Triangle meshes I

CS 4620 Lecture 2
Shape
spheres

approximate sphere

Andrzej Barabasz

Rineau & Yvinec
CGAL manual
finite element analysis
A small triangle mesh

12 triangles, 8 vertices
A large mesh

10 million triangles from a high-resolution 3D scan
about a trillion triangles from automatically processed satellite and aerial photography
Triangles

• Defined by three vertices
• Lives in the plane containing those vertices
• Vector normal to plane is the triangle’s normal
• Conventions (for this class, not everyone agrees):
  – vertices are counter-clockwise as seen from the “outside” or “front”
  – surface normal points towards the outside (“outward facing normals”)

Triangle meshes

- A bunch of triangles in 3D space that are connected together to form a surface
- Geometrically, a mesh is a piecewise planar surface
  - almost everywhere, it is planar
  - exceptions are at the edges where triangles join
- Often, it’s a piecewise planar approximation of a smooth surface
  - in this case the creases between triangles are artifacts—we don’t want to see them
Representation of triangle meshes

• Compactness
• Efficiency for rendering
  – enumerate all triangles as triples of 3D points
• Efficiency of queries
  – all vertices of a triangle
  – all triangles around a vertex
  – neighboring triangles of a triangle
  – (need depends on application)
    • finding triangle strips
    • computing subdivision surfaces
    • mesh editing
Representations for triangle meshes

• Separate triangles
• Indexed triangle set — shared vertices
• Triangle strips and triangle fans — compression schemes for fast transmission
• Triangle-neighbor data structure — supports adjacency queries
• Winged-edge data structure — supports general polygon meshes

© 2015 Kavita Bala •
(with previous instructor Marschner)
Separate triangles

<table>
<thead>
<tr>
<th>tris[0]</th>
<th>tris[1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>[1]</td>
</tr>
<tr>
<td>$x_0, y_0, z_0$</td>
<td>$x_2, y_2, z_2$</td>
</tr>
<tr>
<td>$x_0, y_0, z_0$</td>
<td>$x_3, y_3, z_3$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

$(x_1, y_1, z_1)$
$(x_0, y_0, z_0)$
$(x_2, y_2, z_2)$
$(x_3, y_3, z_3)$

$T_0$
$T_1$
Separate triangles

• array of triples of points
  – float[n_T][3][3]: about 72 bytes per vertex
    • 2 triangles per vertex (on average)
    • 3 vertices per triangle
    • 3 coordinates per vertex
    • 4 bytes per coordinate (float)

• various problems
  – wastes space (each vertex stored 6 times)
  – cracks due to roundoff
  – difficulty of finding neighbors at all
Indexed triangle set

- Store each vertex once
- Each triangle points to its three vertices

Triangle {
    Vertex vertex[3];
}

Vertex {
    float position[3]; // or other data
}

// ... or ...

Mesh {
    float verts[nv][3]; // vertex positions (or other data)
    int tInd[nt][3]; // vertex indices
}
Indexed triangle set

\[
\begin{array}{c|c}
\text{verts[0]} & x_0, y_0, z_0 \\
\text{verts[1]} & x_1, y_1, z_1 \\
& x_2, y_2, z_2 \\
& x_3, y_3, z_3 \\
\vdots & \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{tInd[0]} & 0, 2, 1 \\
\text{tInd[1]} & 0, 3, 2 \\
\vdots & \\
\end{array}
\]
Estimating storage space

- $n_T = \text{#tris}; n_V = \text{#verts}; n_E = \text{#edges}$

- Euler: $n_V - n_E + n_T = 2$ for a simple closed surface
  - and in general sums to small integer
  - argument for implication that $n_T:n_E:n_V$ is about 2:3:1

[Foley et al.]
- \( n_T = \) #tris; \( n_V = \) #verts; \( n_E = \) #edges

- Euler: \( n_V - n_E + n_T = 2 \) for a simple closed surface
  - and in general sums to small integer
  - argument for implication that \( n_T:n_E:n_V \) is about 2:3:1
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
Data on meshes

• Often need to store additional information besides just the geometry
• Can store additional data at faces, vertices, or edges
• Examples
  – colors stored on faces, for faceted objects
  – information about sharp creases stored at edges
  – any quantity that varies *continuously* (without sudden changes, or *discontinuities*) gets stored at vertices
Key types of vertex data

• Surface normals
  – when a mesh is approximating a curved surface, store normals at vertices

• Texture coordinates
  – 2D coordinates that tell you how to paste images on the surface

• Positions
  – at some level this is just another piece of data
  – position varies continuously between vertices
Differential geometry 101

- **Tangent plane**
  - at a point on a smooth surface in 3D, there is a unique plane tangent to the surface, called the *tangent plane*

- **Normal vector**
  - vector perpendicular to a surface (that is, to the tangent plane)
  - only unique for smooth surfaces (not at corners, edges)
Surface parameterization

- A surface in 3D is a two-dimensional thing
- Sometimes we need 2D coordinates for points on the surface
- Defining these coordinates is *parameterizing* the surface
- Examples:
  - cartesian coordinates on a rectangle (or other planar shape)
  - cylindrical coordinates \((\theta, y)\) on a cylinder
  - latitude and longitude on the Earth’s surface
  - spherical coordinates \((\theta, \phi)\) on a sphere
Example: unit sphere

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

- normal is position (easy!)
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
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

8%, 11°  2%, 6°  0.5%, 3°