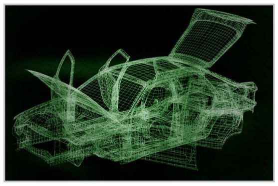


Meshes
Modeling Objects

CS148: Intro to CG
Instructor: Dan Morris
TA: Sean Walker
July 12, 2005

Administrative blah-blah

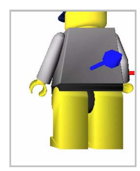
- pp1 was (virtually) handed back
- Exam is coming up
  - What do you need to know?
  - Submit questions!
- Submit video-break videos
- Only one late day on pp4

Outline for today

- Face culling
- Representing meshes
- Representing surfaces
- Drawing surfaces

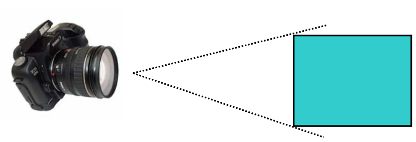
Being stingy with our triangles

- When we draw Lego Man, we might draw lots of triangles that end up getting covered up
- It's not useful to draw the triangles on the other side of Lego Man



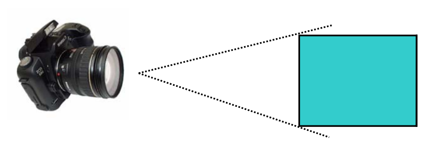
Backface culling

- For any *closed* object, it's not useful to draw any triangles that *face away from the viewer*
- Often want to eliminate backwards triangles (that face away from the viewer) before rasterization
- This is 'backface culling', and OpenGL can do it for you



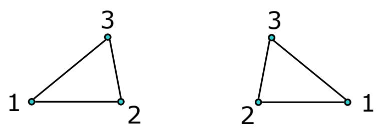
Which way does a triangle "face"?

- Intuitively, we want the side of the triangle on the outside of our object to be the front
- When I draw a triangle, OpenGL doesn't know about inside and outside
- Need a way to specify front and back of a triangle



Which way does a triangle "face"?

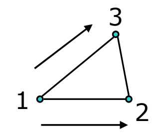
- The order of vertices tells OpenGL which way a triangle faces
- If I look at a triangle, the vertices appear clockwise on one side, counter-clockwise on the other
- In GL, the side ordered *CCW* is the front



Face normals

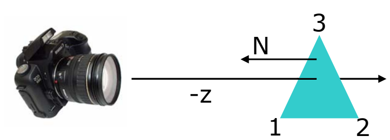
- Normal to a face: vector perpendicular to the face, pointing toward the *front*

How do we compute the normal to this triangle?



Which triangles "face the viewer"?

In *eye coordinates*, how do I decide if a triangle faces the viewer (plain English answer)?

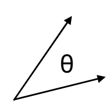


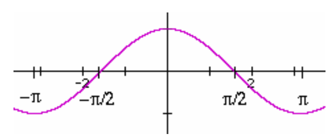
Reminder: dot products

$A \bullet B = |A||B|\cos\theta$

$A \bullet B = 0 \rightarrow$ vectors are perpendicular

$A \bullet B > 0 \rightarrow$ vectors "mostly" point the same way





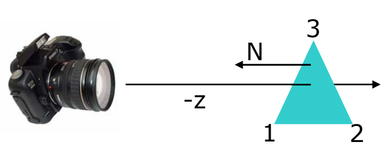
Which triangles "face the viewer"?

$N \bullet (\mathbf{z}) > 0 \rightarrow$ triangle faces camera

$N \bullet (0,0,1) > 0$

$(N_x*0) + (N_y*0) + (N_z*1) > 0$

$N_z > 0$



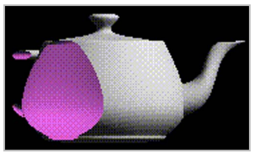
Culling in OpenGL [culling.cpp]

- Default state listed first:
  - glDisable(GL_CULL_FACE);
    glEnable(GL_CULL_FACE);
  - glCullFace(GL_BACK);
    glCullFace(GL_FRONT);
  - glFrontFace(GL_CCW);
    glFrontFace(GL_CW);

When would I want culling disabled?

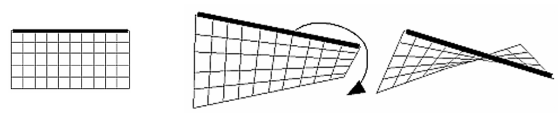
Front-back in OpenGL

- Frontface/backface determination is also used for lighting
- Backfaces can be a different color / brightness / etc.



Illegal Polygons

- Can’t define a normal for a non-planar polygon
- Illegal to send OpenGL non-planar polygons
- Even simple polygons can become non-planar after transformations
- Polygons are generally *tessellated* by OpenGL

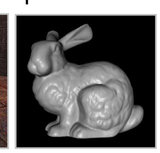


Outline for today

- Face culling
- Representing meshes
- Representing surfaces
- Drawing surfaces

Representing models

- Models so far were made of triangles or simple primitives that we hard-coded in our programs
- “Real” models have too many polygons to create manually
- Need a way to store and represent objects

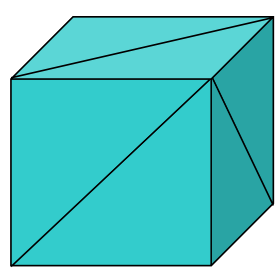


A mesh data structure: take one

```
struct vertex {
   float x,y,z;
}
struct triangle {
   vertex a,b,c;
}
struct object {
   unsigned int nTriangles;
   triangle* triangles;
}
```

What’s wrong with this approach?

How many vertices in a cube?



So what?

What are some disadvantages of having redundant vertices?

A better mesh data structure

```
struct vertex {
   float x,y,z;
}
struct triangle {
   unsigned int a,b,c;
}
struct object {
   unsigned int nVertices;
   vertex* vertices;
   unsigned int nTriangles;
   triangle* triangles;
}
```

Meshes in OpenGL [meshes.cpp]

- Data structures like this are very common
- In fact, a very common way of sending meshes to OpenGL is:

```
// Tell GL where to find vertices
glVertexPointer(3,GL_FLOAT,0,
   myObject.vertices);

// Draw "indexed triangles"
glDrawElements(GL_TRIANGLES,
   myObject.nTriangles,GL_UNSIGNED_INT,
   myObject.triangles);
```

Meshes in OpenGL

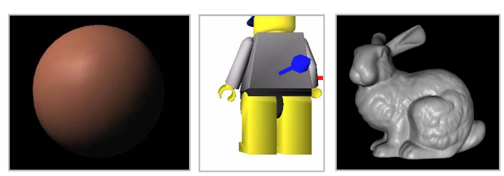
- Most mesh file formats store vertices and triangles in a format more or less like this
  - Popular formats: .3ds, .obj, .stl
- Most formats add a little more data to each vertex:
  - Surface normals
  - Color information
  - Texture coordinates
  - These topics make up the next few classes in CS148

Outline for today

- Face culling
- Representing meshes
- Representing surfaces
- Drawing surfaces

When aren't meshes quite right?

Which of these objects could be represented more efficiently with another approach?



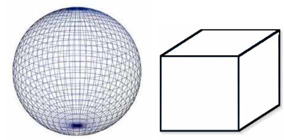
Why not always store our objects as meshes?

- Space
- Level of detail
- Modeling
- Non-polygon renderers

- OpenGL (mostly) only knows about triangles and vertices, so we're leaving the GL universe for a bit...

Another approach: object primitives

- Spheres
- Cubes
- Other objects I can easily parameterize



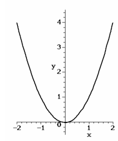
A file format with object primitives

```
Object LegoMan {
   // numbers are arbitrary
   sphere head(0.2,0.3,0.4,0.1);
   cube body(0.2,0.5,0.3,0.2);
   triMesh hand {
        vertices 0.4,0.3,0.5,0.2...
        triangles 1,2,3,2,3,4...
   }
}
```

- In fact, VRML – a popular model format – is not all that far from this

Still many objects we can't compactly represent...

- In particular, we'd like to be able to efficiently represent curves in space
- ...just you can efficiently represent a parabola by writing $y = x^2$, without writing down 100000 points on the parabola.



Parametric Surfaces

- Think of an object's "skin" as a 2D surface
- Use two variables (u,v) to tell me where I am on the surface
- A position function p(u,v) *generates* points in space

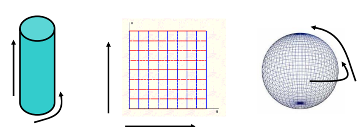
$p(u,v) = (X(u,v),\ Y(u,v),\ Z(u,v))$

- In vector notation:

$\mathbf{p}(u,v) = X(u,v)\mathbf{i} + Y(u,v)\mathbf{j} + Z(u,v)\mathbf{k}$

Parametric Surfaces

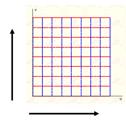
- u and v mean different things for different objects
- A parametric surface gives back different points for different (u,v) values
- A cylinder might interpret (u,v) as (h,θ)
- A plane might interpret (u,v) as (x,y)
- **What might (u,v) represent for a sphere?**



Defining a Parametric Plane

- I can specify a plane with a point **c** and any two vectors **a,b** that live on the plane
- How can I put this in parametric form?
- Define (u,v) = (0,0) as the point **c**
- Can represent any vector in 2D as a combination of two non-collinear vectors
- So our whole plane is:

p(u,v) = u**a** + v**b**

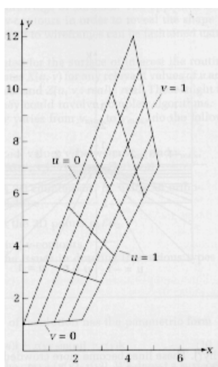


Planar Patches

- Could represent the whole plane by letting u and v range from $-\infty$ to $+\infty$
- Often we want to represent a specific chunk of a plane, e.g. the front of Lego Man
- We often define **a** and **b** so that letting u and v go from 0 to 1 will give us the object we want
- This restricted piece of a parametric surface is called a *patch*

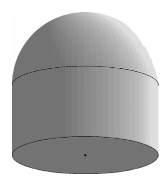
Planar Patches

u	v	corner
0	0	c
0	1	c+b
1	0	c+a
1	1	c+a+b



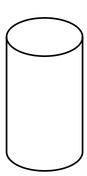
Other Patches

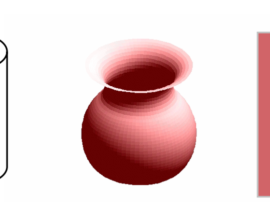
- A sphere might interpret (u,v) as (θ,ρ)
- A cylinder might interpret (u,v) as (h,θ)
- This might be represented as a spherical patch (restricted range of (θ,ρ)) and a cylindrical patch (restricted range of (h,θ))

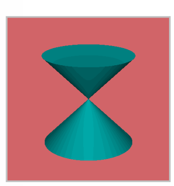


Other surface types

- Ruled surfaces
- Surfaces of revolution
- Quadric surfaces







Aside: parametric Line Segment

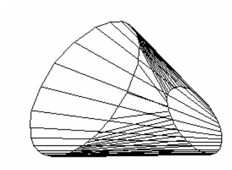
- If we want a line segment from p0 to p1, we might parameterize it as:

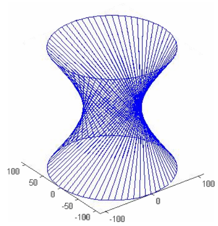
p(v) = (1 - v)**p**0 + v**p**1

...only need one parameter.

Ruled Surfaces

- A "ruled surface" is made of up many straight lines
- Or one straight line moving and rotating through space





Ruled Surfaces

- How do we parameterize a ruled surface?

$\mathbf{p}(u,v) = (1 - v)\mathbf{p}0(u) + v\mathbf{p}1(u)$

- Looks similar to the line equation, but has an extra parameter...

What do u and v represent?

Ruled Surfaces: Example 1

What's unique about this ruled surface?

$\mathbf{p}(u,v) = \mathbf{p}0(u) + v\mathbf{d}$

What surface does this define if p0(u) = (R cos(u), R sin(u), 0)?

Ruled Surfaces: Example 2

What's unique about this ruled surface?

$\mathbf{p}(u,v) = (1-v)\mathbf{p}0 + v\mathbf{p}1(u)$

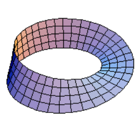
What surface does this define if p1(u) = (R cos(u), R sin(u), 0)?

Ruled Surfaces: Example 3

What ruled surface does this represent?

p =
```
cos(u) + v*cos(u/2) * cos(u),
sin(u)+v*cos(u/2)*sin(u),
sin(u/2);
```

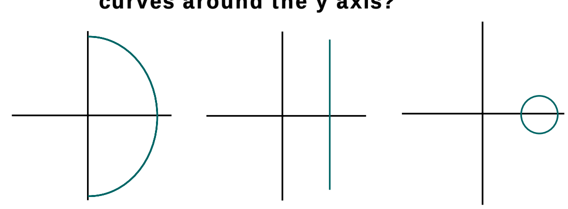
I'm kidding. Don't try to figure it out.



Surfaces of Revolution

- A "surface of revolution" results from sweeping a planar curve around an axis

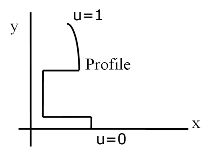
What do I get by sweeping the following curves around the y axis?



Parameterizing surfaces of revolution

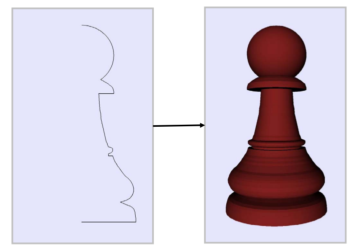
- The curve itself – the "profile" of the surface – is a 1-D parameterized curve x(u),y(u)
- The second parameter v swings this curve around an axis

$\mathbf{p}(u,v) = [\ x(u)\cos(v)\ ,\ y(u)\ ,\ -x(u)\sin(v)\ ]$



Are surfaces of revolution useful?

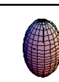
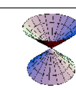
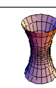
- Very popular in modeling packages for going from 2D --> 3D



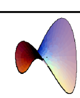
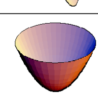
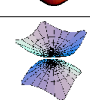
Quadrics

- Surfaces defined by an algebraic equation of degree 2
- It turns out *any* quadric can be transformed into a very small handful of surfaces, so knowing how to draw a small set of surfaces could let us represent a lot of objects

Quadrics

Ellipsoid	$\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}=1$	
Elliptic cone	$\frac{x^2}{a^2}+\frac{y^2}{b^2}-\frac{z^2}{c^2}=0$	
Hyperboloid of one sheet	$\frac{x^2}{a^2}+\frac{y^2}{b^2}-\frac{z^2}{c^2}=1$	

Quadrics

Hypberbolic paraboloid	$\frac{y^2}{b^2}-\frac{x^2}{a^2}=\frac{z}{c}$	
Elliptic paraboloid	$\frac{x^2}{a^2}+\frac{y^2}{b^2}=\frac{z}{c}$	
Hyperboloid of two sheets	$\frac{z^2}{c^2}-\frac{x^2}{a^2}-\frac{y^2}{b^2}=1$	

Outline for today

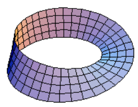
- Face culling
- Representing meshes
- Representing surfaces
- Drawing surfaces

Drawing parametric surfaces

- Imagine you have a black box that spits out points p(u,v), and you want to draw the surface for the range u=0:1, v=0:1
- Note that points close to each other in (u,v) should be close to each other on the surface

How would we draw a parametric surface as quads?

What's (slightly) wrong with this approach?



Drawing parametric surfaces

- Often need surface normals to draw things nicely in OpenGL (details coming soon)

How would we compute the normal to a parametric surface at (u,v)?

$\mathbf{n}(u0, v0) = ((\partial \mathbf{p}/\partial u) \times (\partial \mathbf{p}/\partial v))$

Drawing surfaces of revolution

- Let's say we wanted a wireframe model of a surface of revolution
- Can draw this as "parallels" (circles in the xy-plane) and "meridians" (copies of the profile, rotated around the y axis)

- For fun at home: sketch an OpenGL routine to do this given (x,y) = profile(u)

Drawing quadrics

- GL has support for some basic quadrics

```
GLUquadricObj* qobj = gluNewQuadric();
gluQuadricDrawStyle(qobj, GLU_FILL);
gluSphere(qobj, 1.0, 10, 7);
```

Next time

- Color
- Lighting and shading