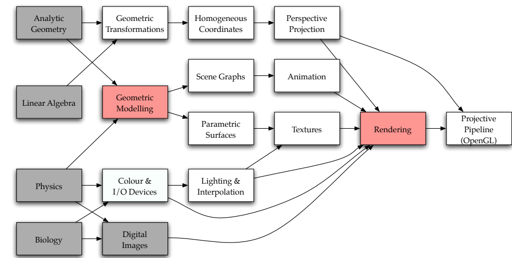
Geometry in 3 Dimensions

COMP 3003
Autumn 2005

Where We Are



Rendering

- *Rendering* converts *maths* to *images*
  - we can choose how we do it
  - *two* basic methods:
    - *vector* graphics - abstract description
    - *raster* graphics - concrete image

Vector vs. Raster

- Vector is better:
  - when you need a *high-level* description
- Raster is better:
  - if you are designing hardware
- Geometric modelling is *vector* graphics
  - OpenGL converts it to *raster* graphics

Vector Graphics

- Are a set of instructions for drawing:
  - Move 1.0 mm left, 2.0 mm down
  - Take a red HB pencil 1 mm wide at tip
  - Draw at angle of 72° for 3 mm
- Example: Postscript, PDF, Illustrator
  - but printer is usually *raster* anyway

Raster Graphics

- Treat image as a large 2-D array of *pixels*
- Almost ubiquitous now (why?)
- This is what we have used so far
  - But it takes a *lot* of pixels
  - Less *compact* than vector graphics
- And it's harder to change the image

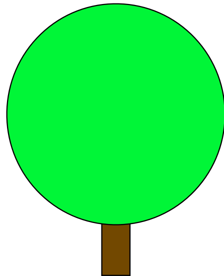
Geometric Modelling

- We need to describe the world
- Most important:
  - where an object is
  - what it is
  - how it behaves

A tree

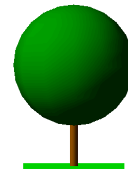
How can we describe this object?

Drawing a Tree



Draw a circle on top of a rectangle

Drawing a 3D Tree



Draw a sphere on top of a cylinder

Complex Geometry

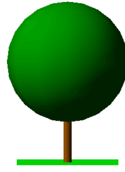
- This isn't a very good tree
- It's too simple geometrically
- A real tree has ?100,000? leaves
- But for now, let's pretend it's OK

Geometric Description

- How to describe a large complex object:
  - break into smaller *primitives*
    - spheres, cylinders, boxes
    - polygons, polyhedra, prisms
  - render each primitive separately

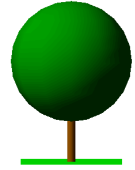
Description

- Draw a sphere
  - radius 10 m
  - centred 7 m above ground
  - colour light green
- Draw a cylinder
  - radius 2 m, height 15 m
  - bottom face on ground
  - colour brown

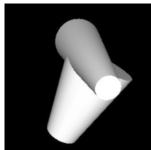


CSG: Constructive Solid Geometry

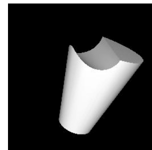
- Tree is built of smaller objects
- Formally, the *union* of them
- Can also do things like:
  - *subtract* objects
  - *xor* objects
  - But not with OpenGL



CSG Example



Union



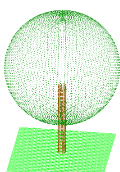
Subtraction

http://www.cs.unc.edu/~geom/CSG/boole.html

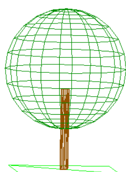
Rendering Methods

- We don't have to render *realistically*
- Sometimes better to retain abstraction
  - *point clouds*
  - *wireframe*
  - *flat shaded surfaces*
  - *smooth shaded surfaces*

Styles of Rendering

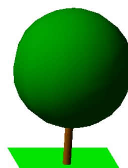


Points



Wireframe (Lines)

Faceted (Flat)



Smooth

Building Objects

- Objects are *built up* from primitives
  - points, lines, triangles
- We will build the following:
  - *Platonic* solids
  - *Solids* of revolution

OpenGL commands

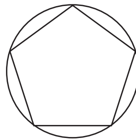
- glVertex3f(x,y,z);
  - specifies a vertex with 3 floats
- glBegin(GL_LINES);
  - starts drawing lines (2 points each)
- glEnd();
  - ends a group of vertices

Polygons

- A *polygon* is a 2-D shape
  - triangle (3 sides)
  - square / rectangle / quadrilateral (4)
  - pentagon (5)
  - hexagon (6)
  - circle (∞)

Regular Pentagon

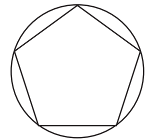
- 5 vertices on a circle
  - spaced $360/5 = 72°$ apart
- Edges (lines) between them



```
glBegin(GL_LINES);
for (vertex = 0; vertex < 5; vertex++)
  { // vertex loop
  theta1 = vertex * 72 / 360 * 2 * PI; // in radians
  theta2 = (vertex + 1) * 72 / 360 * 2 * PI;
  glVertex3f(sin(theta1), cos(theta1),0.0);
  glVertex3f(sin(theta2), cos(theta2),0.0);
  } // vertex loop
glEnd();
```

Regular *n*-gon

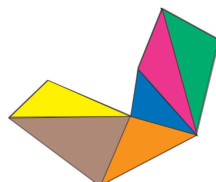
- *n* vertices on a circle
  - spaced $360/n = 72°$ apart
- Edges (lines) between them



```
glBegin(GL_LINES);
for (vertex = 0; vertex < n; vertex++)
  { // vertex loop
  theta1 = vertex * 2 * PI / n; // in radians
  theta2 = (vertex + 1) * 2 * PI / n;
  glVertex3f(sin(theta1), cos(theta1),0.0);
  glVertex3f(sin(theta2), cos(theta2),0.0);
  } // vertex loop
glEnd();
```

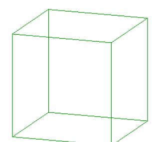
Filling Polygons

- Last week we *rasterized* triangles
  - i.e. we drew *filled* triangles
- Any polygon can be turned into triangles
  - by cutting vertices off



Polyhedra

- A *polyhedron* is a 3-D shape
  - of *vertices*
  - connected with *edges*
- Can be rendered as lines
  - this is called *wireframe*

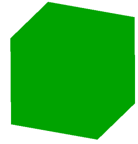


Example: A Cube

```
glBegin(GL_LINES);
glVertex3f(-1.0, -1.0, -1.0); glVertex3f(-1.0, -1.0,  1.0);
glVertex3f(-1.0, -1.0,  1.0); glVertex3f(-1.0,  1.0,  1.0);
glVertex3f(-1.0,  1.0,  1.0); glVertex3f(-1.0,  1.0, -1.0);
glVertex3f(-1.0,  1.0, -1.0); glVertex3f(-1.0, -1.0, -1.0);

glVertex3f(-1.0, -1.0, -1.0); glVertex3f( 1.0, -1.0, -1.0);
glVertex3f(-1.0, -1.0,  1.0); glVertex3f( 1.0, -1.0,  1.0);
glVertex3f(-1.0,  1.0,  1.0); glVertex3f( 1.0,  1.0,  1.0);
glVertex3f(-1.0,  1.0, -1.0); glVertex3f( 1.0,  1.0, -1.0);

glVertex3f( 1.0, -1.0, -1.0); glVertex3f( 1.0, -1.0,  1.0);
glVertex3f( 1.0, -1.0,  1.0); glVertex3f( 1.0,  1.0,  1.0);
glVertex3f( 1.0,  1.0,  1.0); glVertex3f( 1.0,  1.0, -1.0);
glVertex3f( 1.0,  1.0, -1.0); glVertex3f( 1.0, -1.0, -1.0);
glEnd();
```

Solid Polyhedra

- Polyhedra also have *surfaces*
  - each *face* is a polygon
  - broken into triangles
  - specified in CCW order
  - from the outside



Example: A Cube

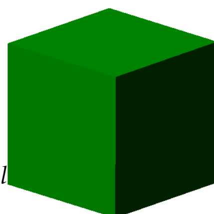
```
GLint vertices[8][3] =
  {-1, -1, -1,   -1, -1,  1,   -1,  1, -1,   -1,  1,  1,
    1, -1, -1,    1, -1,  1,    1,  1, -1,    1,  1,  1};

glBegin(GL_TRIANGLES);

glVertex3iv(vertices[0]); glVertex3iv(vertices[1]); glVertex3iv(vertices[3]);
glVertex3iv(vertices[0]); glVertex3iv(vertices[3]); glVertex3iv(vertices[2]);

glVertex3iv(vertices[0]); glVertex3iv(vertices[4]); glVertex3iv(vertices[5]);
glVertex3iv(vertices[0]); glVertex3iv(vertices[5]); glVertex3iv(vertices[4]);

glVertex3iv(vertices[0]); glVertex3iv(vertices[2]); glVertex3iv(vertices[6]);
glVertex3iv(vertices[0]); glVertex3iv(vertices[6]); glVertex3iv(vertices[4]);

glVertex3iv(vertices[7]); glVertex3iv(vertices[4]); glVertex3iv(vertices[6]);
glVertex3iv(vertices[0]); glVertex3iv(vertices[5]); glVertex3iv(vertices[4]);

glVertex3iv(vertices[0]); glVertex3iv(vertices[2]); glVertex3iv(vertices[3]);
glVertex3iv(vertices[0]); glVertex3iv(vertices[6]); glVertex3iv(vertices[2]);

glVertex3iv(vertices[0]); glVertex3iv(vertices[1]); glVertex3iv(vertices[5]);
glVertex3iv(vertices[0]); glVertex3iv(vertices[3]); glVertex3iv(vertices[1]);
glEnd();
```

Lighting & Shading

- We perceive 3-D objects using *shading*
  - cues to the visual system
  - based on amount of light reflected
  - this depends on the *angle* of reflection
  - which depends on the *surface normal*

Shaded Polyhedra

- Each *face* lies on a plane
  - we can compute a *normal*
  - for reflecting light
- Specify normal with *glNormal*
  - *before* specifying vertices



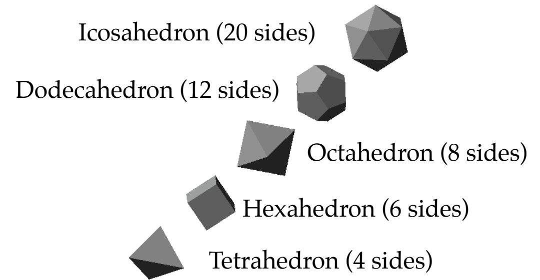
Example: A Cube

```
GLint vertices[8][3] =
  {-1, -1, -1,   -1, -1,  1,   -1,  1, -1,   -1,  1,  1,
    1, -1, -1,    1, -1,  1,    1,  1, -1,    1,  1,  1};
glBegin(GL_TRIANGLES);
glNormal3i(-1, 0, 0);//face 2
glVertex3iv(vertices[0]);glVertex3iv(vertices[2]);glVertex3iv(vertices[3]);
glVertex3iv(vertices[0]);glVertex3iv(vertices[3]);glVertex3iv(vertices[1]);
glNormal3i( 0, -1, 0);//face0
glVertex3iv(vertices[0]);glVertex3iv(vertices[1]);glVertex3iv(vertices[5]);
glVertex3iv(vertices[0]);glVertex3iv(vertices[5]);glVertex3iv(vertices[4]);
glNormal3i( 0, 0, -1);//face1
glVertex3iv(vertices[0]);glVertex3iv(vertices[4]);glVertex3iv(vertices[6]);
glVertex3iv(vertices[0]);glVertex3iv(vertices[6]);glVertex3iv(vertices[2]);
glNormal3i( 1, 0, 0);////face3
glVertex3iv(vertices[6]);glVertex3iv(vertices[5]);glVertex3iv(vertices[4]);
glVertex3iv(vertices[6]);glVertex3iv(vertices[5]);glVertex3iv(vertices[7]);
glNormal3i( 0, 1, 0);//face5
glVertex3iv(vertices[6]);glVertex3iv(vertices[3]);glVertex3iv(vertices[7]);
glVertex3iv(vertices[6]);glVertex3iv(vertices[3]);glVertex3iv(vertices[2]);
glNormal3i( 0, 0, 1);//face4
glVertex3iv(vertices[5]);glVertex3iv(vertices[3]);glVertex3iv(vertices[1]);
glVertex3iv(vertices[5]);glVertex3iv(vertices[3]);glVertex3iv(vertices[7]);
glEnd();
```

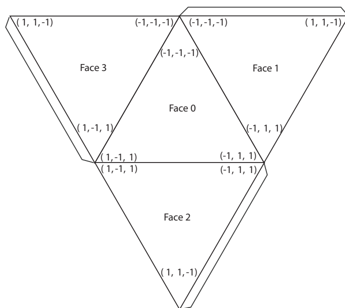
Platonic Solids

- Regular *convex* polyhedra
  - All faces are same size and shape
  - Each face is a regular polygon
  - Each edge is same length
- Convex = no indentations

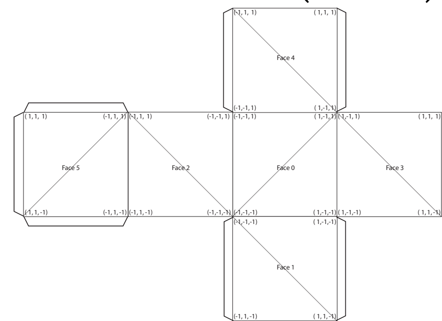
5 Platonic Solids



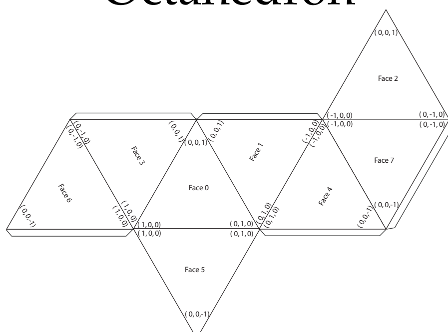
Tetrahedron



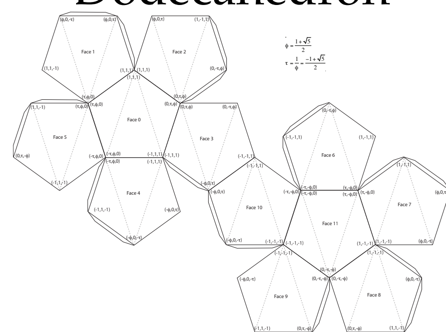
Hexahedron (Cube)



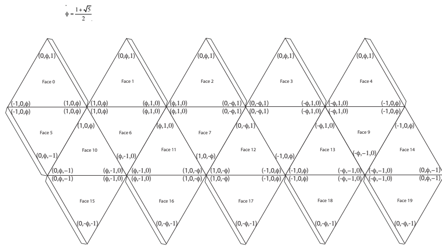
Octahedron



Dodecahedron



Icosahedron

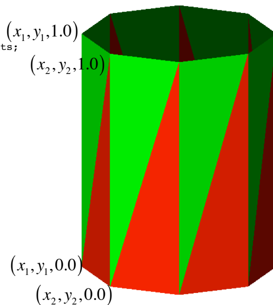


Round Objects

- We *approximate* round objects
  - with polyhedra
  - possibly with *lots* of faces
- How can we approximate a cylinder?
  - as an *extruded* polygon
  - turn each edge into a vertical strip

A Vertical Cylinder

```
for (float i = 0.0; i < nSegments; i += 1.0)
  { /* a loop around circumference of a tube */
  float angle = PI * i * 2.0 / nSegments ;
  float nextAngle = PI * (i + 1.0) * 2.0 / nSegments;

  /* compute sin & cosine */
  float x1 = sin(angle), y1 = cos(angle);
  float x2 = sin(angle), y2 = cos(angle);

  /* draw top (green) triangle */
  glVertex3f(x1, y1, 0.0);
  glVertex3f(x2, y2, 1.0);
  glVertex3f(x1, y1, 1.0);

  /* draw bottom (red) triangle */
  glVertex3f(x1, y1, 0.0);
  glVertex3f(x2, y2, 0.0);
  glVertex3f(x2, y2, 1.0);

  } /* a loop around circumference of a tube */
```

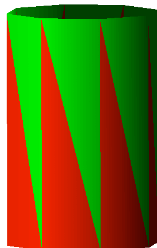


Smoother Cylinders

- How can we make the cylinder *smoother?*
  - provide normals for *each* vertex
  - based on the *tangent plane*
- Cylinder is vertical, so normal is horizontal
  - sticking straight out through each vertex

A Smoother Cylinder

```
for (float i = 0.0; i < nSegments; i += 1.0)
  { /* a loop around circumference of a tube */
  float angle = PI * i * 2.0 / nSegments ;
  float nextAngle = PI * (i + 1.0) * 2.0 / nSegments;

  /* compute sin & cosine */
  float x1 = sin(angle), y1 = cos(angle);
  float x2 = sin(angle), y2 = cos(angle);

  /* draw top (green) triangle */
  glNormal3f(x1, y1, 0.0); glVertex3f(x1, y1, 0.0);
  glNormal3f(x2, y2, 0.0); glVertex3f(x2, y2, 1.0);
  glNormal3f(x1, y1, 0.0); glVertex3f(x1, y1, 1.0);

  /* draw bottom (red) triangle */
  glNormal3f(x1, y1, 0.0); glVertex3f(x1, y1, 0.0);
  glNormal3f(x2, y2, 0.0); glVertex3f(x2, y2, 0.0);
  glNormal3f(x2, y2, 0.0); glVertex3f(x2, y2, 1.0);

  } /* a loop around circumference of a tube */
```



Some Round Objects