & b_{33}(0.67) \\
  b_{03}(1) & b_{13}(1) & b_{23}(1) & b_{33}(1)
\end{pmatrix}
\begin{pmatrix}
  p_0 \\
  p_1 \\
  p_2 \\
  p_3
\end{pmatrix}
\]

\[
\begin{pmatrix}
  p_0 \\
  p_1 \\
  p_2 \\
  p_3
\end{pmatrix}
= 
\begin{pmatrix}
  b_{03}(0) & b_{13}(0) & b_{23}(0) & b_{33}(0) \\
  b_{03}(0.33) & b_{13}(0.33) & b_{23}(0.33) & b_{33}(0.33) \\
  b_{03}(0.67) & b_{13}(0.67) & b_{23}(0.67) & b_{33}(0.67) \\
  b_{03}(1) & b_{13}(1) & b_{23}(1) & b_{33}(1)
\end{pmatrix}^{-1}
\begin{pmatrix}
  f_0 \\
  f_1 \\
  f_2 \\
  f_3
\end{pmatrix}
\]
Bézier Curves

• Summary:
  • Control point approximation (Endpoint interpolation)
  • Convex hull property/Bounding-box property
  • Compact representation
  • Subdivision for rendering
  • Easy manipulation
  • Affine invariance (affine transformation of control points results in affine transformation of Bézier curve)
B-Splines

- Joining Bézier curves is cumbersome
- Number of control points directly influences degree
- High degree: Unstable curves

**Solution:** B-Splines are defined on local sets of points
B-Splines

- **Solution:** B-Splines are defined on local sets of points

Given $m + 1$ control points $p_i \in \{p_0, \ldots, p_m\}$

Construct B-Spline with degree $n$

**Knot vector** $u_i \in \{u_0, \ldots, u_{m+n+1}\}$ defines parameter values

$$f(u) = \sum_{i=0}^{m} B_{in}(u) \cdot p_i \quad \text{defined on } [u_n, \ldots, u_{m+1}]$$
B-Splines

- B-Spline base functions

\[ f(u) = \sum_{i=0}^{m} B_{in}(u) \cdot p_i \]

\[ B_{k0}(u) = \begin{cases} 
1, & u_k \leq u \leq u_{k+1} \\
0, & \text{otherwise}
\end{cases} \]

\[ B_{kn}(u) = \frac{u - u_k}{u_{k+n} - u_k} B_{k,n-1}(u) + \frac{u_{k+n+1} - u}{u_{k+n+1} - u_{k+1}} B_{k+1,n-1}(u) \]
• B-Spline base functions have local influence/support

\[ B_{k0}(u) = \begin{cases} 
1, & u_k \leq u \leq u_{k+1} \\
0, & \text{otherwise}
\end{cases} \]

\[ B_{kn}(u) = \frac{u - u_k}{u_{k+n} - u_k} B_{k,n-1}(u) + \frac{u_{k+n+1} - u}{u_{k+n+1} - u_{k+1}} B_{k+1,n-1}(u) \]
B-Splines

- Knot vector examples (for m = 3, n = 3)

  uniform knot vector \([0, 1, 2, 3, 4, 5, 6, 7]\)
  B-Spline defined on \(u \in [3, 4]\)

  end knot multiplicity of \(n+1\) leads to interpolation

  Non-uniform knot vector \([0, 0, 0, 0, 1, 1, 1, 1]\)
  B-Spline defined on \(u \in [0, 1]\)
  cubic B-Spline identical to cubic Bézier curve
B-Spline Properties

- B-Spline has (local) convex hull property
  \[ \sum_{i=0}^{m} B_{in}(u) = 1 \quad 0 \leq B_{in}(u) \leq 1 \]

- B-Spline is piecewise curve with local control
- \( C^{n-k} \) continuous at knot with multiplicity \( k \)
- Bézier curve is special case of B-Spline
- Affine invariance
Parametric Surfaces

- Extend notion to 3D – two parameters instead of one

\[ f(u, v) = \sum_{i}^{n} \sum_{j}^{m} b_i(u)b_j(v)p_{ij} \]
Rendering Parametric Curves

- Triangles and lines are suitable for rendering

- Flexible triangle mesh generation
  a) Create vertices by sampling the curve/surface along its parameter u (or u,v)
  b) Use subdivision to approximate curve by its convex hull

- Alternative: Ray-Casting
Parametric Curves - Summary

- Multiple established definitions
- Curves defined by control points and parameter values
- Suitable for modeling/editing of complex shapes
- Control over continuity, smoothness
- Rendering requires sampling or subdivision
Implicit Representations

- Object surface is the (zero) level set of a function

\[ f(x, y) = 0 \]

Contour defined by zero level set:

“All (x,y) locations, where function evaluates to 0”

Divides space into regions (positive sign and negative sign)
Implicit Representations

- Easy curve/surface/volume operations
- Examples:

\[ f(x, y, z) = 0 \quad \text{Surface 1} \quad g(x, y, z) = 0 \quad \text{Surface 2} \]

\[ f(x, y, z) = 0 \quad \text{AND} \quad g(x, y, z) = 0 \quad \text{Intersection curve} \]

\[ f(x, y, z) > 0 \quad \text{AND} \quad g(x, y, z) < 0 \quad \text{Volume subtraction} \]

\[ f(x, y, z) > 0 \quad \text{OR} \quad g(x, y, z) < 0 \quad \text{Volume union} \]
Implicit Representations

- Subdivision and rasterization useful for parametric representations
- Triangulation of implicit representations is harder

\[ f(x, y) = (x-mx)^2 + (y-my)^2 - r^2 \]
Triangulating Implicit Representations

- Implicit function given in discretized form
- Extract curve that corresponds to $f(x,y) = 0$

$$f(x,y) = (x-mx)^2 + (y-my)^2 - r^2$$
Triangulating Implicit Representations

- Extract curve that corresponds to \( f(x,y) = 0 \)
- Thresholding?

\[
f(x,y) = (x-mx)^2 + (y-my)^2 - r^2
\]
Triangulating Implicit Representations

• Extract curve that corresponds to \( f(x,y) = 0 \)

• Better: **Marching Quads**
  • Fast computation of curve intersection with grid edges
  • Piecewise linear representation of curve
  • Output suitable for rendering
  • Generalizes to 3D (**Marching Cubes**)
Triangulating Implicit Representations

- Better: **Marching Quads**
- Examine corner values of individual cells (quads)
- Compare corner values to function value (**isovalue**)

![Diagram of a square with corners labeled as + and -]
Triangulating Implicit Representations

- 16 cases for 2D Marching Quads
Triangulating Implicit Representations

- **Marching Quads**
- Lookup table for triangulation cases
- Creates geometry per cell/quad
- Piecewise linear representation

- Linear interpolation along edges

\[ p_{edge} = (1 - t) \cdot p_0 + t \cdot p_1 \]
\[ t = \frac{f_{iso} - f_0}{f_1 - f_0} \]
Triangulating Implicit Representations

- **Marching Quads** – Summary
  - Approximation quality dependent on sampling
  - Non-uniform triangle sizes/line lengths
  - Only linear representation per cell (connectivity information?)
  - Highly parallelizable
  - Seeding strategies possible
Raytracing Implicit Representations

• Alternative: Raytracing

Image plane
Ray through pixel center

\[ f(x,y) < 0 \]

\[ f(x,y) > 0 \]
Rendering Implicit Representations

- Raytracing: Follow ray after first intersection
Creation of Object Representations

• Modeling and Animation Tools (CAD, CSG, …)
  • Blender, Maya, 3D Studio Max, etc.

• Procedural generation
• Reconstruction from measurements
Summary of Chapter 3

• Different object representations may be used

• For efficient rendering and rasterization functions are tesselated into triangles

• Vertices, lines, triangles allow for quick rasterization and interpolation

• Attributes of vertices define colors, textures, etc.

• Up next: What can we “do” to these objects?