OpenGL Tutorial

By Jason Lawrence

OpenGL **IS** an API

- OpenGL **IS** nothing more than a set of functions you call from your program (think of as collection of .h file(s)).
- Hides the details of the display adapter, operating system, etc.
- Comprises several libraries with varying levels of abstraction: GL, GLU, and GLUT

OpenGL Hierarchy

- Several levels of abstraction are provided
- GL
  - Lowest level: vertex, matrix manipulation
  - glVertex3f(point.x, point.y, point.z)
- GLU
  - Helper functions for shapes, transformations
  - gluPerspective(fovy, aspect, near, far)
- GLUT
  - Highest level: Window and interface management
  - glutSwapBuffers()

OpenGL Implementations

- OpenGL IS an API (think of as collection of .h files):
  - #include <GL/gl.h>
  - #include <GL/glu.h>
  - #include <GL/glut.h>
- Windows, Linux, UNIX, etc. all provide a **platform specific** implementation.
- Windows: opengl32.lib glu32.lib glut32.lib
- Linux: -l GL -l GLU –l GLUT

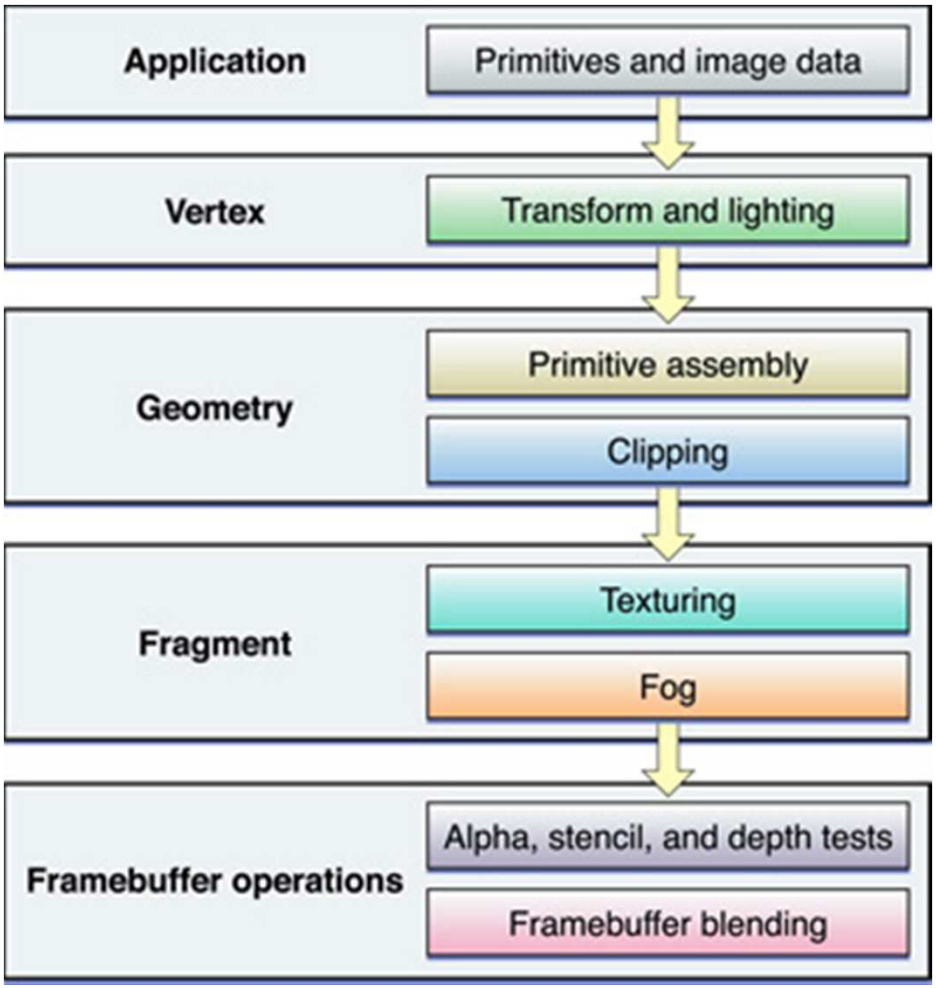
OpenGL API

- As a programmer, you need to do the following things:
  - Specify the location/parameters of camera.
  - Specify the geometry (and appearance).
  - Specify the lights (optional).
- OpenGL will compute the resulting 2D image!

OpenGL Conventions

- Many functions have multiple forms:
  - glVertex2f, glVertex3i, glVertex4dv, etc.
- Number indicates number of arguments
- Letters indicate type
  - f: float, d: double, ub: unsigned byte, etc.
- ‘v’ (if present) indicates a single pointer argument
  - glVertex3f(point.x, point.y, point.z)
  - glVertex3fv(point)

OpenGl Pipeline(Basic Version)



OpenGL: Camera

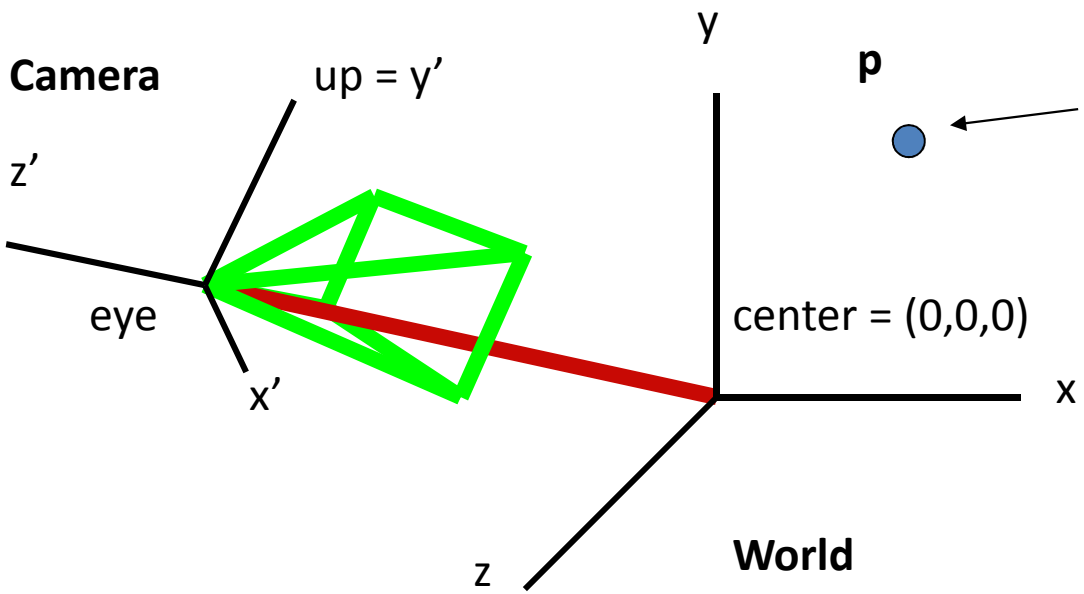
- Two things to specify:
  - Physical location of camera in the scene (MODELVIEW matrix in OpenGL).
    - Where is the camera?
    - Which direction is it pointing?
    - What is the orientation of the camera?
  - Projection properties of the camera (PROJECTION matrix in OpenGL):
    - Depth of field?
    - Field of view in the x and y directions?

OpenGL: Camera

- Coordinate-systems are represented as matrices in OpenGL.
- Think of camera as implying a coordinate-system centered at its image plane.
- Therefore, specifying this matrix implies specifying the physical properties of the camera.

OpenGL: MODELVIEW

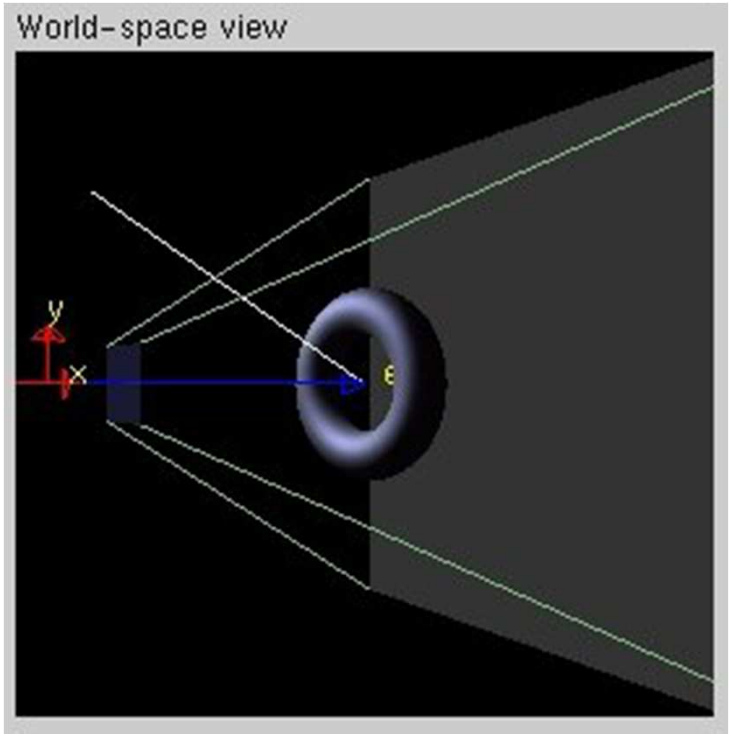
```
glMatrixMode(GL_MODELVIEW);  // Specify matrix mode
glLoadIdentity();                    // Clear modelview matrix
//Utility function provided by the GLU API (included with OpenGL)
gluLookAt(eyex,eyey,eyez,centerx,centery,centerz,upx,upy,upz);
```



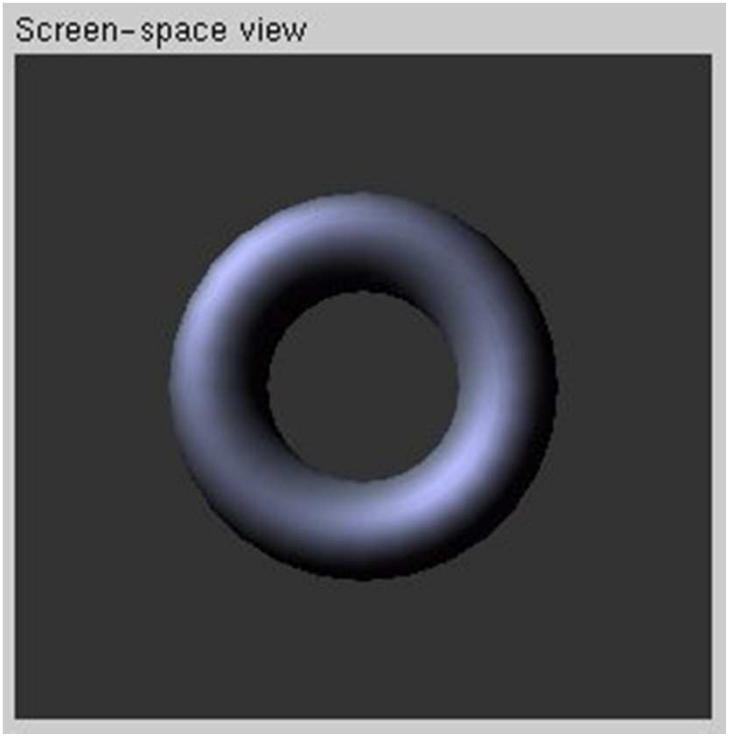
Point **p** in world is:
(MODELVIEW * **p**)
in camera coord-sys!!!

OpenGL: MODELVIEW

World coord-sys:

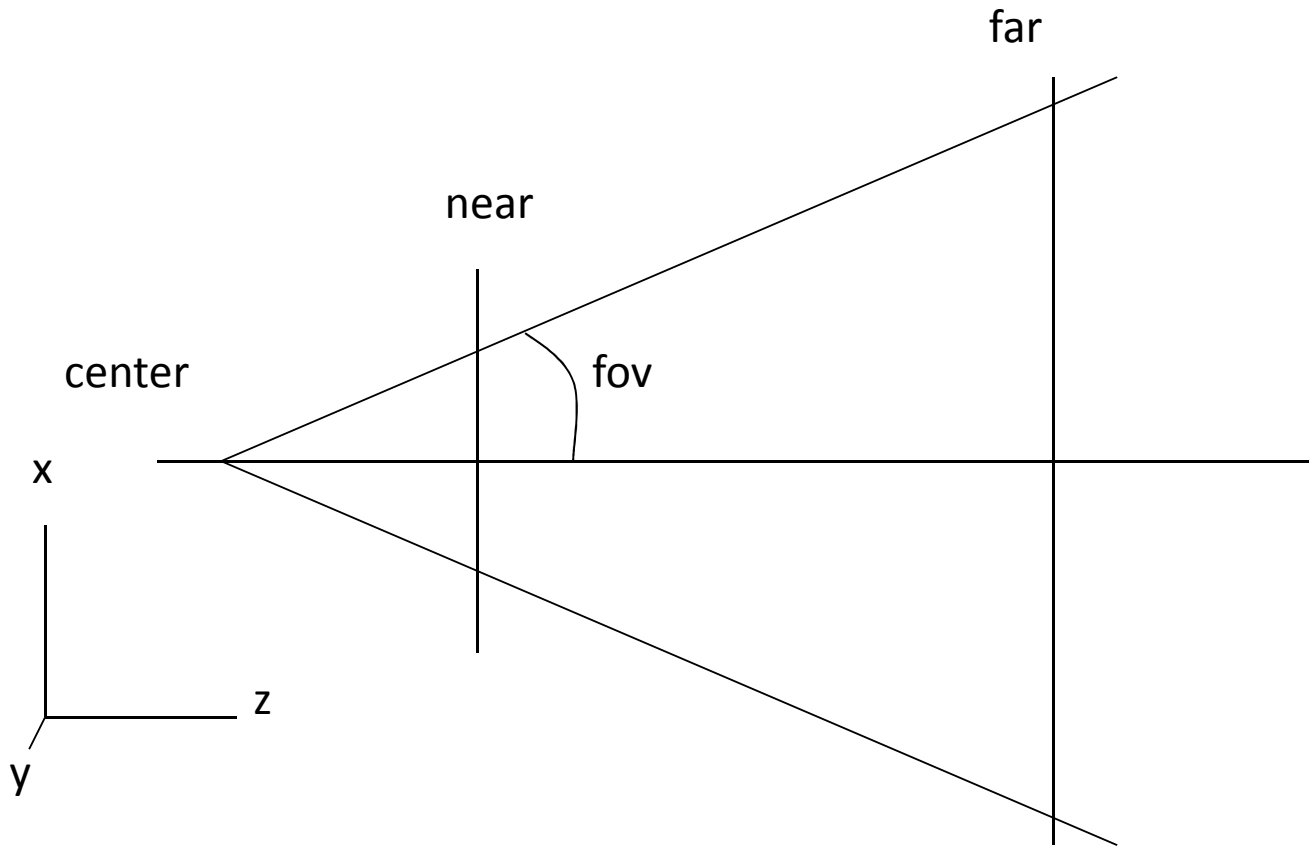


Camera coord-sys:



OpenGL: PROJECTION

- Intrinsic (optical) properties of camera:



OpenGL: PROJECTION

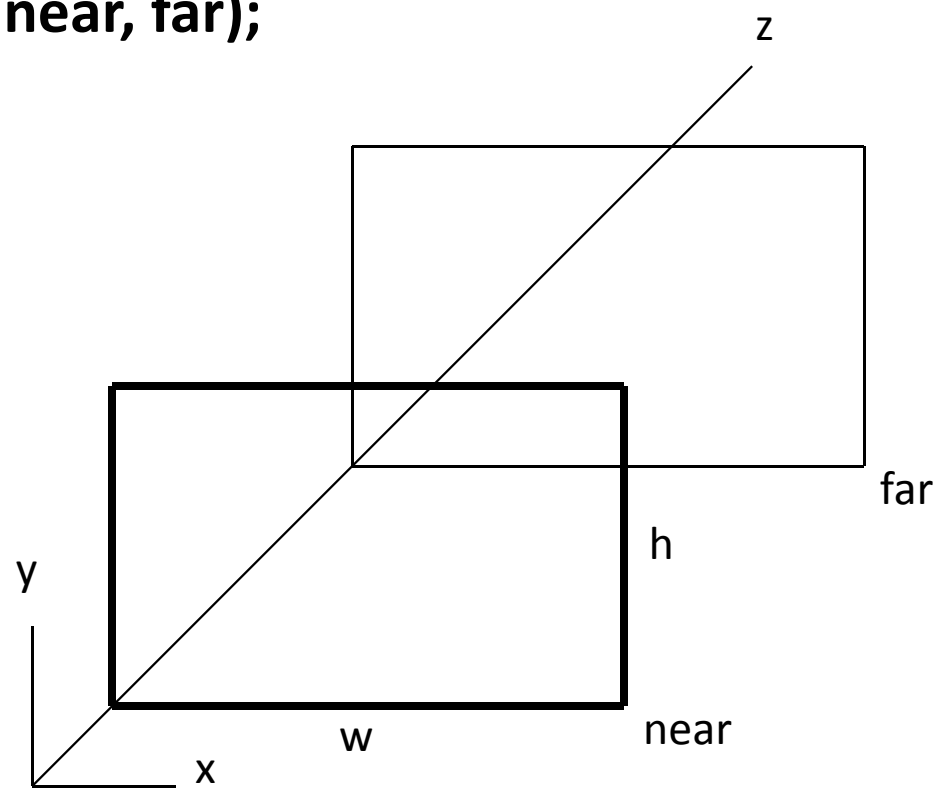
glMatrixMode(GL_PROJECTION);

glLoadIdentity();

gluPerspective(fovy, aspect, near, far);

Camera viewport:

aspect = w/h



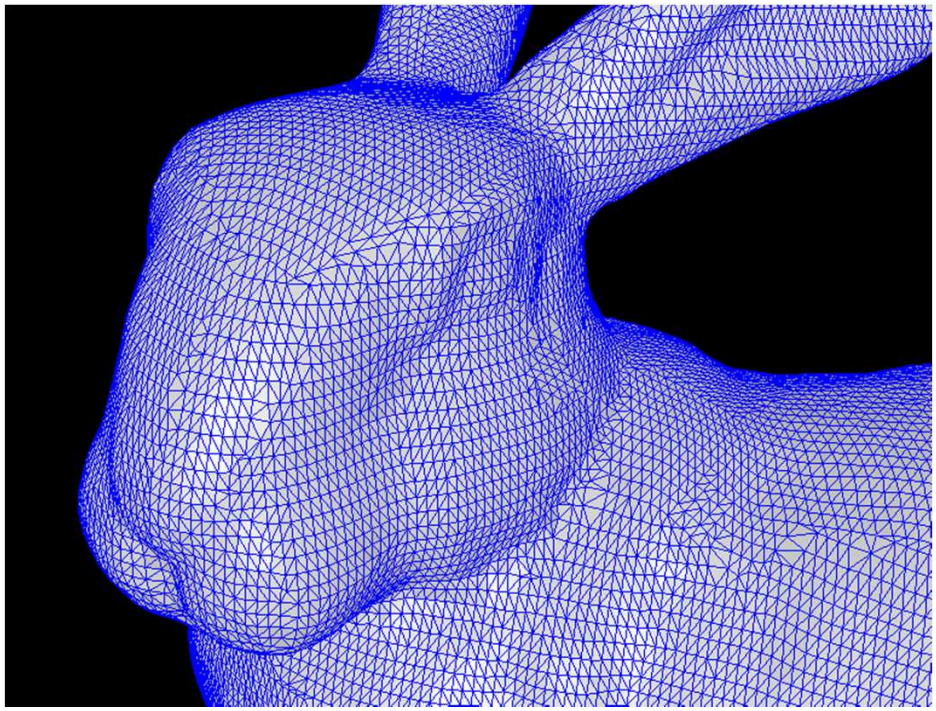
OpenGL: Setting Camera

- Assume window is **widthxheight**:

```
void SetCamera()
{
   glViewport(0, 0, width, height);
   /* Set camera position */
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   gluLookAt(m_vEye[0], m_vEye[1], m_vEye[2],
       m_vRef[0], m_vRef[1], m_vRef[2],
       m_vUp[0],  m_vUp[1],  m_vUp[2]);
   /* Set projection frustrum */
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(m_fYFOV, width / height, m_fNear, m_fFar);
}
```

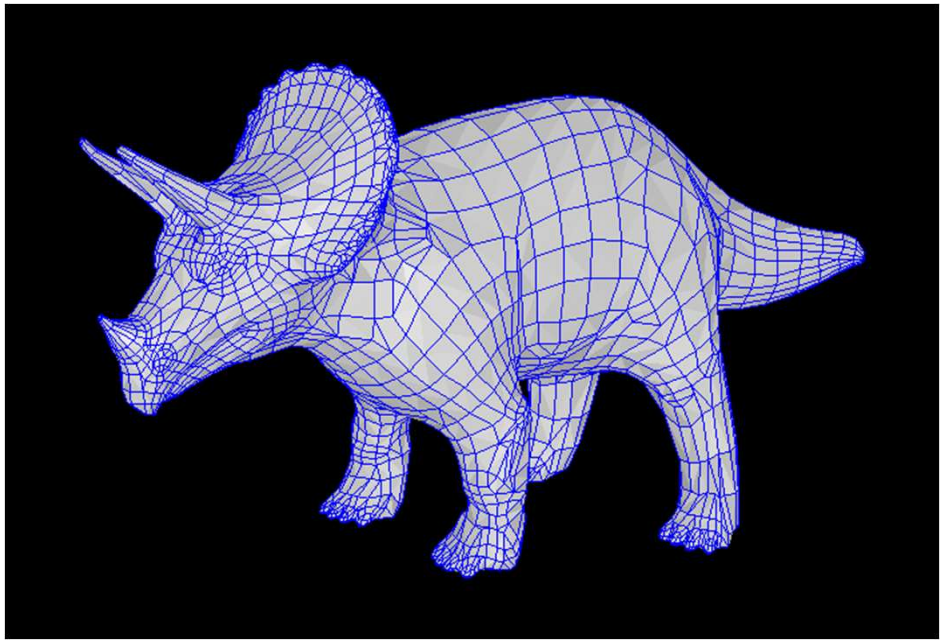
OpenGL: Geometry

- Specify geometry using primitives: triangles, quadrilaterals, lines, etc…



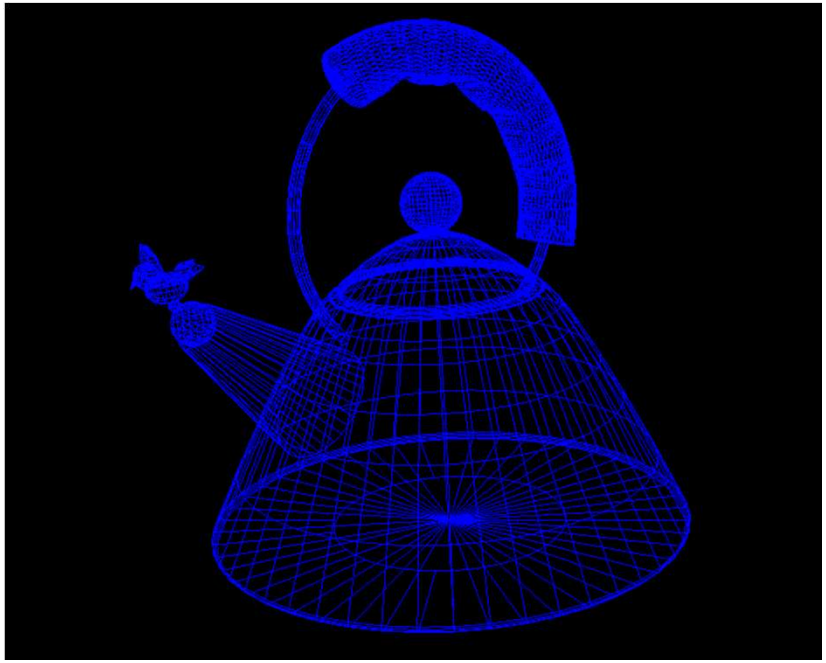
OpenGL: Geometry

- Specify geometry using primitives: triangles, quadrilaterals, lines, points, etc…



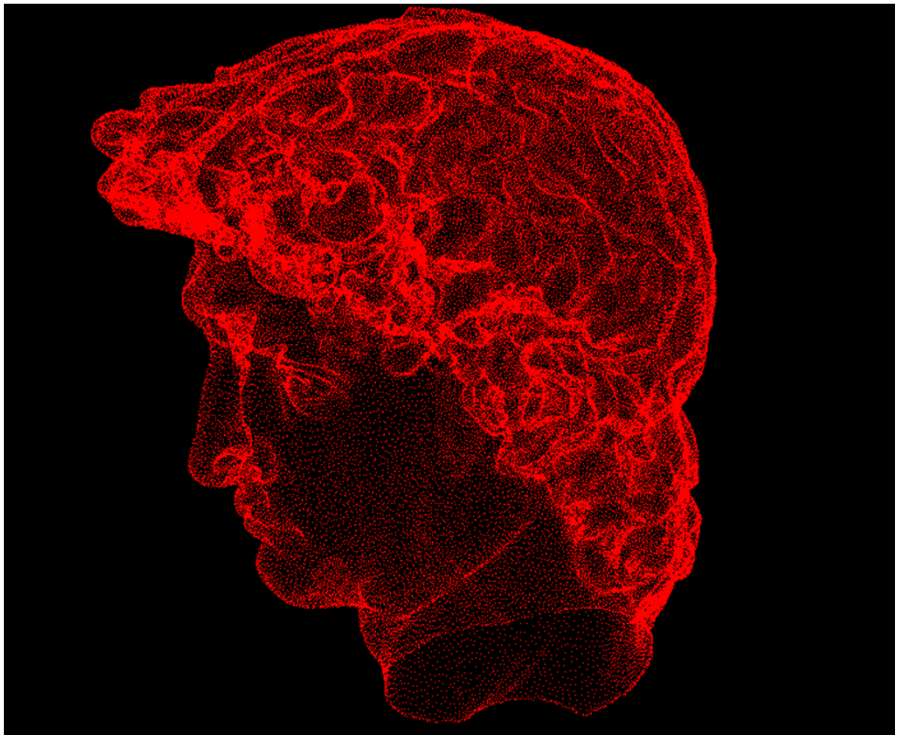
OpenGL: Geometry

- Specify geometry using primitives: triangles, quadrilaterals, lines, points, etc…



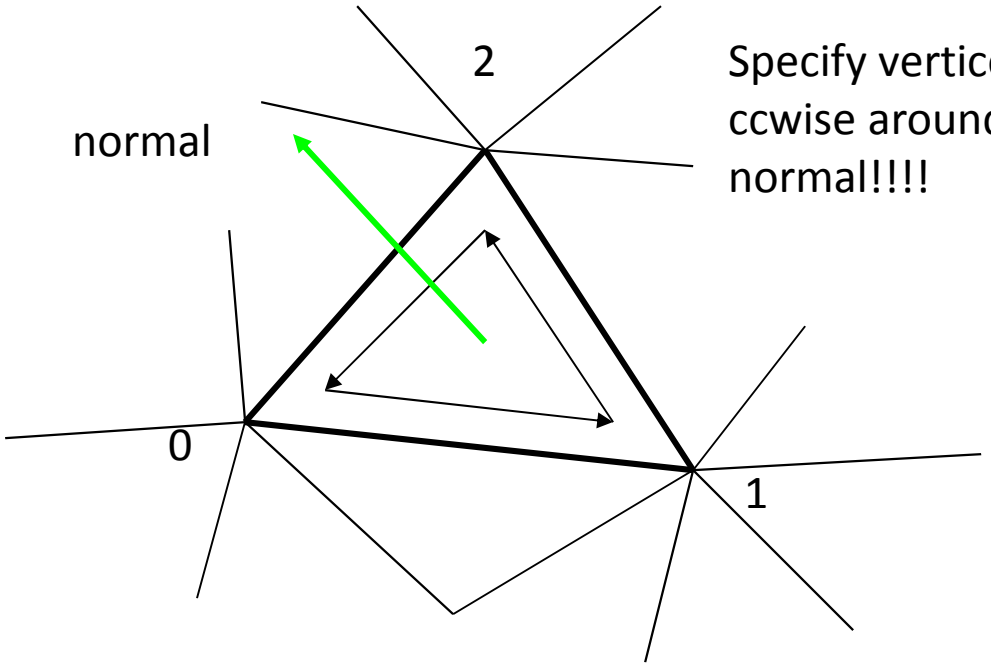
OpenGL: Geometry

- Specify geometry using primitives: triangles, quadrilaterals, lines, points, etc…



OpenGL: Triangle Mesh

- Represent the surface of an object by a collection of **oriented** triangles:



OpenGL: glBegin()…glEnd()

- Geometry passed btwn glBegin(.), glEnd(.)!!!

```
glBegin(GL_TRIANGLES);
for (int i=0; i<ntris; i++)
{
    glColor3f(tri[i].r0,tri[i].g0,tri[i].b0);          // Color of vertex
    glNormal3f(tri[i].nx0,tri[i].ny0,tri[i].nz0);    // Normal of vertex
    glVertex3f(tri[i].x0,tri[i].y0,tri[i].z0);         // Position of vertex
    …
    glColor3f(tri[i].r2,tri[i].g2,tri[i].b2);
    glNormal3f(tri[i].nx2,tri[i].ny2,tri[i].nz2);
    glVertex3f(tri[i].x2,tri[i].y2,tri[i].z2);
}
glEnd();   // Sends all the vertices/normals to the OpenGL library
```

OpenGL: glBegin()…glEnd()

- OpenGL supports many primitives:

  glBegin(GL_LINES);

  glBegin(GL_QUADS);

  glBegin(GL_POLYGON);
- Use GL_LINES to specify a line in 3D space (such as a ray!!!).
- All vertices **can** have a normal (we will see why next!).

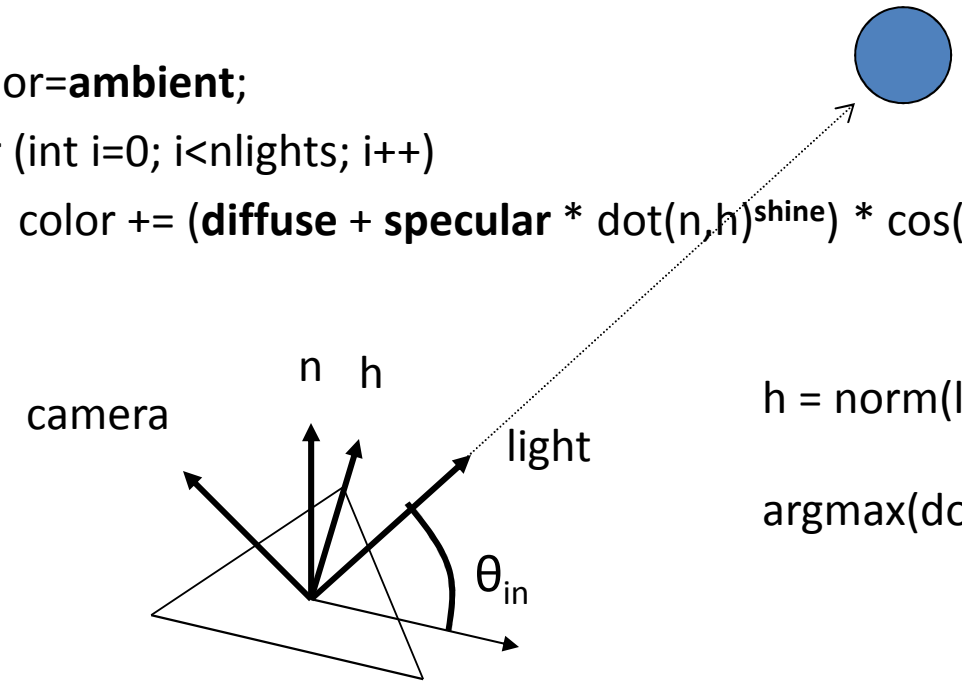
OpenGL: Lighting

- OpenGL computes the color at each vertex with a **shading** calculation:

color=**ambient**;

for (int i=0; i<nlights; i++)

color += (**diffuse** + **specular** * dot(n,h)$^{\textbf{shine}}$) * cos(θ_{in}) * light_color[i];



h = norm(light+camera)

argmax(dot(n,h)): h=n!

OpenGL: Lighting

```
GLfloat mat_diffuse[4] = {0.75, 0.75, 0.75, 1.0};
GLfloat mat_specular[4] = {0.55, 0.55, 0.55, 1.0};
GLfloat mat_shininess[1] = {80};

glMaterialfv(GL_FRONT_AND_BACK,GL_AMBIENT_AND_DIFFUSE,mat_diffuse);
glMaterialfv(GL_FRONT,GL_SPECULAR,mat_specular);
glMaterialfv(GL_FRONT,GL_SHININESS,mat_shininess);

GLfloat light0_ambient[4] = { 0.0, 0.0, 0.0, 1.0};
GLfloat light0_color[4] = { 0.4, 0.4, 0.4, 1.0 };
glLightfv(GL_LIGHT0, GL_AMBIENT, light0_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light0_color);
glLightfv(GL_LIGHT0, GL_SPECULAR, light0_color);

glEnable(GL_LIGHT0);
glEnable(GL_LIGHTING);
```

OpenGL: Lighting

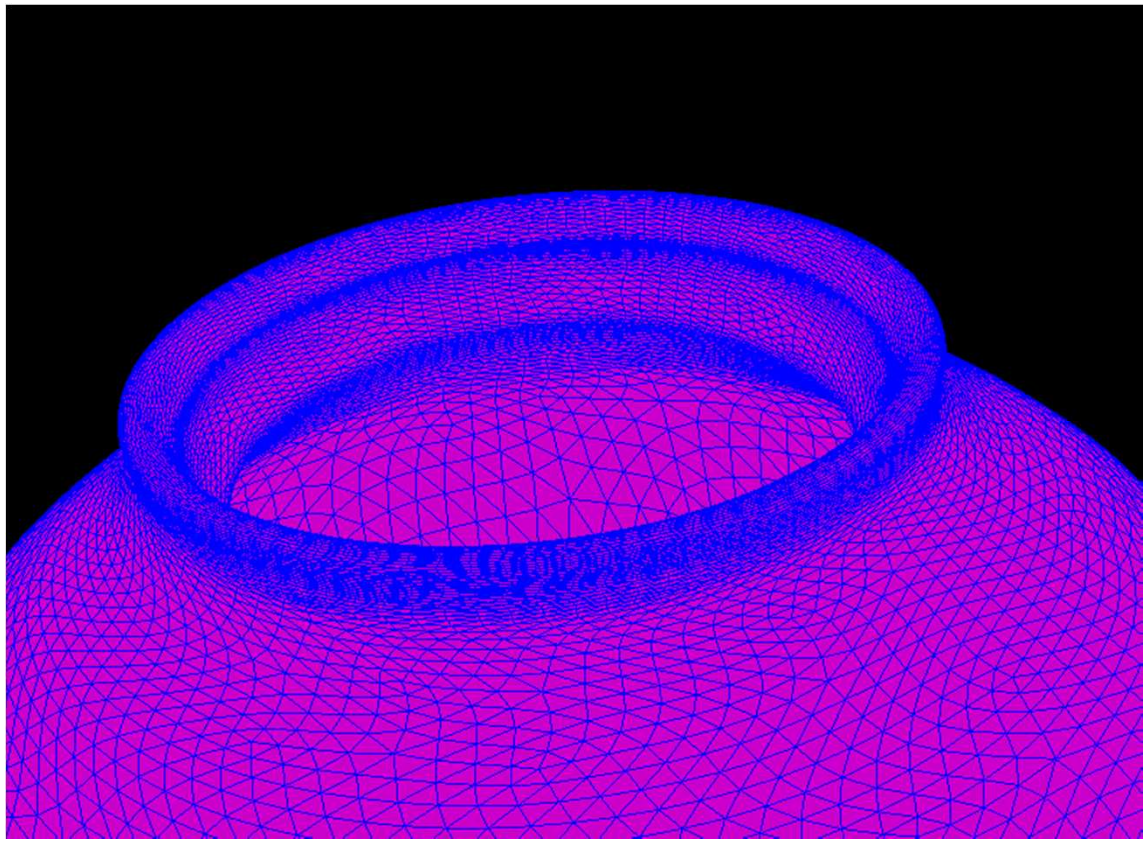
```
GLfloat mat_diffuse[4] = {0.75, 0.75, 0.75, 1.0};
GLfloat mat_specular[4] = {0.55, 0.55, 0.55, 1.0};
GLfloat mat_shininess[1] = {80};

glMaterialfv(GL_FRONT_AND_BACK,GL_AMBIENT_AND_DIFFUSE,mat_diffuse);
glMaterialfv(GL_FRONT,GL_SPECULAR,mat_specular);
glMaterialfv(GL_FRONT,GL_SHININESS,mat_shininess);

GLfloat light0_ambient[4] = { 0.0, 0.0, 0.0, 1.0};
GLfloat light0_color[4] = { 0.4, 0.4, 0.4, 1.0 };
glLightfv(GL_LIGHT0, GL_AMBIENT, light0_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light0_color);
glLightfv(GL_LIGHT0, GL_SPECULAR, light0_color);

glEnable(GL_LIGHT0);
glEnable(GL_LIGHTING);
```

OpenGL: Lighting

- There are two “shading modes” in OpenGL:
  - Flat shading: entire face is the color computed at the 0^{th} vertex.
  - Gouraud (“smooth”) shading: color computed at each vertex and interpolated across polygon.

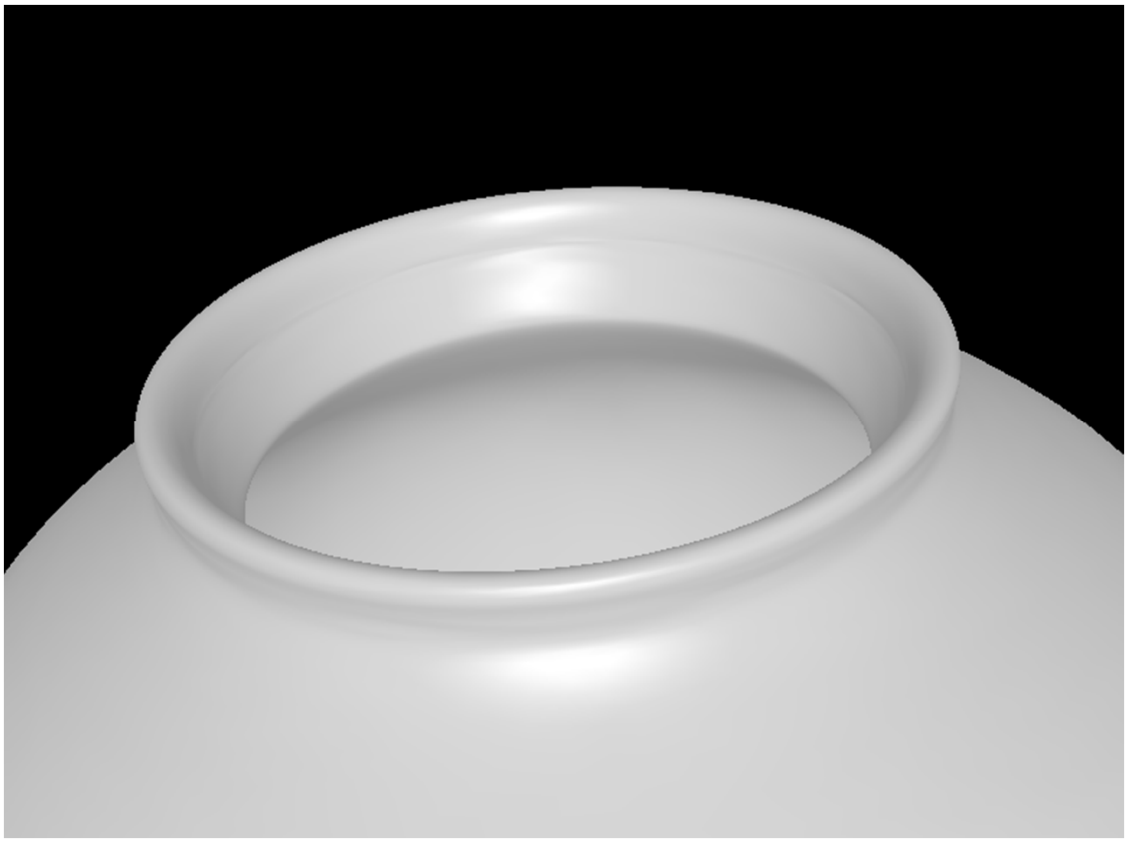
OpenGL: Flat Shading

glShadeModel(GL_FLAT);



OpenGL: Smooth Shading

glShadeModel(GL_SMOOTH);



OpenGL: Lighting is Optional!

- Lighting is an option!
  - glEnable(GL_LIGHTING);
  - glDisable(GL_LIGHTING);
- When disabled, the color of each vertex is directly glColor3f(r,g,b).
- Do not need lighting when debugging your ray tracer.

GLUT

- GLUT is a **very simple** API that supports creating GUI+OpenGL apps.

http://www.xmission.com/~nate/glut.html

GLUT

- Call these functions in your main:

```
/* Initialize glut engine */
 glutInit(&argc,argv);
 glutInitWindowPosition(100,100);
 glutInitWindowSize(window_width,window_height);
 glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
 main_window = glutCreateWindow(“Ray Viewer");

 /* Register callbacks */
 glutDisplayFunc(MainRedraw);   // Main redraw function

 /* Run the interactive interface */
 glutMainLoop();
```

GLUT: MainRedraw

```
void MainRedraw()
{
    glClearColor(0,0,0,1);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    SetCamera();
    DrawGeometry();
    glutSwapBuffers();  // Update the screen
}
```

OpenGL: Resources

- OpenGL Programming Guide
  - http://www.glprogramming.com/red/
  - http://www.cse.chalmers.se/edu/year/2011/course/TDA361/2007/redbook.pdf
- Other online resources:
  - Industry-wide OpenGL Information: http://www.opengl.org
  - OpenGL specifications: http://www.opengl.org/documentation/specs
  - OpenGL Man Pages: http://www.opengl.org/sdk/docs/man
  - OpenGL SDK: http://www.opengl.org/sdk
  - GLSL Quick Reference Guide: http://www.opengl.org/sdk/libs/OpenSceneGraph/glsl_quickref.pdf
  - OpenGL Tutorials: http://www.lighthouse3d.com/opengl/tutorials.shtml