Triangle meshes (contd.)

CS 4620 Lecture 3
Announcements

• A1 is out
  – Part written: do ALONE
  – Programming: do in pairs, can do alone but fully responsible
• KB: Traveling starting tomorrow (No office hours)
• Wed: Blender lecture by Nic
• Friday: History of graphics (video), flows into 4621 class
• Monday
  – Labor Day!

• See you next Wednesday
Indexed triangle set

- array of vertex positions
  - float[$n_V$][3]: 12 bytes per vertex
    - (3 coordinates x 4 bytes) per vertex
- array of triples of indices (per triangle)
  - int[$n_T$][3]: about 24 bytes per vertex
    - 2 triangles per vertex (on average)
    - (3 indices x 4 bytes) per triangle
- total storage: 36 bytes per vertex (factor of 2 savings)
- represents topology and geometry separately
- finding neighbors is at least well defined
Triangle strips

• Take advantage of the mesh property
  – each triangle is usually adjacent to the previous
  – let every vertex create a triangle by reusing the second and third vertices of the previous triangle
  – every sequence of three vertices produces a triangle (but not in the same order)
  – e.g., 0, 1, 2, 3, 4, 5, 6, 7, … leads to
    (0 1 2), (2 1 3), (2 3 4), (4 3 5), (4 5 6), (6 5 7), …
  – for long strips, this requires about one index per triangle
Triangle strips

<table>
<thead>
<tr>
<th>verts[0]</th>
<th>$x_0, y_0, z_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>verts[1]</td>
<td>$x_1, y_1, z_1$</td>
</tr>
<tr>
<td></td>
<td>$x_2, y_2, z_2$</td>
</tr>
<tr>
<td></td>
<td>$x_3, y_3, z_3$</td>
</tr>
<tr>
<td></td>
<td>$\vdots$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tStrip[0]</th>
<th>4, 0, 1, 2, 5, 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>tStrip[1]</td>
<td>6, 9, 0, 3, 2, 10, 7</td>
</tr>
<tr>
<td></td>
<td>$\vdots$</td>
</tr>
</tbody>
</table>
Triangle strips

• array of vertex positions
  – float[\(n_V\)][3]: 12 bytes per vertex
    • (3 coordinates x 4 bytes) per vertex
• array of index lists
  – int[\(n_S\)][variable]: 2 + \(n\) indices per strip
  – on average, (1 + \(\varepsilon\)) indices per triangle (assuming long strips)
    • 2 triangles per vertex (on average)
    • about 4 bytes per triangle (on average)
• total is 20 bytes per vertex (limiting best case)
  – factor of 3.6 over separate triangles; 1.8 over indexed mesh
Triangle fans

- Same idea as triangle strips, but keep oldest rather than newest
  - every sequence of three vertices produces a triangle
  - e.g., 0, 1, 2, 3, 4, 5, ... leads to (0 1 2), (0 2 3), (0 3 4), (0 4 5), ...
  - for long fans, this requires about one index per triangle
- Memory considerations exactly the same as triangle strip
Example: unit sphere

- position:
  \[ x = \cos \theta \sin \phi \]
  \[ y = \sin \theta \]
  \[ z = \cos \theta \cos \phi \]

- normal is position
  (easy!)
Interpolated normals—2D example

- Approximating circle with increasingly many segments
- Max error in position error drops by factor of 4 at each step
- Max error in normal only drops by factor of 2
How to think about vertex normals

• Piecewise planar approximation converges pretty quickly to the smooth geometry as the number of triangles increases

• But the surface normals don’t converge so well

• Better: store the “real” normal at each vertex, and interpolate to get normals that vary gradually across triangles
Topology vs. geometry

• two completely separate issues:
• mesh topology: how the triangles are connected (ignoring the positions entirely)
• geometry: where the triangles are in 3D space
Topology/geometry examples

• same geometry, different mesh topology:

• same mesh topology, different geometry:
Topological validity

• strongest property: be a manifold
  – this means that no points should be "special"
  – edge points: each edge must have exactly 2 triangles
  – vertex points: each vertex must have one loop of triangles
Topological validity

• Consistent orientation
  – Which side is the “front” or “outside” of the surface and which is the “back” or “inside?”
  – rule: you are on the outside when you see the vertices in counter-clockwise order
  – in mesh, neighboring triangles should agree about which side is the front!
  – caution: not always possible

OK  bad

non-orientable
Texture Mapping

- Cannot model every single change using primitives
- Instead we make the shading parameters (and other properties) vary across the surface
Texture Mapping: applications

• Surprisingly simple idea but with big results
Examples

From Computer Desktop Encyclopedia
Reproduced with permission.
© 2001 Intergraph Computer Systems
Examples of projector functions

• For a sphere: latitude-longitude coordinates
  – maps point to its latitude and longitude
Cylinder