Today

- General curve and surface representations
- Splines and other polynomial bases
Geometry Representations

- Constructive Solid Geometry (CSG)
- Parametric
  - Polygons
  - Subdivision surfaces
- Implicit Surfaces
- Point-based Surface

Not always clear distinctions
- *i.e. CSG done with implicit*
Geometry Representations

Object made by CSG
Converted to polygons
Converted to implicit surface

Geometry Representations

CSG on implicit surfaces
Geometry Representations

**Point-based surface descriptions**

[A image of a head model](Ohtake, et al., SIGGRAPH 2003)

**Subdivision surface (different levels of refinement)**

[Images from Subdivision.org]
### Geometry Representations

- Various strengths and weaknesses
  - Ease of use for design
  - Ease/speed for rendering
  - Simplicity
  - Smoothness
  - Collision detection
  - Flexibility (in more than one sense)
  - Suitability for simulation
- many others...

### Parametric Representations

- **Curves:** \[ \mathbf{x} = \mathbf{x}(u) \quad \mathbf{x} \in \mathbb{R}^n \quad u \in \mathbb{R} \]
- **Surfaces:** \[ \mathbf{x} = \mathbf{x}(u, v) \quad \mathbf{x} \in \mathbb{R}^n \quad u, v \in \mathbb{R} \]
- **Volumes:** \[ \mathbf{x} = \mathbf{x}(u, v, w) \quad \mathbf{x} \in \mathbb{R}^n \quad u, v, w \in \mathbb{R} \]

and so on...

Note: a vector function is really \( n \) scalar functions
Parametric Rep. Non-unique

- Same curve/surface may have multiple formulae

\[ \mathbf{x}(u) = [u, v] \]
\[ \mathbf{x}(u) = [u^2, v] \]

Simple Differential Geometry

- Tangent to curve
  \[ \mathbf{t}_u = \frac{\partial \mathbf{x}}{\partial u} \]
- Tangents to surface
  \[ \mathbf{t}_{u,v} = \frac{\partial \mathbf{x}}{\partial u,v} \]
  \[ \mathbf{t}_v = \frac{\partial \mathbf{x}}{\partial v} \]
- Normal of surface
  \[ \mathbf{n} = \frac{\mathbf{t}_u \times \mathbf{t}_v}{|\mathbf{t}_u \times \mathbf{t}_v|} \]
- Also: curvature, curve normals, curve bi-normal, others...
- Degeneracies: \( \frac{\partial \mathbf{x}}{\partial u} = 0 \) or \( \mathbf{t}_u \times \mathbf{t}_v = 0 \)
Tangent Space

- The tangent space at a point on a surface is the vector space spanned by:
  \[ \frac{\partial \mathbf{x}(u)}{\partial u} \quad \frac{\partial \mathbf{x}(u)}{\partial v} \]
- Definition assumes that these directional derivatives are linearly independent.
- Tangent space of a surface may exist even if the parameterization is bad.
- For surfaces, the space is a plane.
- Generalized to higher dimension manifolds.

Non Orthogonal Tangents

\[
\begin{bmatrix}
\cos(\theta_2) & \cos(\phi_2)
\end{bmatrix}
\begin{bmatrix}
\sin(\phi_2)
\end{bmatrix}
\]
\[
\begin{bmatrix}
\cos(\theta_3) & \cos(\phi_3)
\end{bmatrix}
\begin{bmatrix}
\sin(\phi_3)
\end{bmatrix}
\]
\[
\begin{bmatrix}
\cos(\theta_4) & \cos(\phi_4)
\end{bmatrix}
\begin{bmatrix}
\sin(\phi_4)
\end{bmatrix}
\]
\[
\begin{bmatrix}
\cos(\theta_5) & \cos(\phi_5)
\end{bmatrix}
\begin{bmatrix}
\sin(\phi_5)
\end{bmatrix}
\]
\[
\begin{bmatrix}
\cos(\theta_6) & \cos(\phi_6)
\end{bmatrix}
\begin{bmatrix}
\sin(\phi_6)
\end{bmatrix}
\]
Discretization

- Arbitrary curves have an uncountable number of parameters

\[ i.e. \text{specify function value at all points on real number line} \]

Discretization

- Arbitrary curves have an uncountable number of parameters
- Pick complete set of basis functions
  - Polynomials, Fourier series, etc.
- Truncate set at some reasonable point

\[ x(u) = \sum_{i=0}^{\frac{3}{2}} c_i \phi_i(u) \]

- Function represented by the vector (list) of \( c_i \)
- The \( c_i \) may themselves be vectors

\[ x(u) = \sum_{i=0}^{1} c_i \phi_i(u) \]
Polynomial Basis

- Power Basis

\[ x(u) = \sum c_i u^i \]

\[ x(u) = C \cdot P^d \]

\[ C = [c_0, c_1, c_2, \ldots, c_d] \]

\[ P^d = [1, u, u^2, \ldots, u^d] \]

The elements of \( P^d \) are \textit{linearly independent}

\( i.e. \) no good approximation

\[ u^e \neq \sum c_i u^i \]

Skipping something would lead to bad results... odd stiffness

---

Specifying a Curve

Given desired values (constraints) how do we determine the coefficients for cubic power basis?

For now, assume

\( t_0 = 0 \quad t_1 = 1 \)

For now assume

\( t_0(w_0) \quad t_1(w_1) \quad x(w_0) \quad x(w_1) \)
Specifying a Curve

Given desired values (constraints) how do we determine the coefficients for cubic power basis?

\[ x(0) = c_0 = x_0 \]
\[ x(1) = c_1 = x_1 \]
\[ x'(0) = c_1 = x'_0 \]
\[ x'(1) = c_1 = x'_1 \]

\[ x(0) = c_0 \]
\[ x(1) = c_1 \]
\[ x'(0) = c_1 \]
\[ x'(1) = c_1 \]
Specifying a Curve

Given desired values (constraints) how do we determine the coefficients for cubic power basis?

\[ c = \beta_u \cdot p \]

\[ x(u) = p^3 \cdot c = \begin{bmatrix} 1 + 0u - 3u^2 + 2u^3 \\ 0 + 8u + 3u^2 - 2u^3 \\ 0 + 1u - 2u^2 + 1u^3 \\ 0 + 0u - 1u^2 + 1u^3 \end{bmatrix} \]

\[ \beta_u = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 1 & 3 \end{bmatrix} \]
Specifying a Curve

Given desired values (constraints) how do we determine the coefficients for cubic power basis?

\[ c = \beta_h \cdot \mathbf{p} \]

\[ x(u) = \begin{bmatrix}
1 + 0u - 3u^2 + 2u^3 \\
0 + 0u + 3u^2 - 2u^3 \\
0 + 1u - 2u^2 + 1u^3 \\
0 + 0u - 1u^2 + 1u^3 \\
\end{bmatrix} \cdot \mathbf{p} \]

\[ x(u) = \sum_{i=0}^{3} p_i b_i(u) \]

Hermite basis functions
Hermite Basis

- Specify curve by
  - Endpoint values
  - Endpoint tangents (derivatives)
- Parameter interval is arbitrary (most times)
  - Don’t need to recompute basis functions
- These are **cubic Hermite**
  - Could do construction for any odd degree
  - \((d - 1)/2\) derivatives at end points

Cubic Bézier

- Similar to Hermite, but specify tangents indirectly

\[
\begin{align*}
  x_0 &= p_0 \\
  x_1 &= p_3 \\
  x'_0 &= 3(p_1 - p_0) \\
  x'_1 &= 3(p_3 - p_2)
\end{align*}
\]

Note: all the control points are points in space, no tangents.
Cubic Bézier

• Similar to Hermite, but specify tangents indirectly

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 \\
0 & 1 & 2 & 3
\end{bmatrix}
\begin{bmatrix}
-3 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & -3 & 1
\end{bmatrix}
\]

\[c = \beta_z p\]

\[x_0 = p_0\]
\[x_1 = 3(p_2 - p_1)\]
\[x_1' = 3(p_3 - p_2)\]

Bézier basis functions

\[c = \beta_z p\]

\[x(u) = p^3 \cdot c\]

\[x(u) = \begin{bmatrix}
1 - 3u + 3u^2 - u^3 \\
0 + 3u - 6u^2 + 3u^3 \\
0 + 0u + 3u^2 - 3u^3 \\
0 + 0u + 0u^2 + 1u^3
\end{bmatrix}\]
Changing Bases

- Power basis, Hermite, and Bézier all are still just cubic polynomials
- The three basis sets all span the same space
- Like different axes in
- Changing basis

\[ c = \beta_z p_z \]
\[ c = \beta_z p_h \]
\[ p_z = \beta_z^{-1} \beta_h p_h \]

Useful Properties of a Basis

- Convex Hull
  - All points on curve inside convex hull of control points
  - Bézier basis has convex hull property

\[ \sum_{i=1}^{n} b_i(u) = 1 \]
\[ b_i(u) \geq 0 \quad \forall u \in \Omega \]
Useful Properties of a Basis

• Invariance under class of transforms
  • Transforming curve is same as transforming control points
  • Bézier basis invariant for affine transforms
  • Bézier basis NOT invariant for perspective transforms
    • NURBS are though...

\[ \mathbf{z}(u) = \sum p_i \mathbf{b}_i(u) \leftrightarrow \mathbf{Tz}(u) = \sum \mathbf{Tp}_i \mathbf{b}_i(u) \]

Useful Properties of a Basis

• Local support
  • Changing one control point has limited impact on entire curve
• Nice subdivision rules
• Orthogonality (\( \int \mathbf{b}_i(u) \mathbf{b}_j(u) du = \delta_{ij} \))
• Fast evaluation scheme
• Interpolation vs. approximation
DeCasteljau Evaluation

- A geometric evaluation scheme for Bézier

Notice tangent line

Opps, error...

Blue line is always tangent to curve.

From Wikipedia
DeCasteljau Evaluation

Blue line is always tangent to curve.

Adaptive Tessellation

- Midpoint test subdivision
- Possible problem
  - Simple solution if curve basis has convex hull property

If curve inside convex hull and the convex hull is nearly flat, curve is nearly flat and can be drawn as straight line.

Better: draw convex hull

Works for Bézier because the ends are interpolated.
Bézier Subdivision

- Form control polygon for half of curve by evaluating at \( u = 0.5 \)
Bézier Subdivision

- Form control polygon for half of curve by evaluating at \( u = 0.5 \)
- Repeated subdivision makes smaller/flatter segments
- Also works for surfaces...
- We'll extend this idea later on...

Joining

If you change \( a, b, \) or \( c \) you must change the others

But if you change \( a, b, \) or \( c \) you do not have to change beyond those three. "Local Support"
"Hump" Functions

- Constraints at joining can be built in to make new basis

Tensor-Product Surfaces

- Surface is a curve swept through space
- Replace control points of curve with other curves

\[
x(u, v) = \sum_i p_i b_i(u)
\]
\[
q_i(u) = \sum_j p_j b_j(u)
\]

\[
x(u, v) = \sum_{ij} p_{ij} b_i(u) b_j(v)
\]

\[
h_i(u) = b_i(u) b_j(v)
\]

\[
x(u, v) = \sum_{ij} p_{ij} b_{ij}(u, v)
\]
<table>
<thead>
<tr>
<th>Tensor-Product Surfaces</th>
<th>40-1</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Diagram" /></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tensor-Product Surfaces</th>
<th>40-2</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image2" alt="Diagram" /></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bézier Surface Patch

- Bézier surface and 4 x 4 array of control points

Adaptive Tessellation

- Given surface patch
- If close to flat, draw it
- Else subdivide 4 ways
### Adaptive Tessellation

- Avoid cracking

<table>
<thead>
<tr>
<th>Passes flatness test</th>
<th>Fails flatness test</th>
</tr>
</thead>
</table>

Crack in the surface

Cracks may be okay in some contexts...
Adaptive Tessellation

- Avoid cracking

Test interior and boundary of patch
Split boundary based on boundary test
Table of polygon patterns
May wish to avoid “slivers”
Adaptive Tessellation

- Triangle Based Method (no cracks)
Adaptive Tessellation

- Triangle Based Method (no cracks)

Center test tends to generate slivers. Often better to leave it out.

Adaptive Tessellation

Without center test

With center test
Adaptive Tessellation

Second row shows typical error of swapping tests.

Yiding Jia, CS184 S08 -- I broke his code to make this example.

Adaptive Tessellation

Visible artifacts from cracks.

Apollo Ellis, CS184 S08
# given the control points of a bezier curve
# and a parametric value, return the curve
# point and derivative
bezcurveinterp(curve, u)
# first, split each of the three segments
# to form two new ones AB and BC
A = curve[0] * (1.0-u) + curve[1] * u
# now, split AB and BC to form a new segment DE
D = A * (1.0-u) + B * u
E = B * (1.0-u) + C * u
# finally, pick the right point on DE,
# this is the point on the curve
p = D * (1.0-u) + E * u
# compute derivative also
dPdu = 3 * (E - D)
return p, dPdu

# given a control patch and (u,v) values, find
# the surface point and normal
bezpatchinterp(patch, u, v)
# build control points for a Bezier curve in v
vcurve[0] = bezcurveinterp(patch[0][0:3], u)
vcurve[1] = bezcurveinterp(patch[1][0:3], u)
vcurve[2] = bezcurveinterp(patch[2][0:3], u)
vcurve[3] = bezcurveinterp(patch[3][0:3], u)
# build control points for a Bezier curve in u
ucurve[0] = bezcurveinterp(patch[0:3][0], v)
ucurve[1] = bezcurveinterp(patch[0:3][1], v)
ucurve[2] = bezcurveinterp(patch[0:3][2], v)
ucurve[3] = bezcurveinterp(patch[0:3][3], v)
# evaluate surface and derivative for u and v
p, dPdv = bezcurveinterp(vcurve, v)
p, dPdu = bezcurveinterp(ucurve, u)
# take cross product of partials to find normal
n = cross(dPdu, dPdv)
# output as is

Beziersurfaces. Smooth Operators.

# given a patch, perform uniform subdivision
subdividepatch(patch, step)
# compute how many subdivisions there
# are for this step size
numdiv = (1 + epsilon) / step
# for each parametric value of u
for (iu = 0 to numdiv)
    u = iu * step
# for each parametric value of v
for (iv = 0 to numdiv)
    v = iv * step
# evaluate surface
p, n = bezpatchinterp(patch, u, v)
savesurfacepointandnormal(p, n)