Triangle meshes I
CS4620/21 late policy

• **We use slip days**

• **You have 7 slip days for 4620, 7 separate ones for 4621**
  – e.g. you could turn in Ray 1 4 days late and Splines 3 days late. You are out of slip days for further 4620 assignments, but you could still turn in one 4621 assignment 7 days late

• **Accounting is separate per individual**
  – so it’s possible for you to have slip days left but your partner not to

• **Each late day beyond 7 incurs a 10 point late penalty**
  – i.e. project earns 93/100, is 2 days late, receives 73/100

• **Regardless of late penalties, assignments can’t be turned in more than 7 days late**

• **No slip days for 4621 final project**
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-triangle worldwide model from semi-automatically processed satellite, aerial, and street 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 \textit{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

important for first assignment
Separate triangles

<table>
<thead>
<tr>
<th>tris[0]</th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>x₀, y₀, z₀</td>
<td>x₂, y₂, z₂</td>
<td>x₁, y₁, z₁</td>
<td></td>
</tr>
<tr>
<td>x₀, y₀, z₀</td>
<td>x₃, y₃, z₃</td>
<td>x₂, y₂, z₂</td>
<td></td>
</tr>
<tr>
<td>⋮</td>
<td>⋮</td>
<td>⋮</td>
<td></td>
</tr>
</tbody>
</table>

\( (x₀, y₀, z₀) \)\n\( T₀ \)
\( (x₁, y₁, z₁) \)
\( (x₂, y₂, z₂) \)
\( (x₃, y₃, z₃) \)
Separate triangles

- **array of triples of points**
  - float$[nT][3][3]$: 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**

```cpp
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

| verts[0] | x₀, y₀, z₀ |
| verts[1] | x₁, y₁, z₁ |
|          | x₂, y₂, z₂ |
|          | x₃, y₃, z₃ |
|          | ⋮          |

| tInd[0] | 0, 2, 1  |
| tInd[1] | 0, 3, 2  |
|          | ⋮        |
Estimating storage space

- $n_T = \#\text{tris}; \ n_V = \#\text{verts}; \ n_E = \#\text{edges}$
- Rule of thumb: $n_T:n_E:n_V$ 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

- **Surface parameterizations**
  - providing a 2D coordinate system 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
- **Spoiler alert:**
  - in graphics, parameterizations are most often used for texture mapping.
  - therefore many systems call the parameters “texture coordinates.”
Example: unit sphere

- **position:**
  
  \[
  \begin{align*}
  x &= \cos \theta \sin \phi \\
  y &= \sin \theta \\
  z &= \cos \theta \cos \phi
  \end{align*}
  \]

- **normal is position**
  (easy!)

- **texture coordinates**
  
  \[
  \begin{align*}
  u &= \frac{\theta}{\pi} + \frac{1}{2} \\
  v &= \frac{\phi}{2\pi}
  \end{align*}
  \]
How to think about vertex normals

• **Piecewise planar approximation converges pretty quickly to the smooth geometry as the number of triangles increases**
  – for mathematicians: error is $O(h^2)$

• **But the surface normals don’t converge so well**
  – normal is constant over each triangle, with discontinuous jumps across edges
  – for mathematicians: error is only $O(h)$

• **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
Parameterizing a single triangle

- **Triangles**
  - specify \((u,v)\) for each vertex
  - define \((u,v)\) for interior by linear interpolation
Validity of triangle meshes

• in many cases we care about the mesh being able to bound a region of space nicely

• in other cases we want triangle meshes to fulfill assumptions of algorithms that will operate on them (and may fail on malformed input)

• 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"
  - interior points are fine
  - edge points: each edge must have exactly 2 triangles
  - vertex points: each vertex must have one loop of triangles

- **slightly looser: manifold with boundary**
  - weaken rules to allow boundaries
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

\[ \text{OK} \quad \text{bad} \]
Geometric validity

• generally want non-self-intersecting surface
• hard to guarantee in general
  – because far-apart parts of mesh might intersect
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

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

\[
\begin{array}{c|c}
\text{tStrip[0]} & 4, 0, 1, 2, 5, 8 \\
\text{tStrip[1]} & 6, 9, 0, 3, 2, 10, 7 \\
& \vdots \\
\end{array}
\]
Triangle strips

- **array of vertex positions**
  - float\[^nV\][3]: 12 bytes per vertex
    - (3 coordinates \(\times\) 4 bytes) per vertex
- **array of index lists**
  - int\[^nS\][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**