

CS4620/5620: Lecture 18

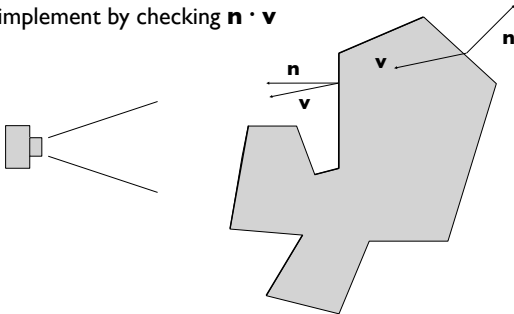
Meshes

Announcements

- Prelim next Monday
 - In class, closed book
 - Including material on Friday
- PPA I out
 - Class on Friday, start early!
- TA evaluations
 - Will receive email online, make sure to fill them out
- 5625, Spring 2012: MW 2:55-4:10

Back face culling

- For closed shapes you will never see the inside
 - therefore only draw surfaces that face the camera
 - implement by checking $\mathbf{n} \cdot \mathbf{v}$



The z buffer

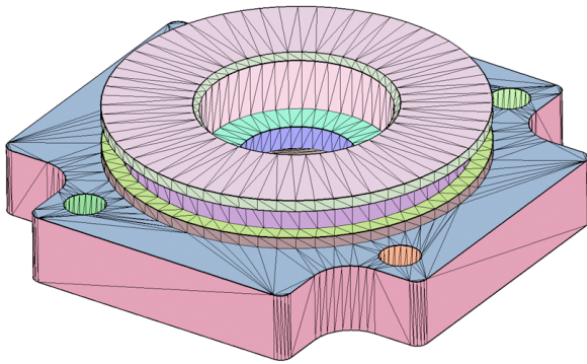
- In many (most) applications maintaining a z sort is too expensive
 - changes all the time as the view changes
 - many data structures exist, but complex
- Solution: draw in any order, keep track of closest
 - allocate extra channel per pixel to keep track of closest depth so far
 - when drawing, compare object's depth to current closest depth and discard if greater

Precision in z buffer

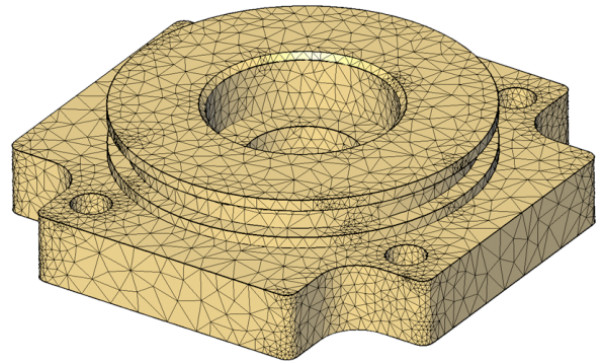
- The precision is distributed between the near and far clipping planes
 - this is why these planes have to exist
 - also why you can't always just set them to very small and very large distances
- Generally use z' (not world z) in z buffer

Polygon Meshes

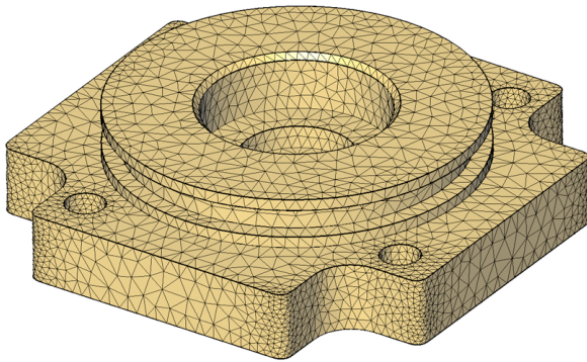
Ch12.1, "Triangle Meshes"



<http://rallyx.inria.fr/2008/Raweb/geometrical/uid15.html>



<http://rallyx.inria.fr/2008/Raweb/geometrical/uid15.html>



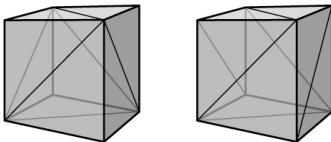
<http://rallyx.inria.fr/2008/Raweb/geometrical/uid15.html>

Aspects of meshes

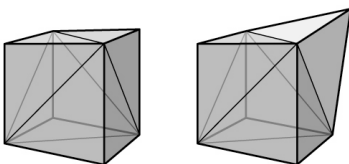
- in many cases we care about the mesh being able to bound a region of space nicely
- in other cases we want triangle meshes to fulfill assumptions of algorithms that will operate on them (and may fail on malformed input)
- two completely separate issues:
 - topology: how the triangles are connected (ignoring the positions entirely)
 - geometry: where the triangles are in 3D space

Topology/geometry examples

- same geometry, different mesh topology:

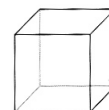


- same mesh topology, different geometry:



Notation

- $n_T = \#tris$; $n_V = \#verts$; $n_E = \#edges$
- Euler: $n_V - n_E + n_T = 2$ for a simple closed surface
 - and in general sums to small integer
 -



$$\begin{matrix} V=8 \\ E=12 \\ F=6 \end{matrix}$$








$$\begin{matrix} V=5 \\ E=6 \\ F=4 \end{matrix}$$



$$\begin{matrix} V=6 \\ E=12 \\ F=8 \end{matrix}$$

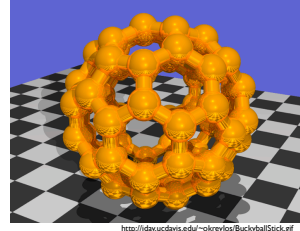


Examples of simple convex polyhedra

Name	Image	Vertices V	Edges E	Faces F	Euler characteristic: $V - E + F$
Tetrahedron		4	6	4	2
Hexahedron or cube		8	12	6	2
Octahedron		6	12	8	2
Dodecahedron		20	30	12	2
Icosahedron		12	30	20	2

http://en.wikipedia.org/wiki/Euler_characteristic

Examples of simple convex polyhedra



<http://dax.ucdavis.edu/~olkyfou/Buckyball/sock.gif>

Buckyball

$$V = 60$$





$$E = 90$$

$$F = 32 \text{ (12 pentagons + 20 hexagons)}$$

$$V - E + F = 60 - 90 + 32 = 2$$



Examples (nonconvex polyhedra!)

Name	Image	Vertices V	Edges E	Faces F	Euler characteristic: $V - E + F$
Tetrahemihexahedron		6	12	7	1
Octahemioctahedron		12	24	12	0
Cubohemioctahedron		12	24	10	-2
Great icosahedron		12	30	20	2

http://en.wikipedia.org/wiki/Euler_characteristic

Euler's Formula

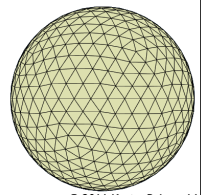
$$n_V = \# \text{verts}; \quad n_E = \# \text{edges}; \quad n_F = \# \text{faces}$$

• Euler's Formula for a convex polyhedron:

$$n_V - n_E + n_F = 2$$

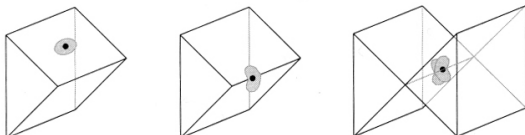
• Other meshes often sum to small integer

– argument for implication that $n_V:n_E:n_F$ is about 1:3:2



Topological validity

- Strongest property, and most simple: be a manifold
 - this means that no points should be "special"
 - interior points are fine
 - edge points: each edge should have exactly 2 triangles
 - vertex points: each vertex should have one loop of triangles
 - not too hard to weaken this to allow boundaries



[Foley et al.]

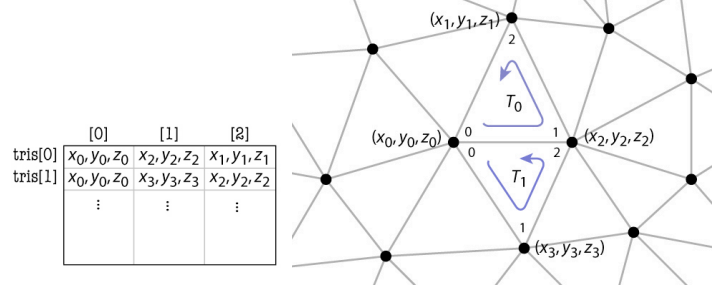
Representation of triangle meshes

- 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 transmission to hardware
- Triangle-neighbor data structure
 - supports adjacency queries
- Winged-edge data structure
 - supports general polygon meshes

Separate triangles



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

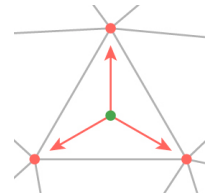
```

Triangle {
    Vertex vertex[3];
}

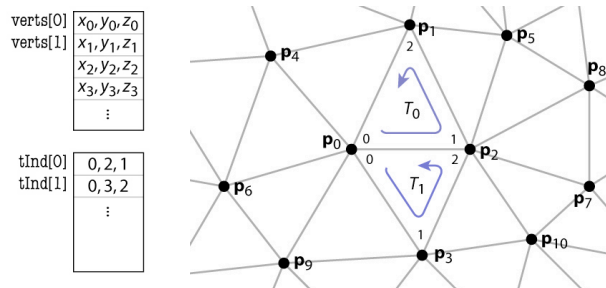
Vertex {
    float position[3]; // or other data
}

// ... or ...

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



Indexed triangle set



Indexed triangle set

- array of vertex positions
 - `float[nV][3]`: 12 bytes per vertex
 - (3 coordinates × 4 bytes) per vertex
- array of triples of indices (per triangle)
 - `int[nT][3]`: about 24 bytes per vertex
 - 2 triangles per vertex (on average)
 - (3 indices × 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

Representations for triangle meshes

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