CptS 442/542 (Computer Graphics)
Unit 5: Meshes

Bob Lewis

School of Engineering and Applied Sciences
Washington State University

Fall, 2017

Motivation

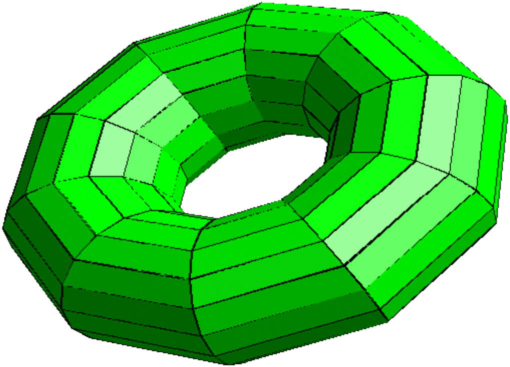
- Meshes are the most common form of model in computer graphics.
- Graphics hardware is optimized for rendering them.
- They are the basis (often via tessellation) of more complicated models (spline surfaces, subdivision surfaces).

How Should We Represent Solid Objects?

Goals:

- wide range of representable objects
- accurate (no approximation required)
- impossible to create a non-solid
- compact
- efficient

Most Common Approach: A Boundary Representation



Boundary Representations

aka, "b-reps"

- object defined by its bounding surfaces
- frequently polygons (i.e. polyhedra), but curved surfaces also possible
- "2-manifolds": every point surrounded by a disk-like region
  - implies "closed" volume
  - no "openings"
- Euler's Formula (for polyhedra):

$$V - E + F = 2$$

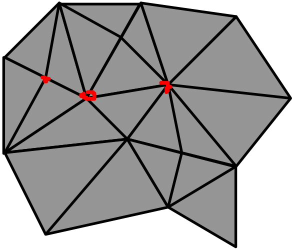
with V vertices, E edges, and F faces.

Polygonal Meshes

- made of
  - vertices (of abitrary dimensionality)
  - edges connect vertices
  - faces (usually triangles or quads) bound by edges (and vertices)
- normals (later) required for
  - lighting
  - collision detection
  - rendering efficiency

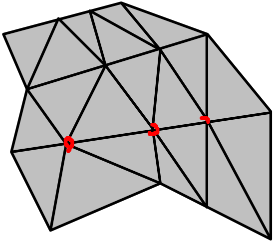
Irregular vs. Regular Meshes

Irregular Mesh



The number of edges sharing a vertex varies. This is the most general. We'll use it now.

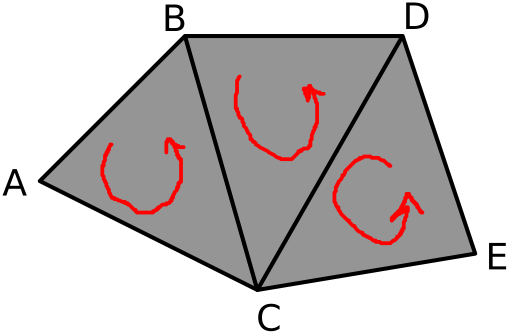
Regular Mesh



The number of edges sharing a(n interior) vertex is the same. Less general, but can be more efficient to render. We'll use it later.

Representing Polygonal Meshes I

Suppose we have a mesh like this?



How might we represent it?

Representing Polygonal Meshes III

There are several ways to store polygon (let's say triangular) meshes to represent surfaces. We could put the coordinates in a table:

glDrawArrays()

x	y	z
x_A	y_A	z_A
x_C	y_C	z_C
x_B	y_B	z_B
x_B	y_B	z_B
x_C	y_C	z_C
x_D	y_D	z_D
x_D	y_D	z_D
x_C	y_C	z_C
x_E	y_E	z_E

Every three vertices defines a triangle.

This is how we'll initially specify irregular meshes in *coaster*.

But this is inefficient: Look how many times the vertices of C are included. Every vertex needs to go through the vertex shader.

RESUME

Representing Polygonal Meshes IV

There is much truth in this:

"All problems in computer science can be solved by another level of indirection." – David Wheeler, 1st Ph.D. in CS (1951)

Case in point:

vertex	x	y	z
A	x_A	y_A	z_A
B	x_B	y_B	z_B
C	x_C	y_C	z_C
D	x_D	y_D	z_D
E	x_E	y_E	z_E

face	vertex 1	vertex 2	vertex 3
1	A	C	B
2	B	C	D
3	D	C	E

glDrawElements()

The OBJ Format

- common format for mesh (and other graphics) data
- others include PLY and STL
- *n*-dimensional (*n* is usually 3)
- first developed for Wavefront Technologies, which was bought by Silicon Graphics, which merged it with Alias to form Alias|Wavefront, which is now part of Autodesk
- ASCII-based (easy to read and write in code)
- viewable with `meshlab(1)` (open source, recommended)
- We're using a small subset.
- See `demos` contents and `car.obj`.
- acceptable to 3D printers

Polyhedra

- Platonic solids:

Name	V	F	E	Schlafli Symbol (p, q)
tetrahedron	4	4	6	(3,3)
hexahedron	8	6	12	(4,3)
octahedron	6	8	12	(3,4)
icosahedron	12	20	30	(3,5)
dodecahedron	20	12	30	(5,3)

- Schlafli: Each face is a p-gon and each vertex joins q faces.
- test Euler's Formula: $V - E + F =?$ 2
- these are all representable by regular meshes
- note duality

Adding Attributes to Mesh Entities

We can bind any attributes to each mesh entity

- vertex
  - position (initially)
  - color
  - normal
  - texture coordinates

  GLSL makes it easy to assign these and any other properties to vertices to use in a vertex shader.
- face
  - vertices (initially)
  - normal

  We would assign these in the geometry shader.

We can, but don't, do anything with edges in `coaster`, but see below.

Navigational Queries

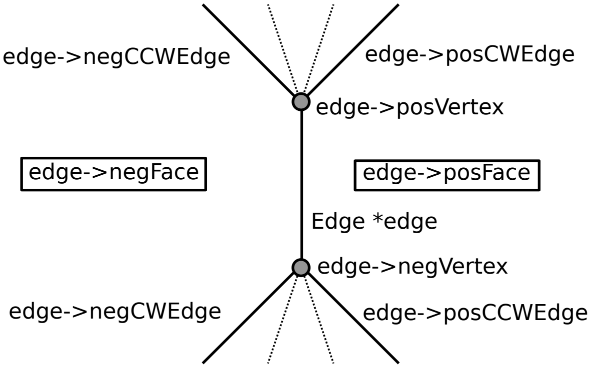
The design of the mesh, like any database, should be based on the kinds of queries most likely to be asked of it.
These can be hard to answer with the `mesh.h` we're using for *coaster*:

- If I split this edge with a new vertex, what faces need to be modified?
- If I remove this vertex, what faces need to be re-tesselated?
- Which faces intersect this (cutting) plane?

So we need to enhance the data structure. One way to do this is...

Beyond *coaster*: Winged Edges

Everything is based on a (new) `Edge` structure with these fields:



We use pointers here – they're traditional – but array indices (including a new "edge" array) would work as well.