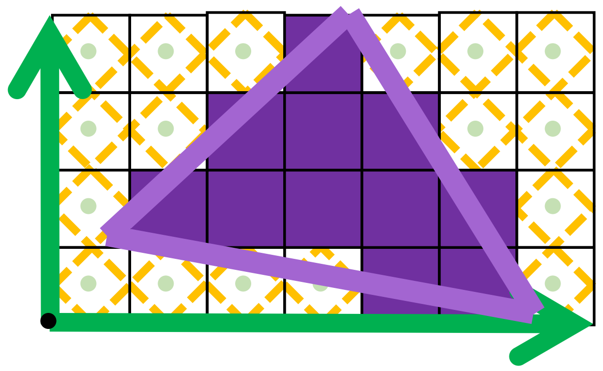
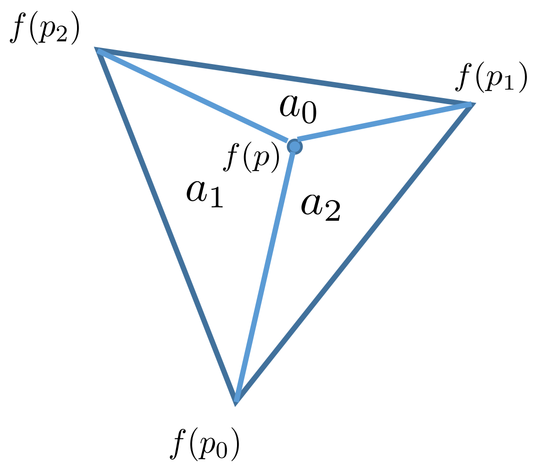
OpenGL

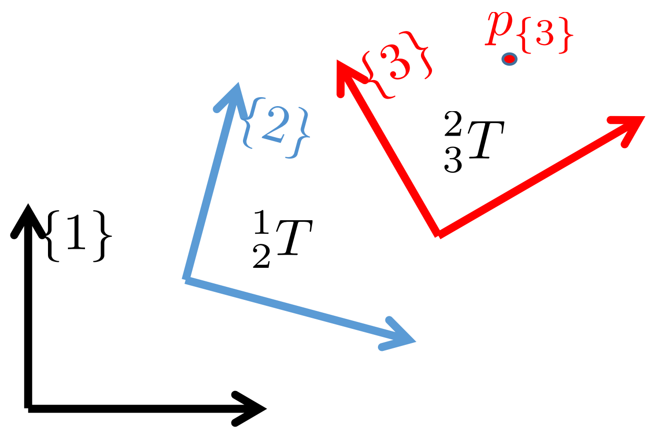
CS 148: Summer 2016
Introduction of Graphics and Imaging
Zahid Hossain

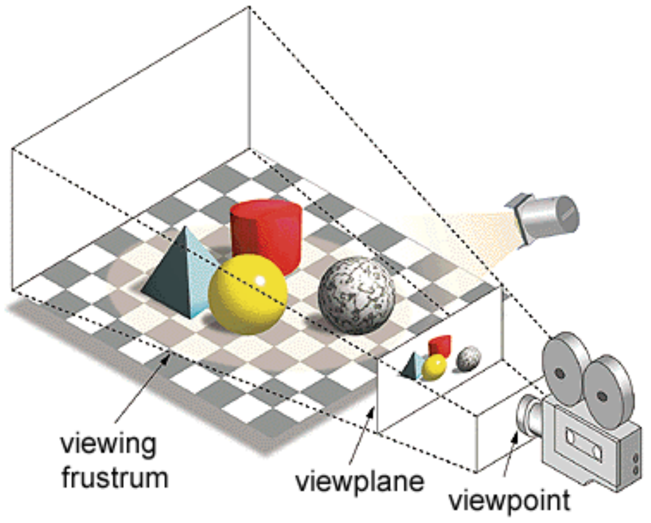
http://www.pling.org.uk/cs/cgv.html

So Far: Theory of Rasterization









Observation

- Apply transformation
- Barycentric Interpolation
- Rasterize
- Compute Light and Shading (More on it later)
- Lookup Textures (More on it later)
- And lot more ….

Observation

- Apply transformation
- Barycentric Interpolation
- Rasterize
- Compute Light and Shading (More on it later)
- Lookup Textures (More on it later)
- And lot more ….

To
Millions of Triangles
and
Billions of Fragments

Observation

- Apply transformation
- Barycentric Interpolation
- Rasterize
- Compute Light and Shading (More on it later)
- Lookup Textures (More on it later)
- And lot more ….

To
<u>Millions of Triangles</u>
and
<u>Billions of Fragments</u>

SIMD
(Single Instruction Multiple Data)

Graphics Hardware

2560 cores! (upto 1733 MHz, 8GB Mem)

http://www.geforce.com/hardware/10series/geforce-gtx-1080

Advice

Leave implementation of low-level features to the experts.

Advice

Leave implementation of low-level features to the experts.

Why ?

- Abstract away hardware differences
- Rasterization should be *fast*

Introducing …

Industry Standard API for Computer Graphics (Cross Platform, since 1992)

Not the Only One

Microsoft®
DirectX®

PIXAR's
RenderMan

AMD
MANTLE API
TECHNOLOGY

Vulkan™

Related APIs in the OpenGL Family

WebGL

Google Maps

OpenGL|ES

Angry Birds by Rovio

OpenVG WebCL OpenGL|SC OpenCL

Related APIs in the OpenGL Family

WebGL™

OpenGL|ES™

y Rovio

KHRONOS GROUP

OpenVG™ WebCL™ OpenGL|SC™

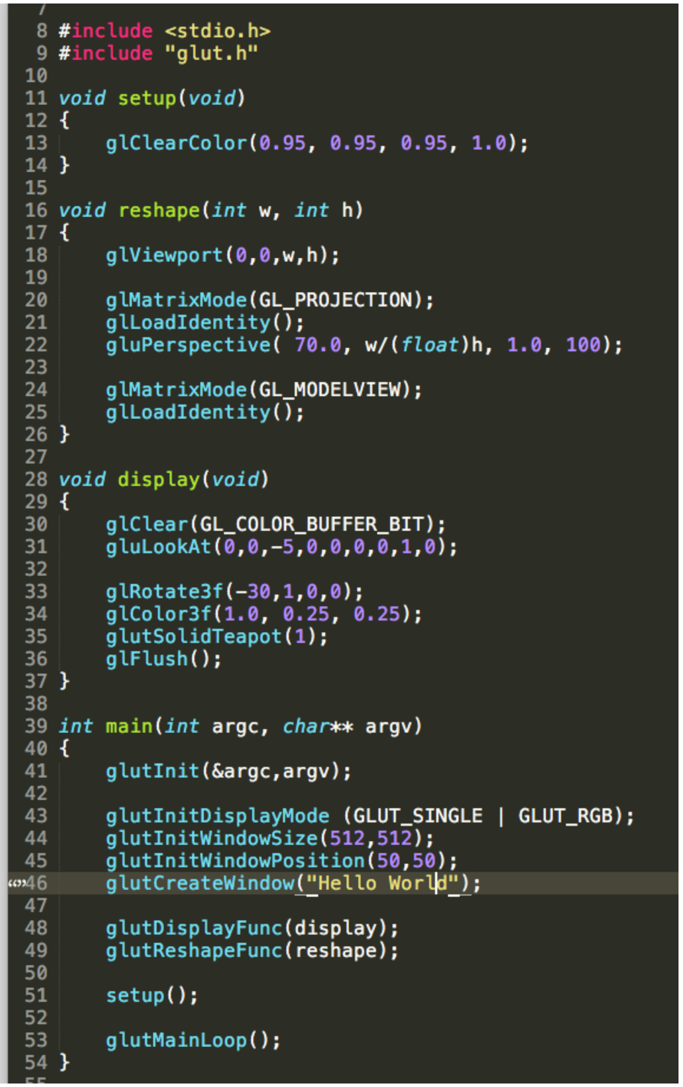
OpenCL

OpenGL 1.x

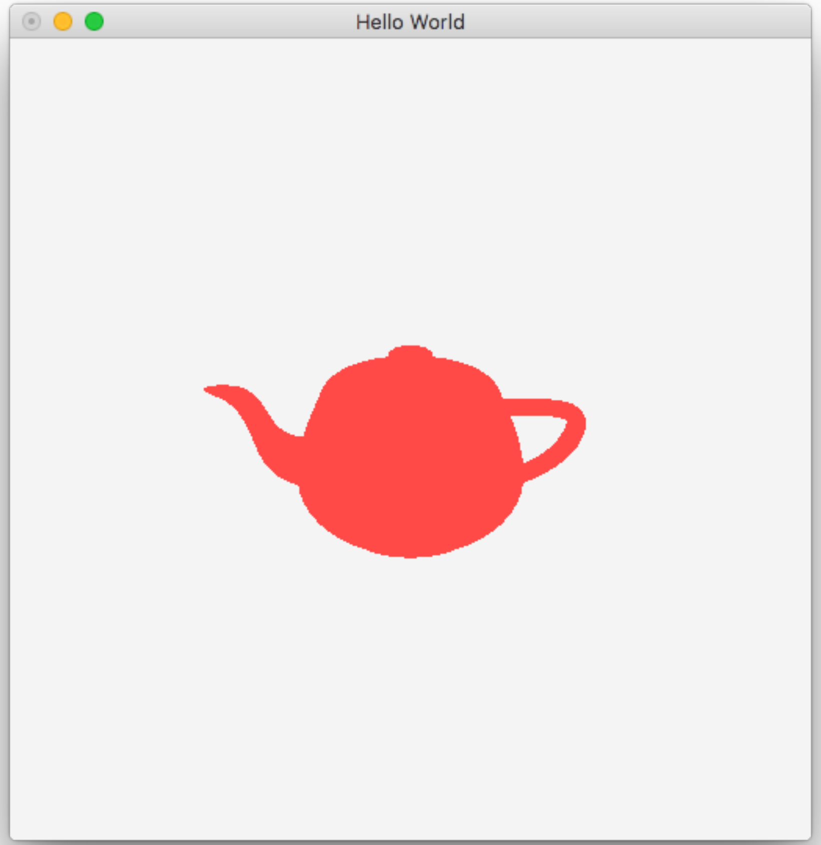
OpenGL 1.x: Why ?

- Easy to learn!
- Little code
- Instructional

Simple First Program



```
#include <stdio.h>
#include "glut.h"

void setup(void)
{
    glClearColor(0.95, 0.95, 0.95, 1.0);
}

void reshape(int w, int h)
{
    glViewport(0,0,w,h);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective( 70.0, w/(float)h, 1.0, 100);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    gluLookAt(0,0,-5,0,0,0,0,1,0);

    glRotate3f(-30,1,0,0);
    glColor3f(1.0, 0.25, 0.25);
    glutSolidTeapot(1);
    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc,argv);

    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(512,512);
    glutInitWindowPosition(50,50);
    glutCreateWindow("Hello World");

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);

    setup();

    glutMainLoop();
}
```



What OpenGL Is *Not*

OpenGL is not a windowing system

Platform Independent !

Introducing GLUT (GL Utility Toolkit)

- Simple cross-platform windowing API
  - `glutInitDisplay(), glutInitWindowSize(..)`
- Bindings: C, C++, Fortran, Ada, …
- Features:
  - Multiple windows, menus
  - Keyboard/mouse/other input
  - Assorted callbacks: idle, timers
  - Basic font support
  - `glutSolidTeapot, glutSolidSphere, glutSolidCube, …`

Introducing GLUT (GL Utility Toolkit)

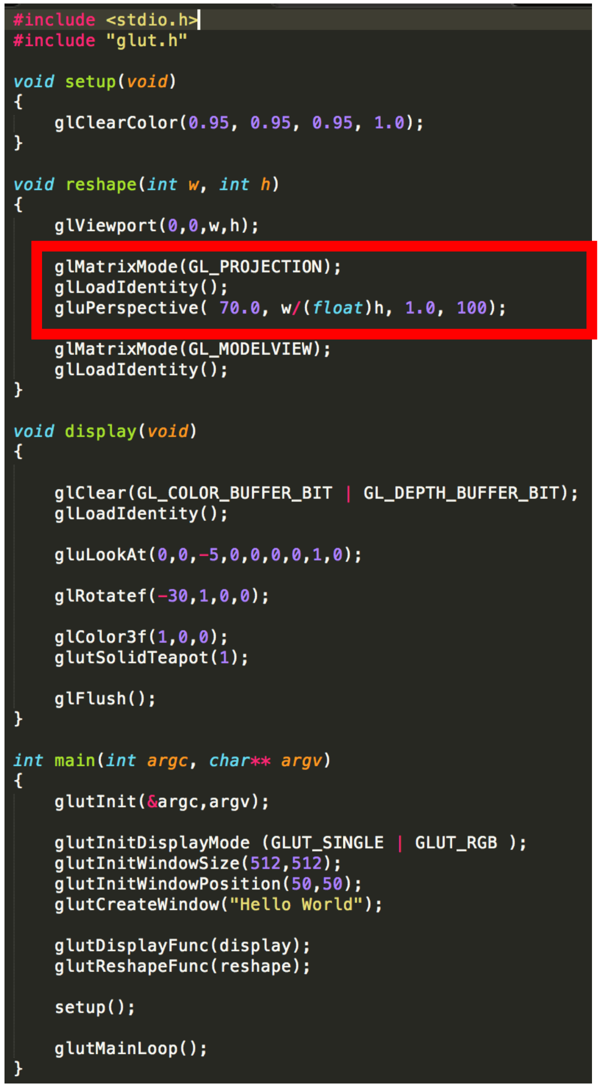
- Simple cross-platform windowing API
  - `glutInitDisplay(), glutInitWindowSize(..)`
- Bindings: C, C++, Fortran, Ada, …
- Features:
  - Multiple windows, menus
  - Keyboard/mouse/other input
  - Assorted callbacks: idle, timers
  - Basic font support
  - `glutSolidTeapot, glutSolidSphere, glutSolidCube, …`

Other options: glfw, SDL

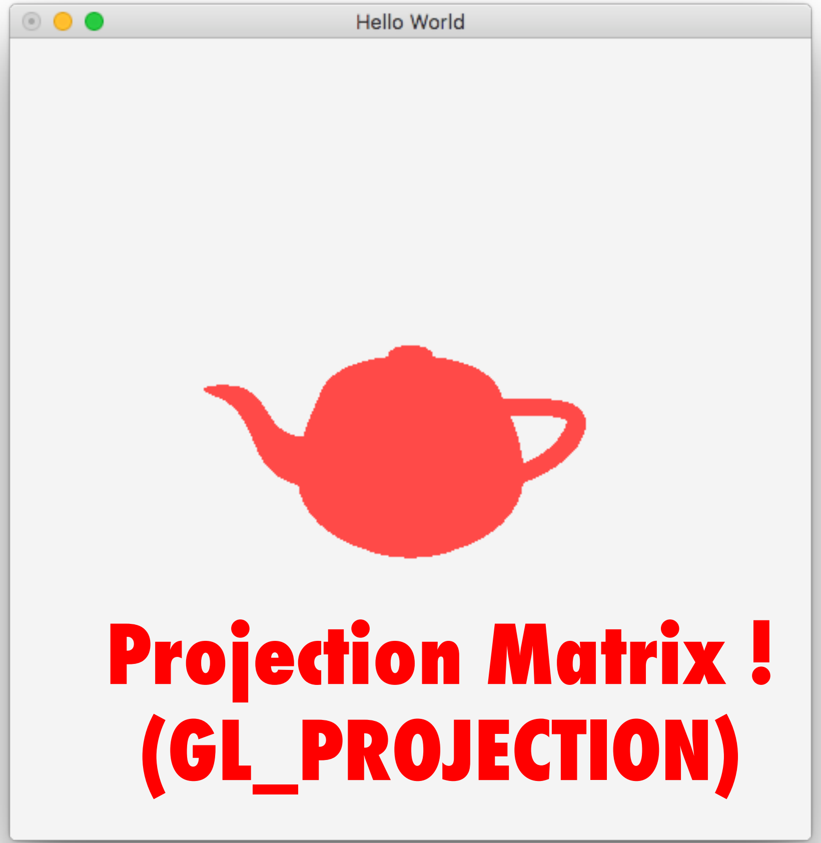
Introducing GLU (GL Utility)

- High-level graphics commands
- *Not* included in OpenGL ES
- Some interesting features:
    - Mapping between world and screen coordinates
    - Texturing support
    - Tessellation and other geometric utilities
    - OpenGL error code lookup
    - More primitives: spheres, cylinders, disks, …
    - Camera support: `gluLookAt, gluOrtho2D, …`

Simple First Program

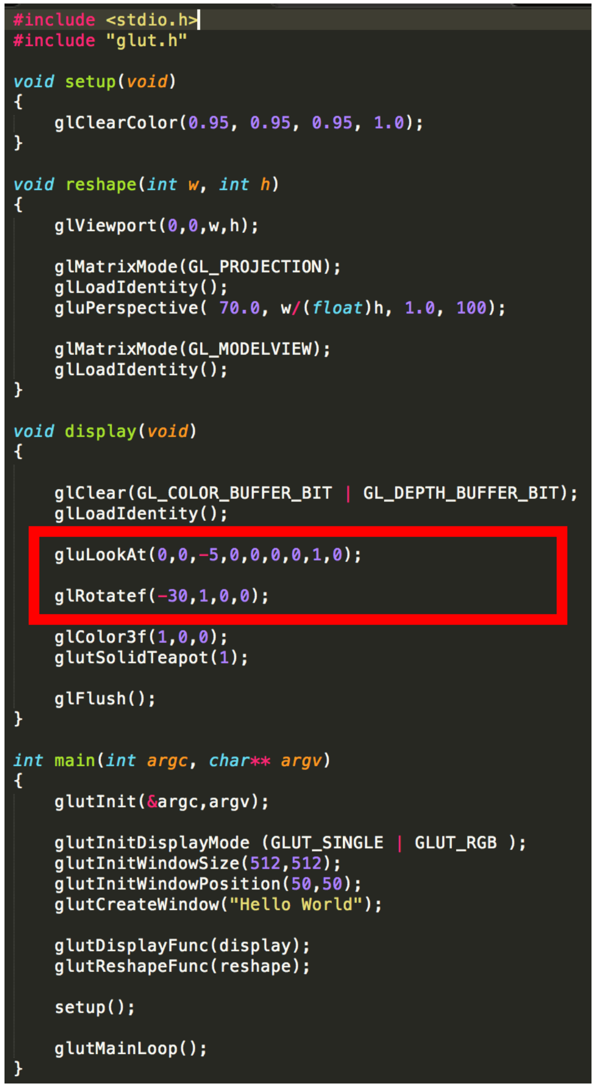


```
#include <stdio.h>
#include "glut.h"

void setup(void)
{
    glClearColor(0.95, 0.95, 0.95, 1.0);
}

void reshape(int w, int h)
{
    glViewport(0,0,w,h);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective( 70.0, w/(float)h, 1.0, 100);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void display(void)
{

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();

    gluLookAt(0,0,-5,0,0,0,0,1,0);

    glRotatef(-30,1,0,0);

    glColor3f(1,0,0);
    glutSolidTeapot(1);

    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc,argv);

    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB );
    glutInitWindowSize(512,512);
    glutInitWindowPosition(50,50);
    glutCreateWindow("Hello World");

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);

    setup();

    glutMainLoop();
}
```

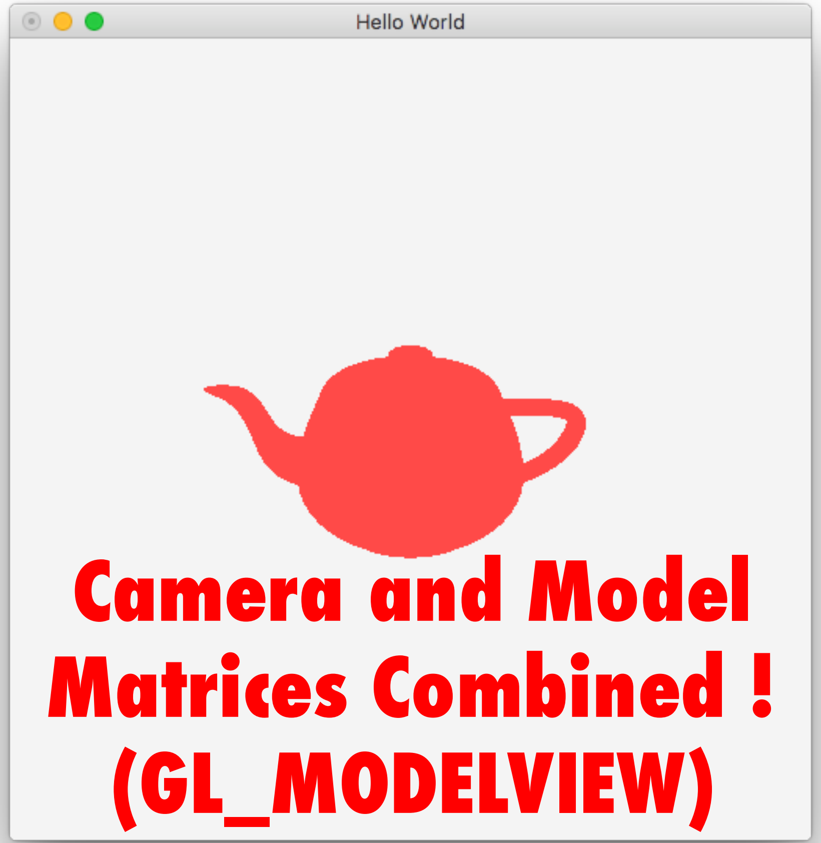


Projection Matrix !
(GL_PROJECTION)

Simple First Program



```
#include <stdio.h>
#include "glut.h"

void setup(void)
{
    glClearColor(0.95, 0.95, 0.95, 1.0);
}

void reshape(int w, int h)
{
    glViewport(0,0,w,h);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective( 70.0, w/(float)h, 1.0, 100);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void display(void)
{

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();

    gluLookAt(0,0,-5,0,0,0,0,1,0);

    glRotatef(-30,1,0,0);

    glColor3f(1,0,0);
    glutSolidTeapot(1);

    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc,argv);

    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB );
    glutInitWindowSize(512,512);
    glutInitWindowPosition(50,50);
    glutCreateWindow("Hello World");

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);

    setup();

    glutMainLoop();
}
```



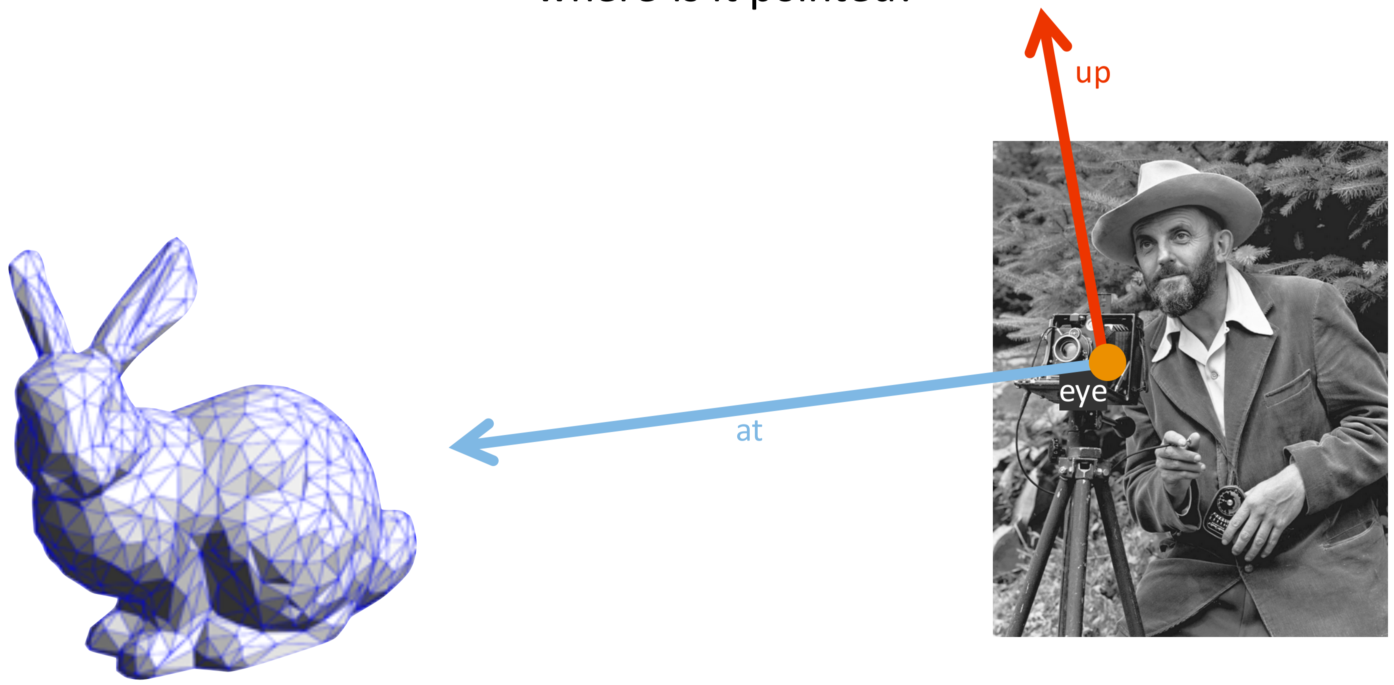
Camera and Model Matrices Combined ! (GL_MODELVIEW)

gluLookAt(eye, at, up)

where is the viewpoint?

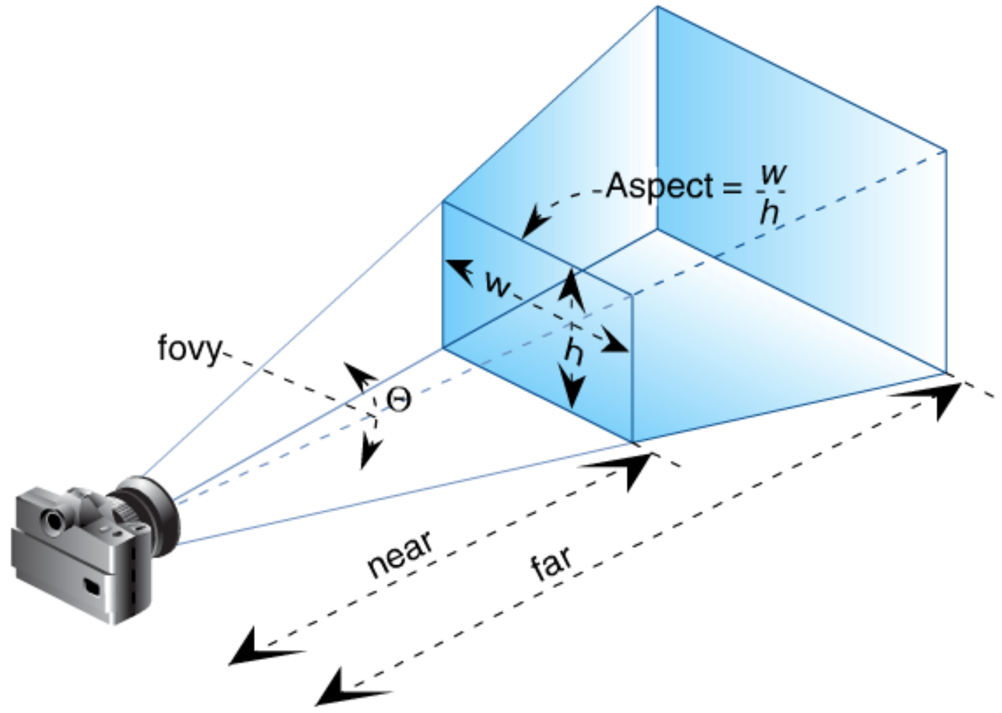
where is it pointed?

which way is up?



defines the camera/viewing matrix!

gluPerspective(fovy, aspect, near, far)



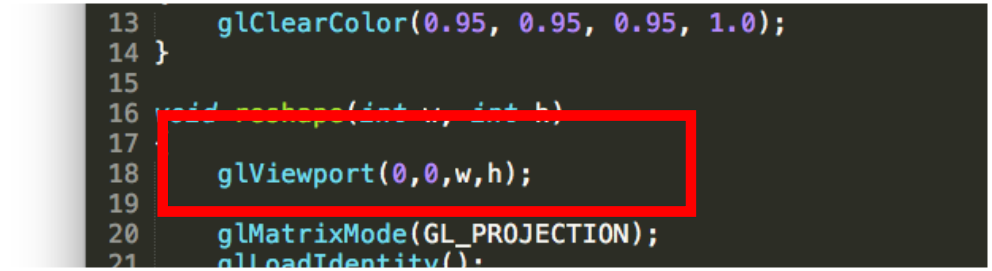
The Redbook, fig. 3-14 (p. 155)

Transformation Recall

$$NDC = M_{proj} \cdot \underbrace{M_{eye} \cdot M_{model}}_{\text{ModelView}} \cdot v$$

Transformation Recall

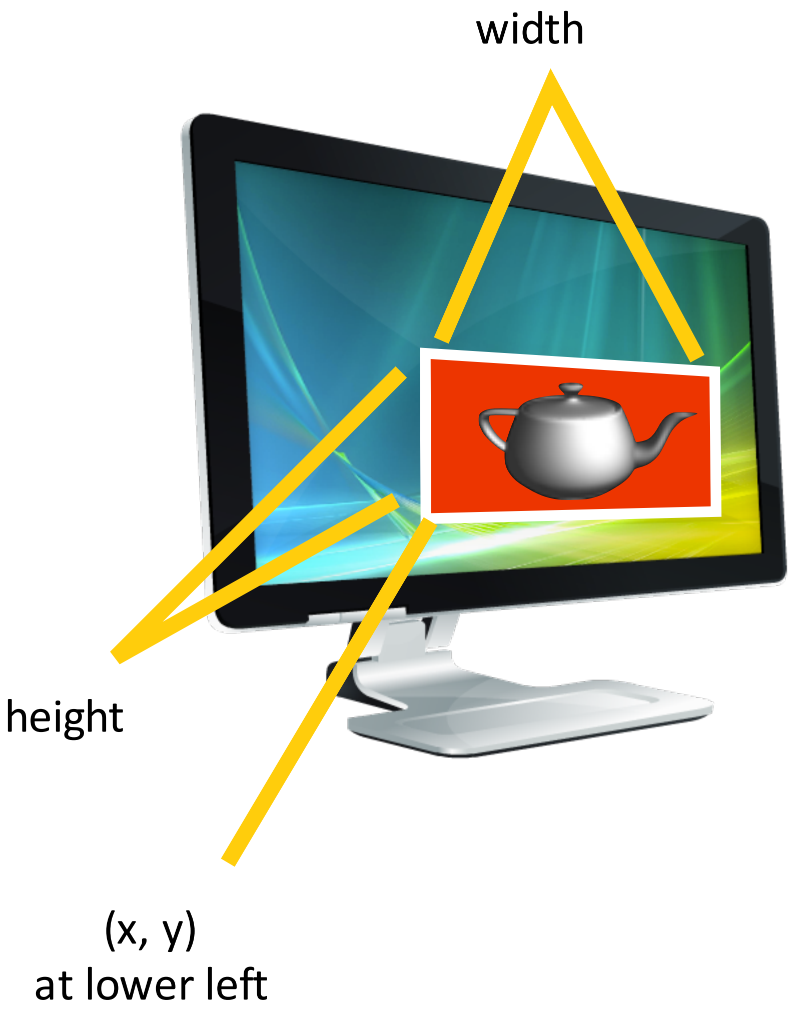
$$NDC = M_{proj} \cdot \underbrace{M_{eye} \cdot M_{model}}_{\text{ModelView}} \cdot v$$



```
    glClearColor(0.95, 0.95, 0.95, 1.0);
}

void reshape(int w, int h)
{
    glViewport(0,0,w,h);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
```

NDC → Viewport Transformation → Final Image

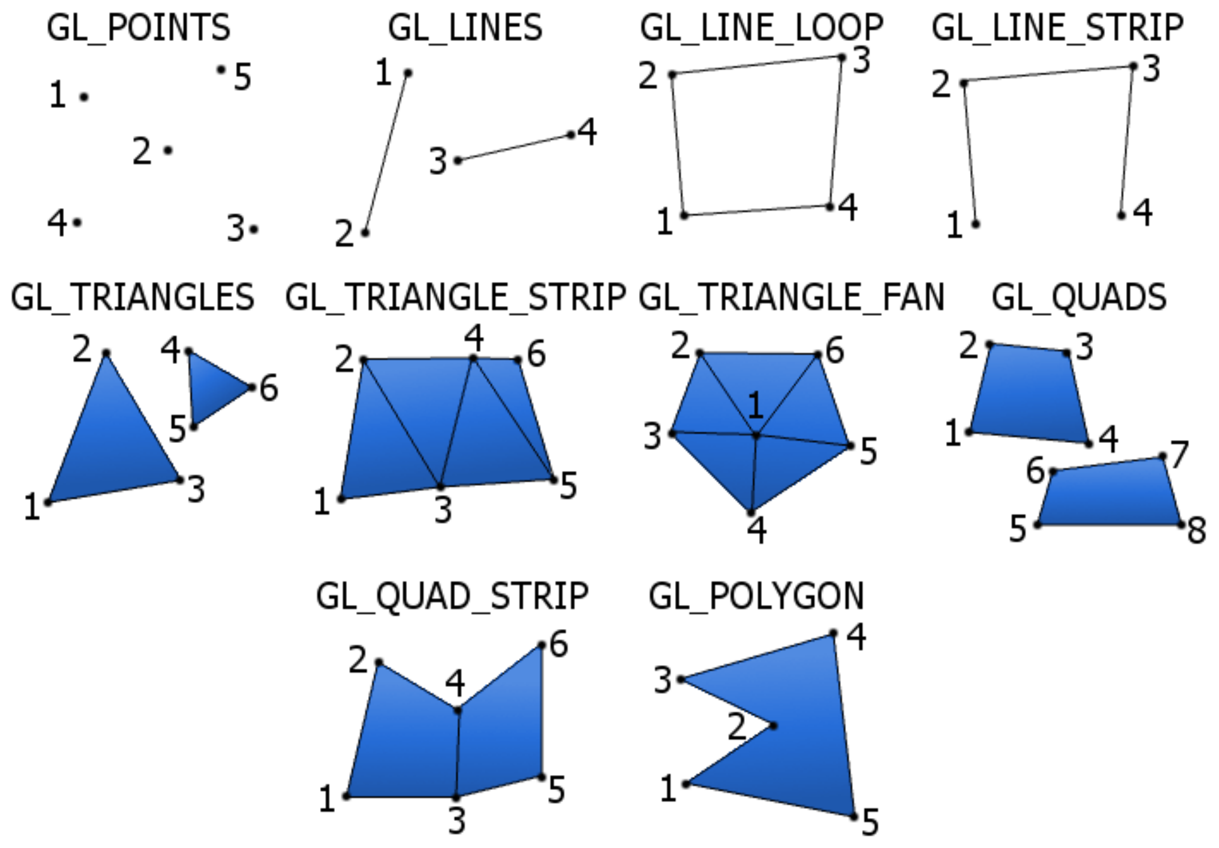
glViewport(x, y, width, height)



Primitive [prim-i-tiv]:

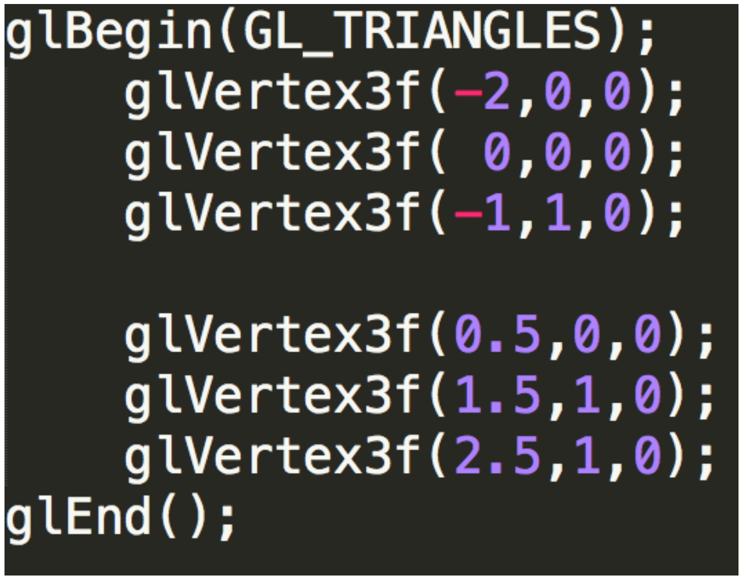
A small piece of geometry that can be rendered in OpenGL; e.g. triangles, lines, points etc.

OpenGL Primitive Types

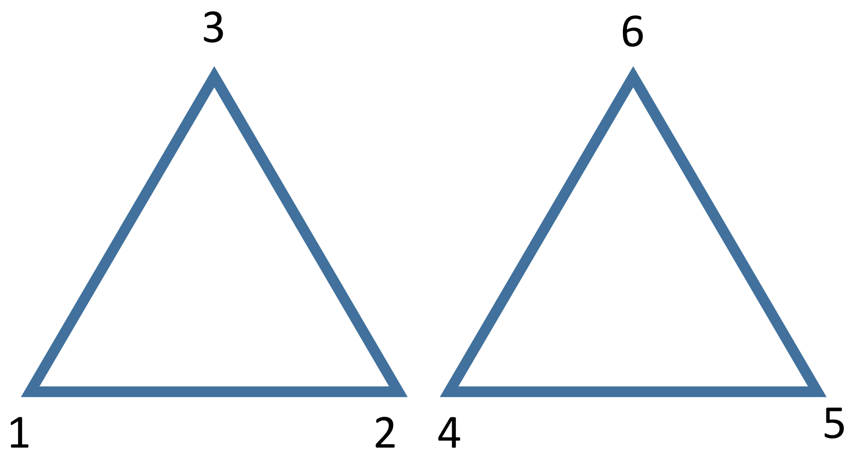


http://www.opengl2go.net/wp-content/uploads/2015/10/gl-primitives-with-background.png

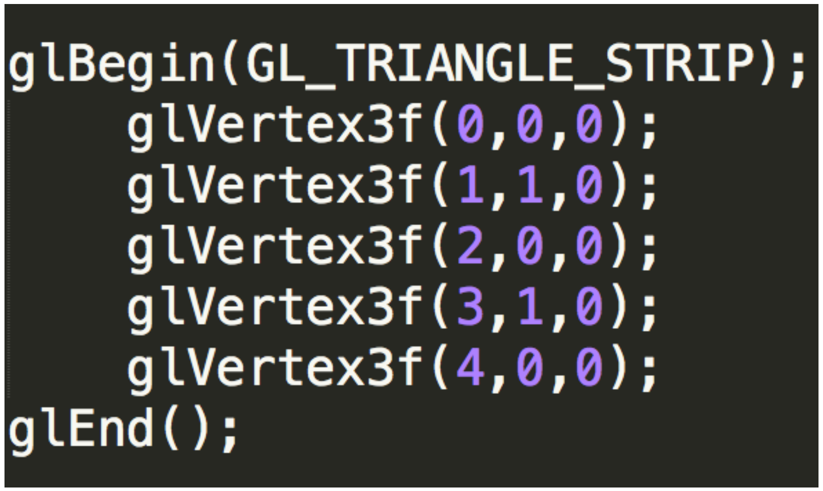
OpenGL Primitives (Triangles)



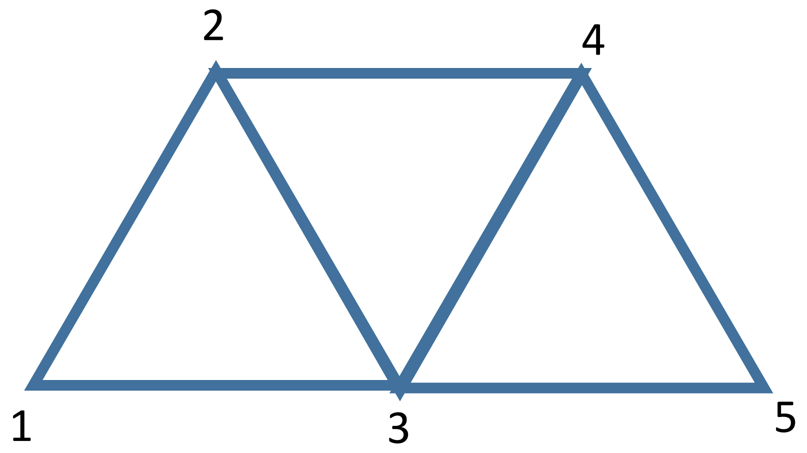
```
glBegin(GL_TRIANGLES);
    glVertex3f(-2,0,0);
    glVertex3f( 0,0,0);
    glVertex3f(-1,1,0);

    glVertex3f(0.5,0,0);
    glVertex3f(1.5,1,0);
    glVertex3f(2.5,1,0);
glEnd();
```



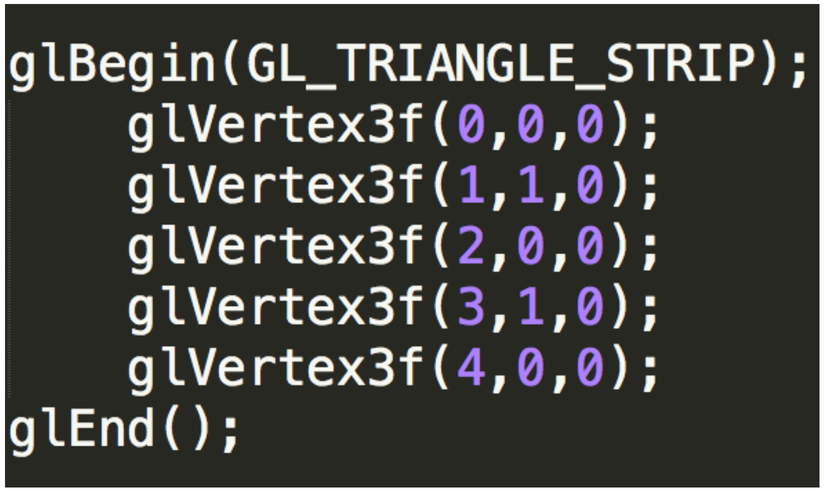
OpenGL Primitives (Triangle Strip)



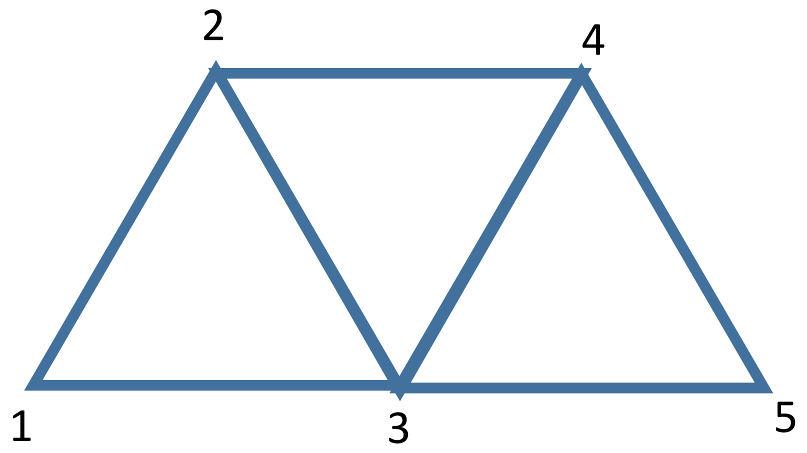
```
glBegin(GL_TRIANGLE_STRIP);
    glVertex3f(0,0,0);
    glVertex3f(1,1,0);
    glVertex3f(2,0,0);
    glVertex3f(3,1,0);
    glVertex3f(4,0,0);
glEnd();
```



OpenGL Primitives (Triangle Strip)

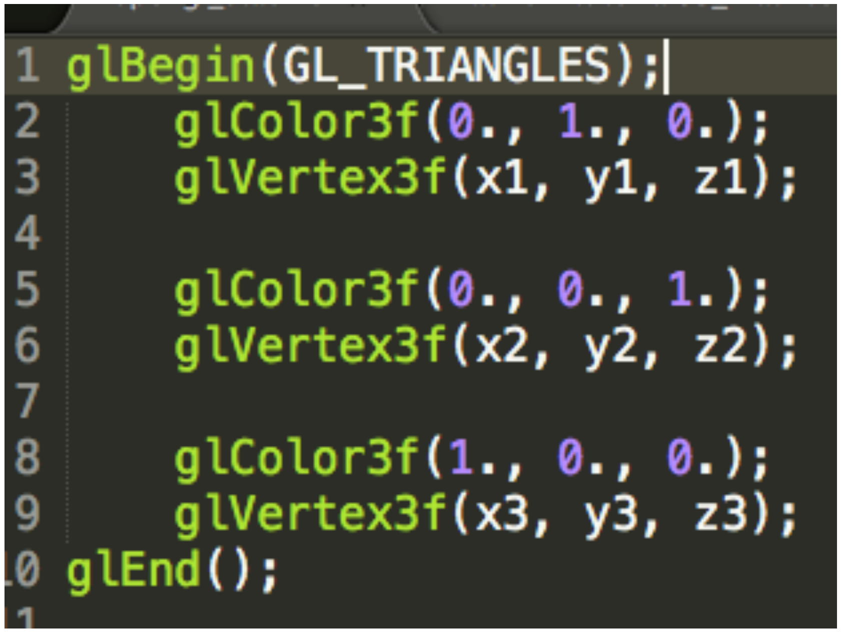


```
glBegin(GL_TRIANGLE_STRIP);
    glVertex3f(0,0,0);
    glVertex3f(1,1,0);
    glVertex3f(2,0,0);
    glVertex3f(3,1,0);
    glVertex3f(4,0,0);
glEnd();
```

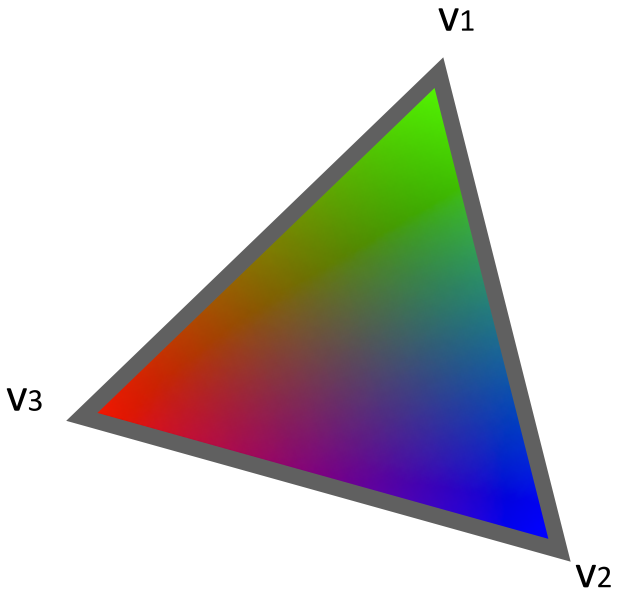


Immediate Mode (Deprecated)

Color: glColor3f(red,green,blue)



```
glBegin(GL_TRIANGLES);
    glColor3f(0., 1., 0.);
    glVertex3f(x1, y1, z1);

    glColor3f(0., 0., 1.);
    glVertex3f(x2, y2, z2);

    glColor3f(1., 0., 0.);
    glVertex3f(x3, y3, z3);
glEnd();
```



GL, GLU, and GLUT notations

gl...

e.g., glColor3f(...)

core OpenGL function

glu...

e.g., gluLookAt(...)

OpenGL utility function, makes common tasks easier

(defined in terms of gl... functions)

glut...

e.g., glutSolidTeapot(...)

GLUT functions

GL, GLU, and GLUT notations

glVertex3f(...)

...3f	takes 3 floats
...3d	takes 3 doubles
...3i	takes 3 integers
...2f	takes 2 floats
...4f	takes 4 floats
	(etc)

Composing Transformations

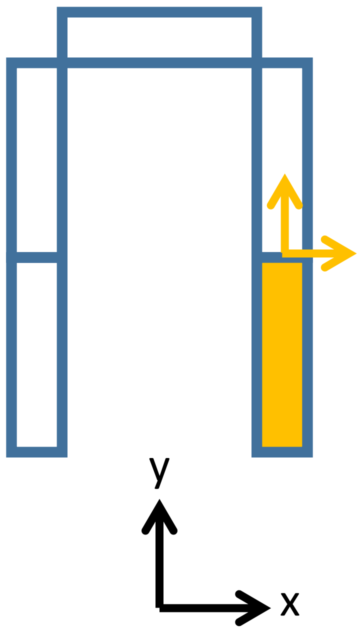
```
glLoadMatrixf(A);
glMultMatrixf(B);
glMultMatrixf(C);
```

$$T_{new} = A \cdot B \cdot C$$

Convenience Functions

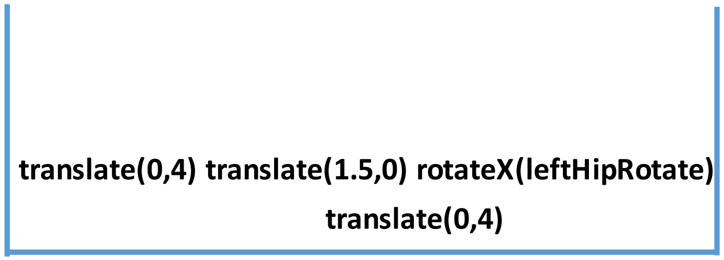
- `glTranslatef(tx, ty, tz)`
- `glRotatef(degrees, x, y, z)`
- `glScalef(sx, sy, sz)`

Hierarchical Modeling



```
translate(0,4)
drawTorso()
pushMatrix()
    translate(1.5,0)
    rotateX(leftHipRotate)
    drawThigh()
    pushMatrix()
        translate(0,-2)
        rotateX(leftKneeRotate)
        drawLeg()
         ...
    popMatrix()
popMatrix()
pushMatrix()
    translate(-1.5,0)
    rotateX(rightHipRotate)
    // Draw the right side
    ...
...
```

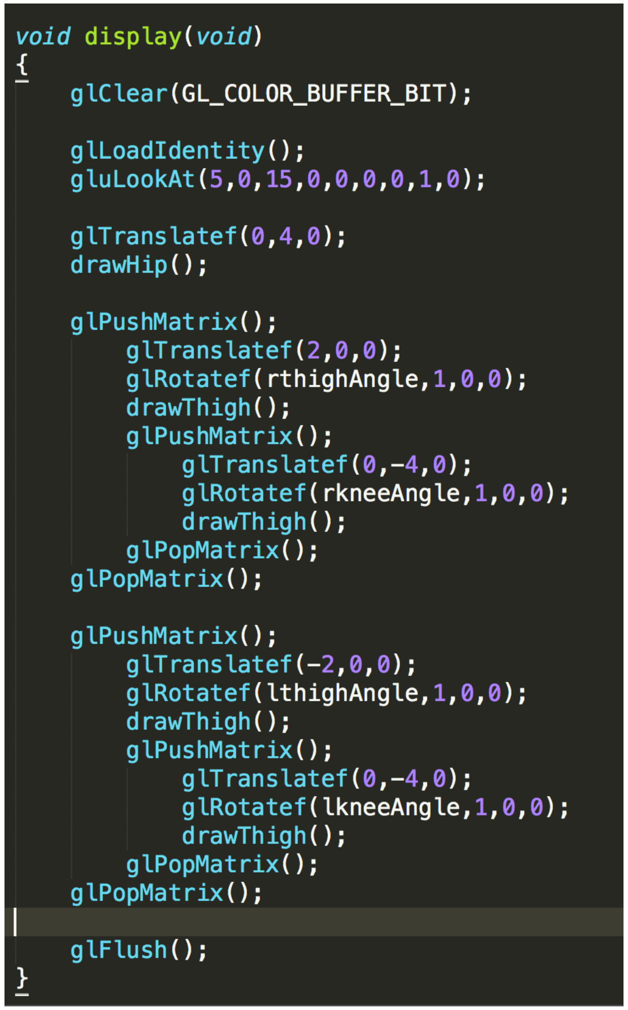
Recall !



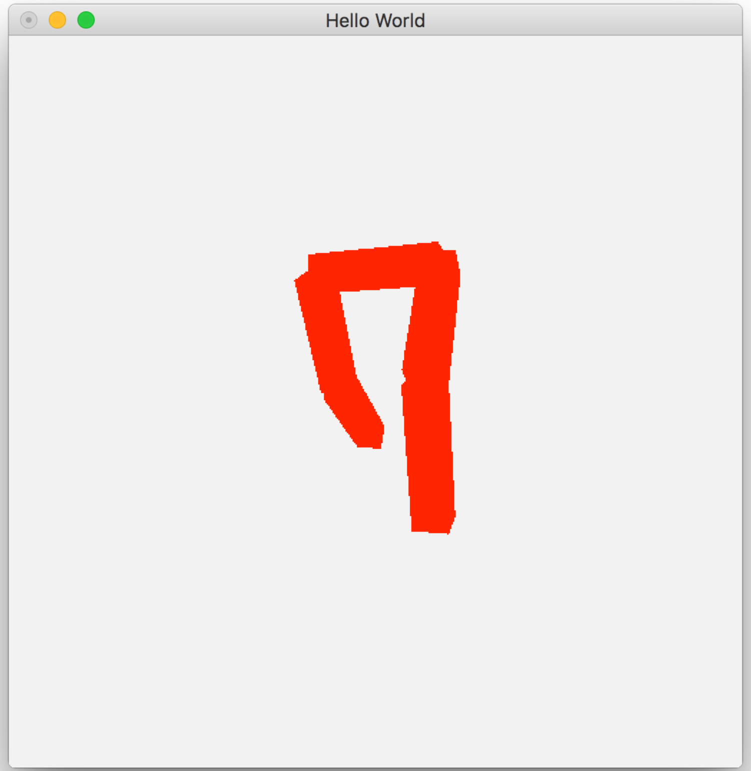
Matrix Stack

CurrentMatrix = translate(0,4) translate(1.5,0) rotateX(leftHipRotate) translate(0,-2) rotate(leftKneeRotate)

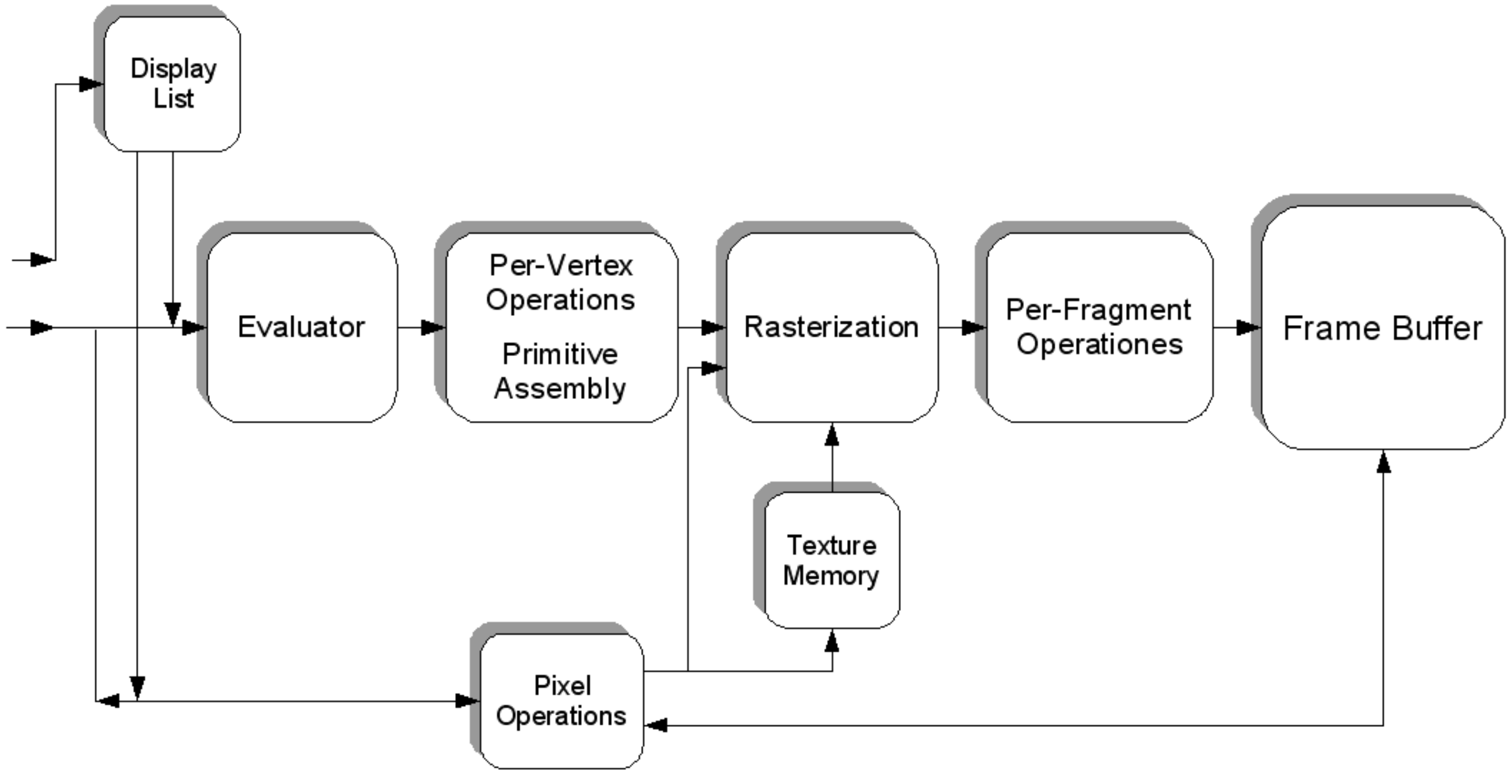
OpenGL Matrix Stack



```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glLoadIdentity();
    gluLookAt(5,0,15,0,0,0,0,1,0);

    glTranslatef(0,4,0);
    drawHip();

    glPushMatrix();
        glTranslatef(2,0,0);
        glRotatef(rthighAngle,1,0,0);
        drawThigh();
        glPushMatrix();
            glTranslatef(0,-4,0);
            glRotatef(rkneeAngle,1,0,0);
            drawThigh();
        glPopMatrix();
    glPopMatrix();

    glPushMatrix();
        glTranslatef(-2,0,0);
        glRotatef(lthighAngle,1,0,0);
        drawThigh();
        glPushMatrix();
            glTranslatef(0,-4,0);
            glRotatef(lkneeAngle,1,0,0);
            drawThigh();
        glPopMatrix();
    glPopMatrix();

    glFlush();
}
```

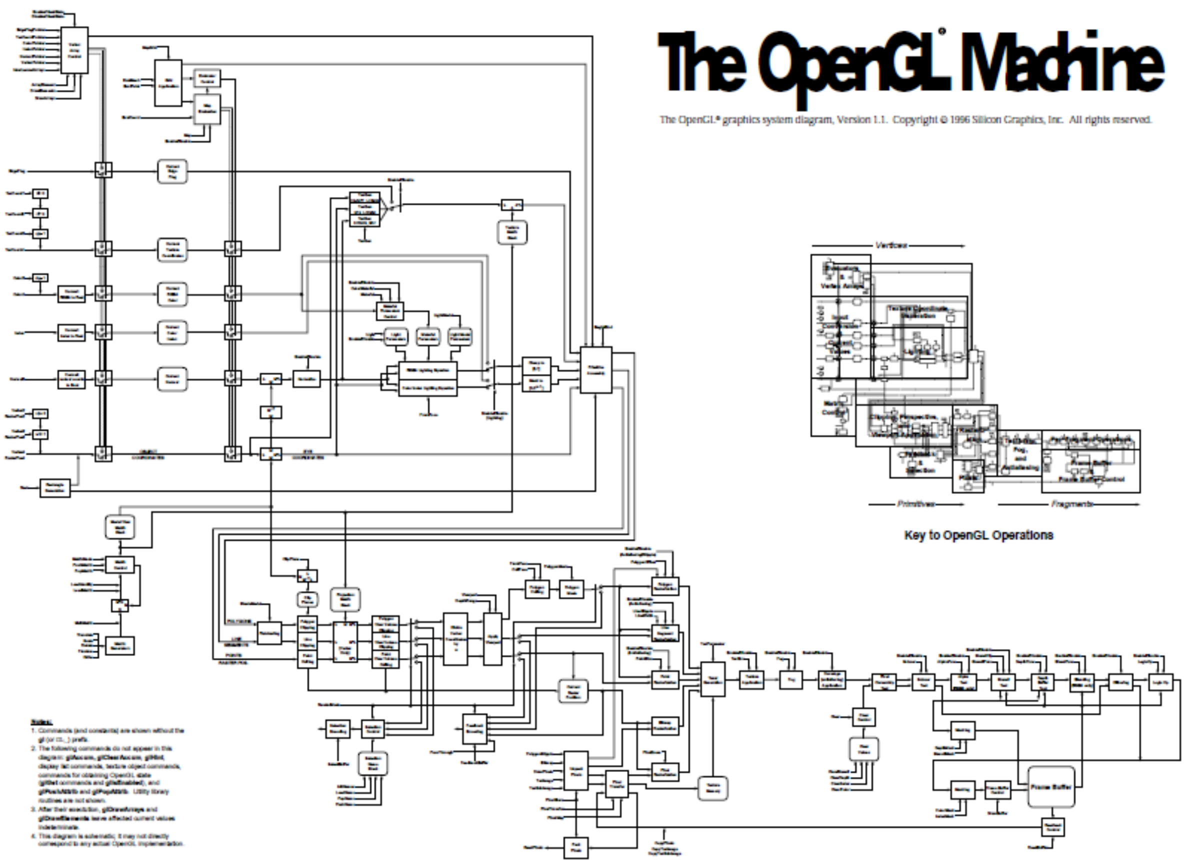


OpenGL 1.x Pipeline (Simplified)



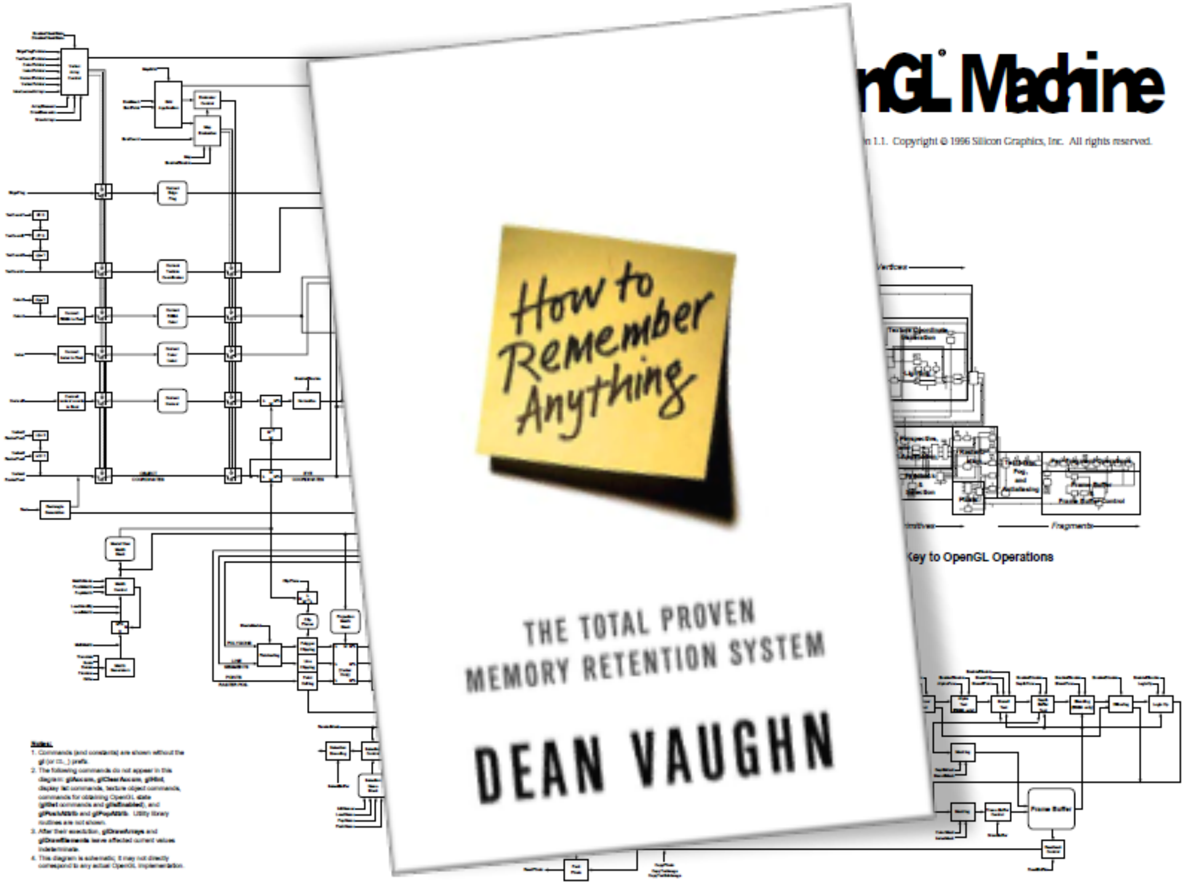
http://upload.wikimedia.org/wikipedia/commons/b/bb/Pipeline_OpenGL_%28en%29.png

OpenGL 1.x Pipeline (Unsimplified)


The OpenGL Machine
The OpenGL® graphics system diagram, Version 1.1. Copyright © 1996 Silicon Graphics, Inc. All rights reserved.
Key to OpenGL Operations
Vertices
Primitives
Fragments

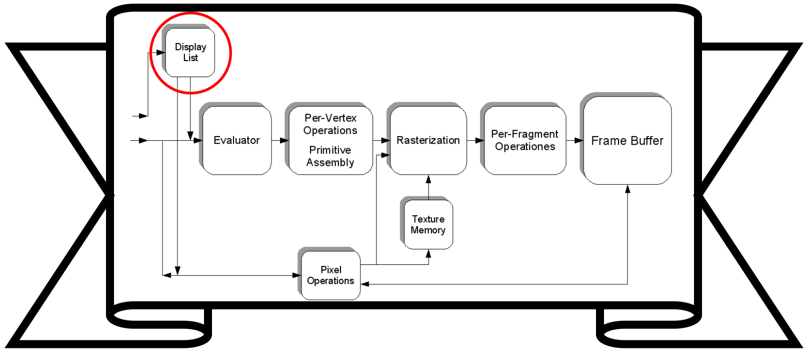
http://www.opengl.org/documentation/specs/version1.1/state.pdf

OpenGL 1.x Pipeline (Unsimplified)

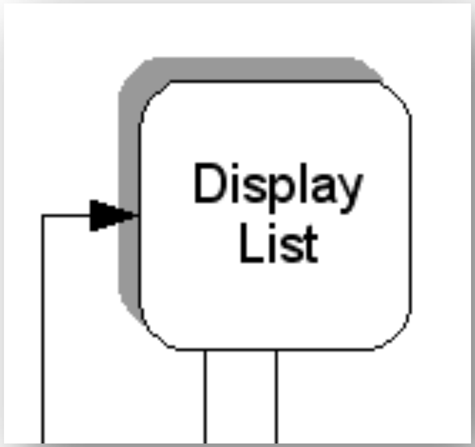

nGL Machine
n 1.1. Copyright © 1996 Silicon Graphics, Inc. All rights reserved.
Key to OpenGL Operations
How to Remember Anything
THE TOTAL PROVEN MEMORY RETENTION SYSTEM
DEAN VAUGHN

http://www.opengl.org/documentation/specs/version1.1/state.pdf

Pieces of the Pipeline

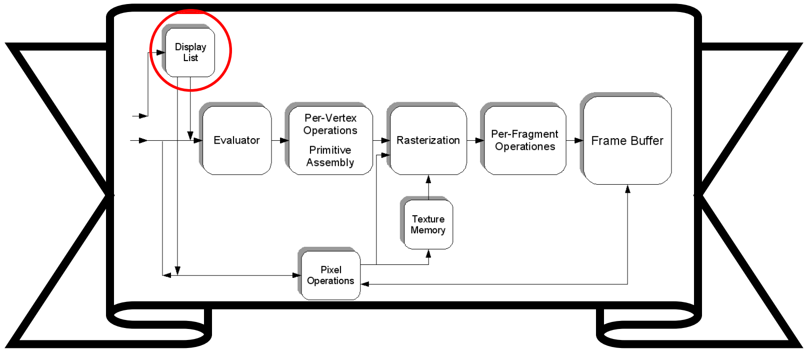


Stores "subroutines"

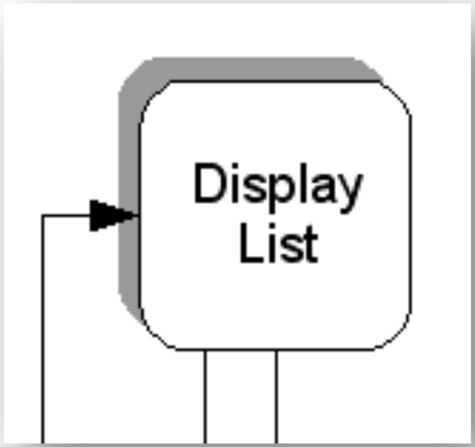


```
GLuint boxList;
boxList = glGenLists(1);
glNewList(boxList, GL_COMPILE);
   // draw box
glEndList(boxList);
…
glCallList(boxList);
```

Pieces of the Pipeline



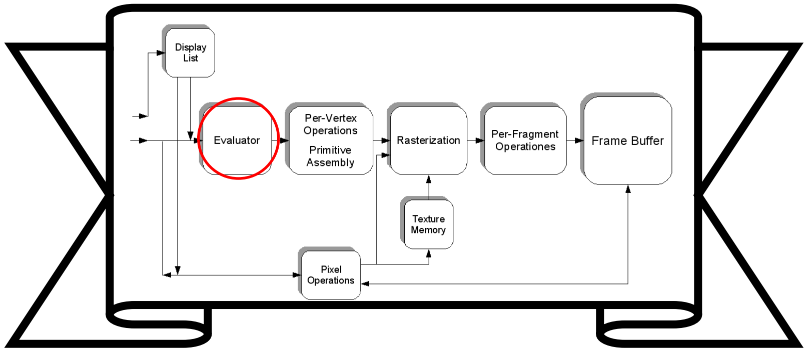
Stores "subroutines"



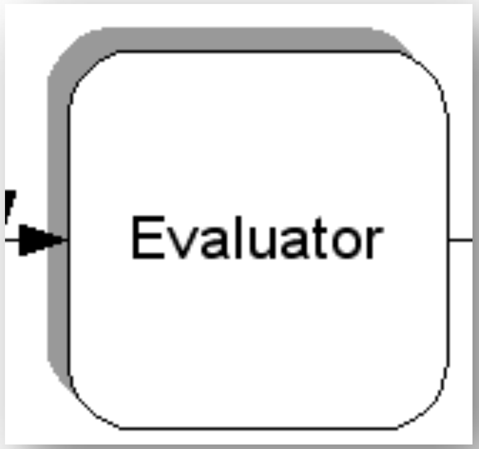
Faster !

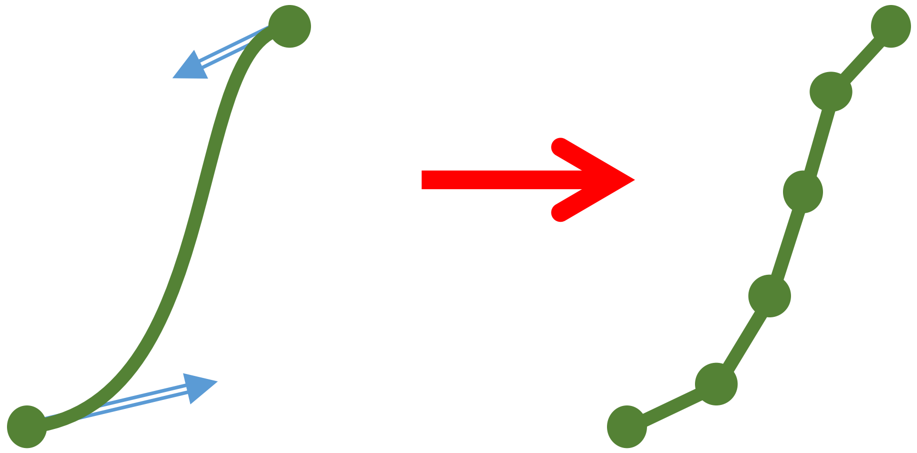
- Pre-compiled
- Store on GPU
- Pre-compute transformation
  - (sometimes)

Pieces of the Pipeline

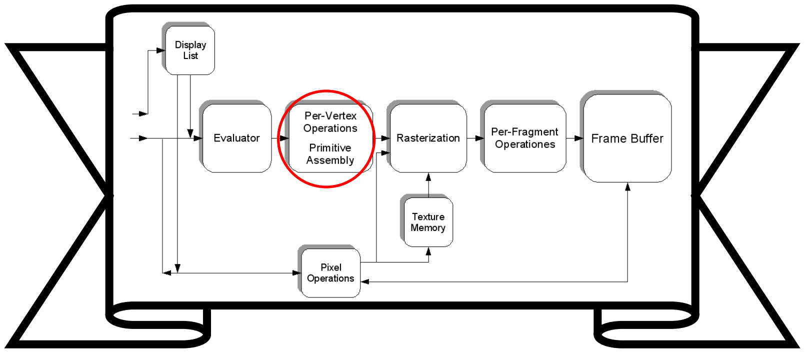


Construct geometric objects

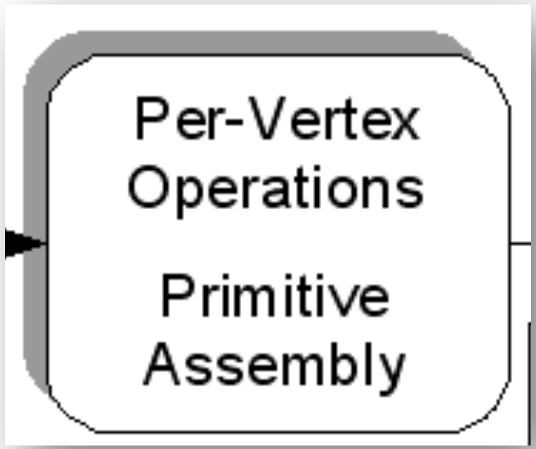




Pieces of the Pipeline

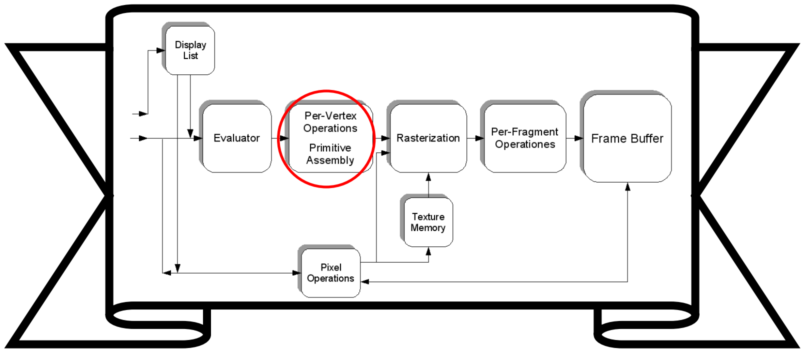


Change geometry

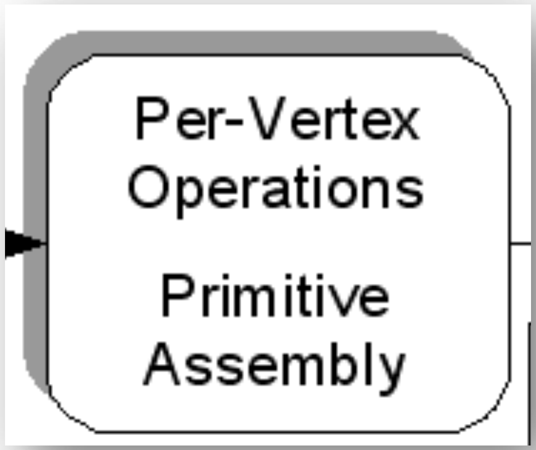


Store primitive shapes

Pieces of the Pipeline



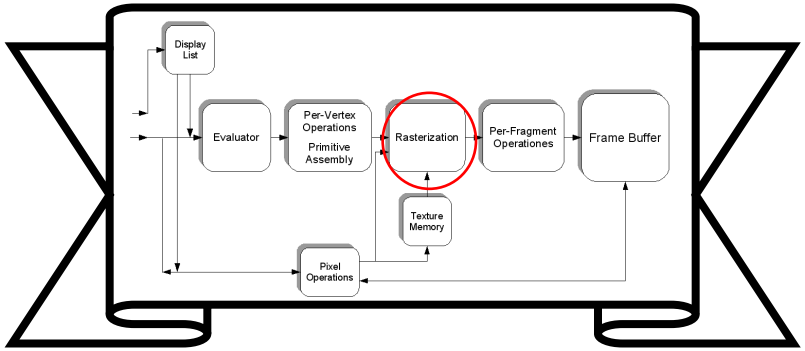
Change geometry

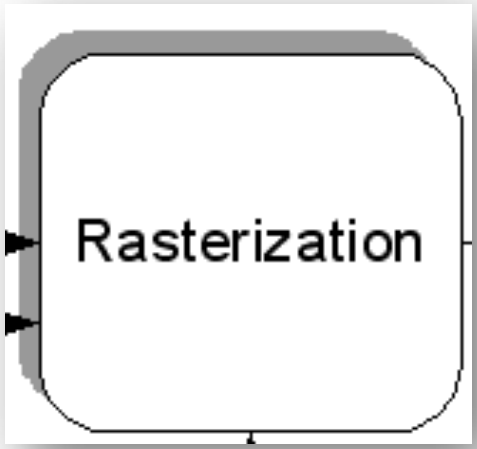


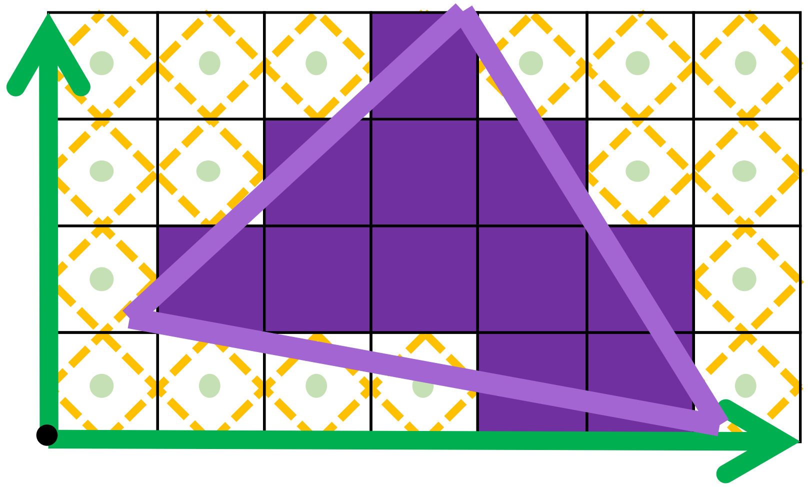
Store primitive shapes

Includes Clipping

Pieces of the Pipeline



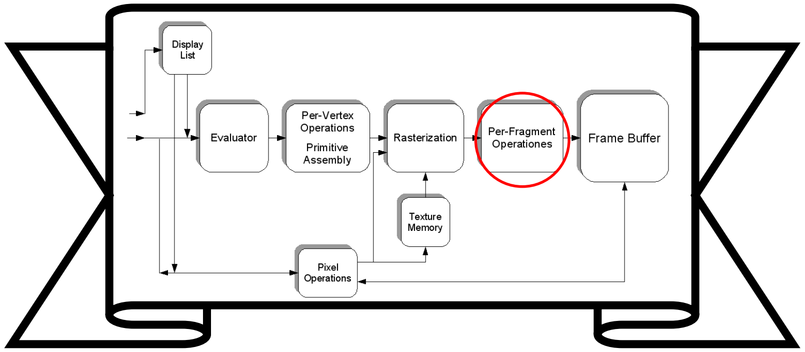




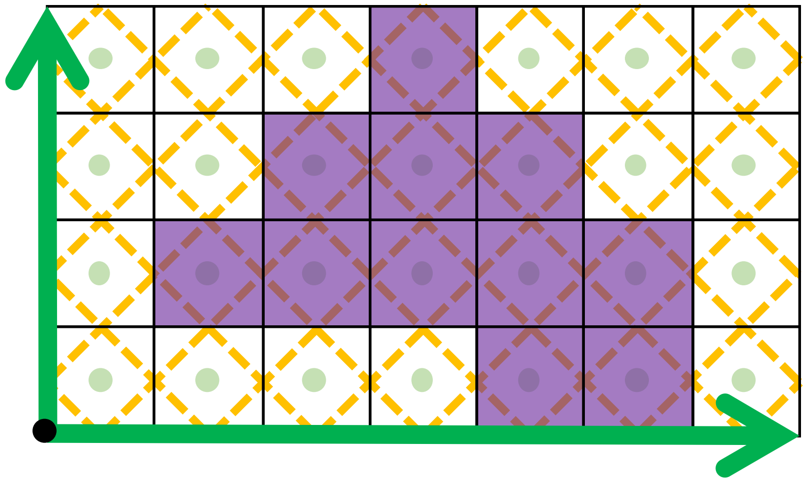
Fragment [frag-muhnt]:

The data necessary to generate a single pixel's worth of a primitive.

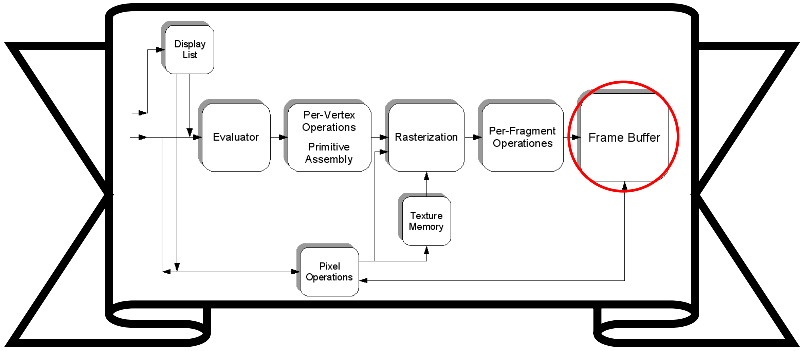
Pieces of the Pipeline



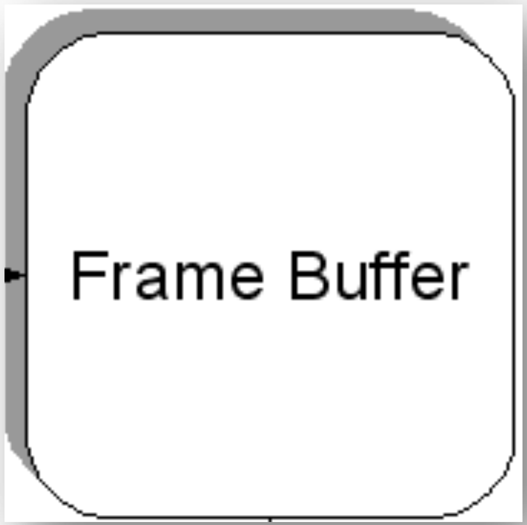
Modify and combine per-pixel information

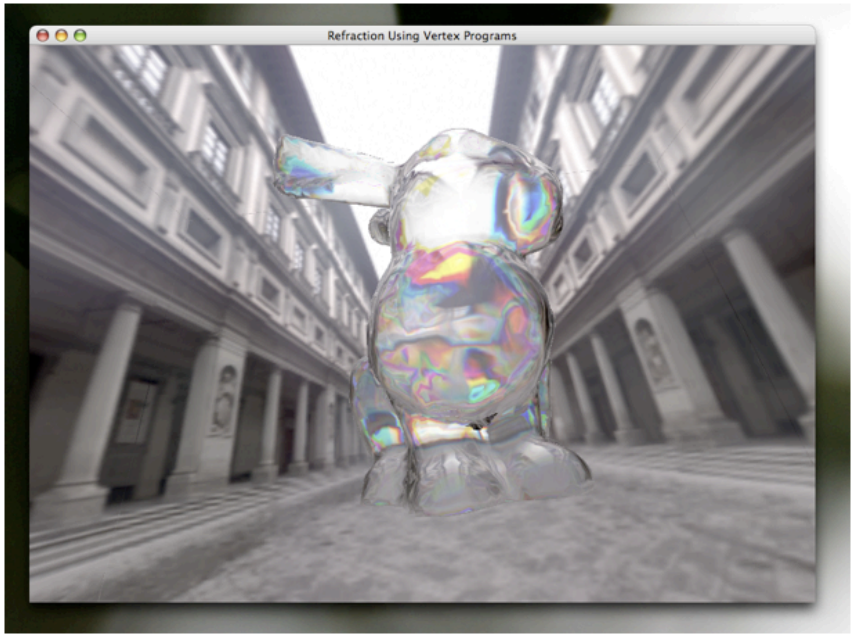


Pieces of the Pipeline



Prepare image to be displayed





OpenGL State Machine

Change State

Draw

Draw

Change State

Draw

Change State

Change State

Draw

OpenGL State Machine

Set State	Get State
`glColor3f(…)`	`glGetFloatv(…)`
`glEnable(…)`	`glIsEnabled(…)`
`glLineStipple(…)`	`glGetLineStipple(…)`

OpenGL State Machine

Set State	Get State
`glColor3f(…)`	`glGetFloatv(…)`
`glEnable(…)`	`glIsEnabled(…)`
`glLineStipple(…)`	`glGetLineStipple(…)`

Efficiently managing state changes is a major implementation challenge

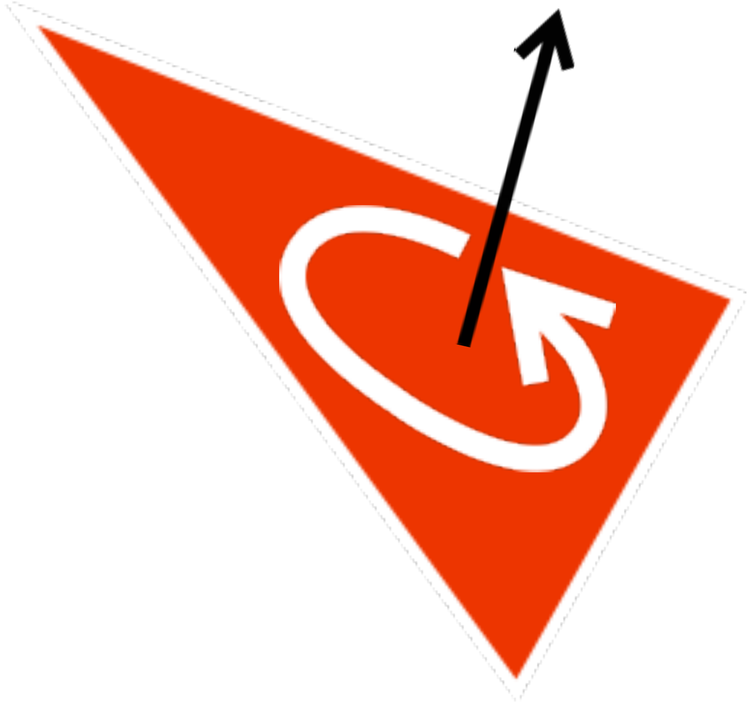
WARNING
OUT OF DATE.

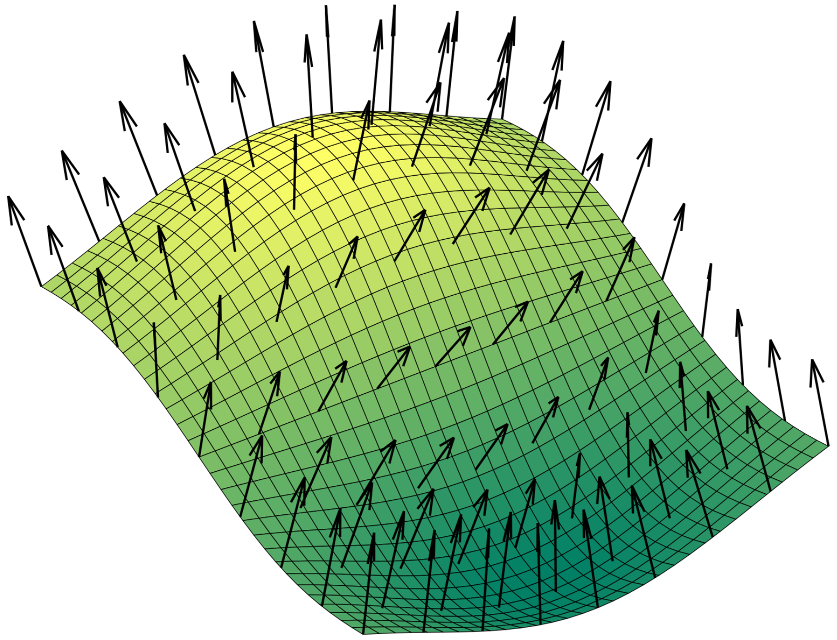
OpenGL >= 2.x later

Vertex Lighting

Normal

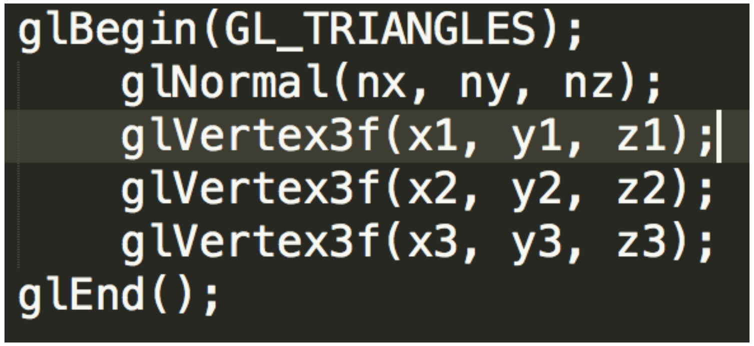
A vector perpendicular to a surface; constant over a plane



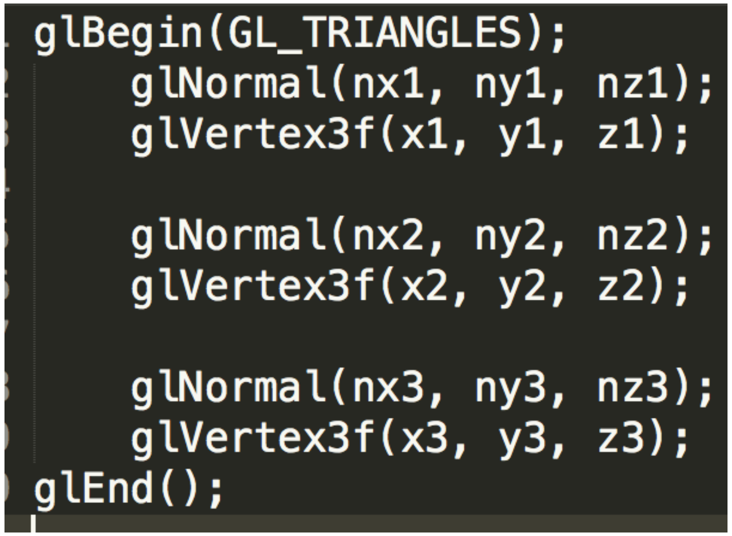


https://en.wikipedia.org/wiki/Normal_(geometry)

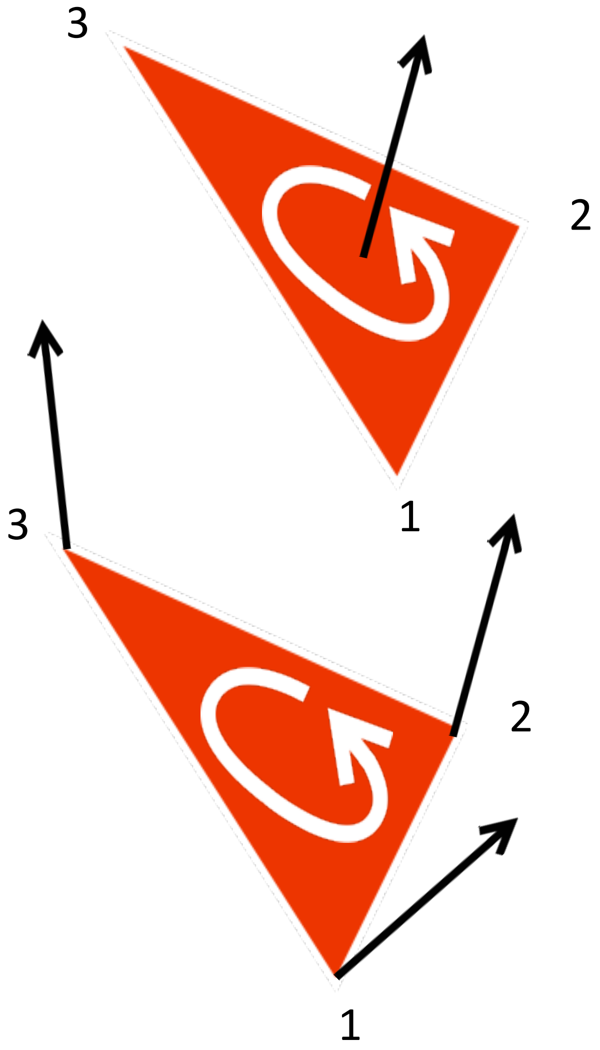
Specifying Normals



```
glBegin(GL_TRIANGLES);
    glNormal(nx, ny, nz);
    glVertex3f(x1, y1, z1);
    glVertex3f(x2, y2, z2);
    glVertex3f(x3, y3, z3);
glEnd();
```

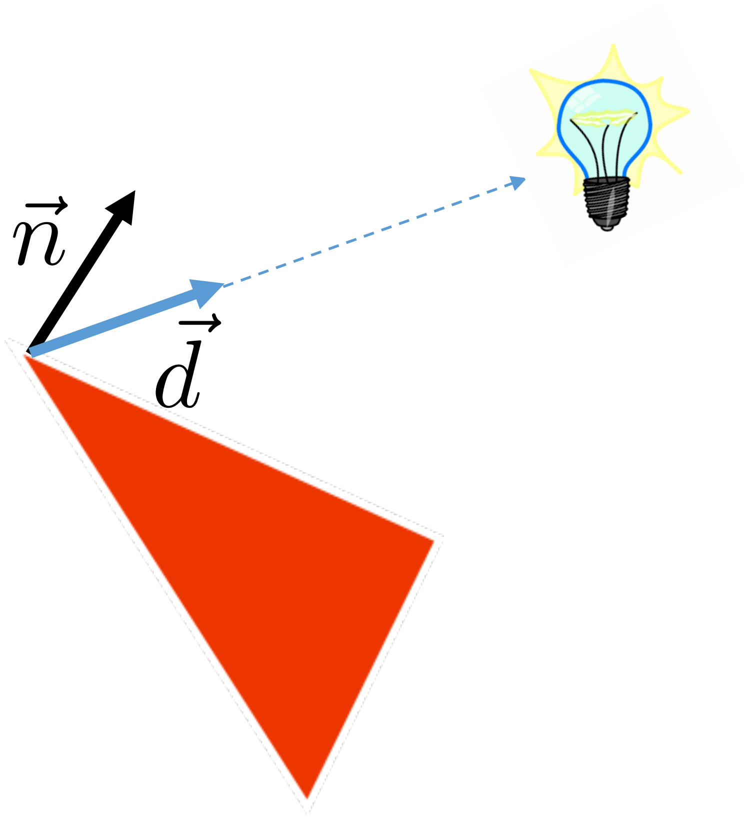


```
glBegin(GL_TRIANGLES);
    glNormal(nx1, ny1, nz1);
    glVertex3f(x1, y1, z1);

    glNormal(nx2, ny2, nz2);
    glVertex3f(x2, y2, z2);

    glNormal(nx3, ny3, nz3);
    glVertex3f(x3, y3, z3);
glEnd();
```



Vertex Lighting

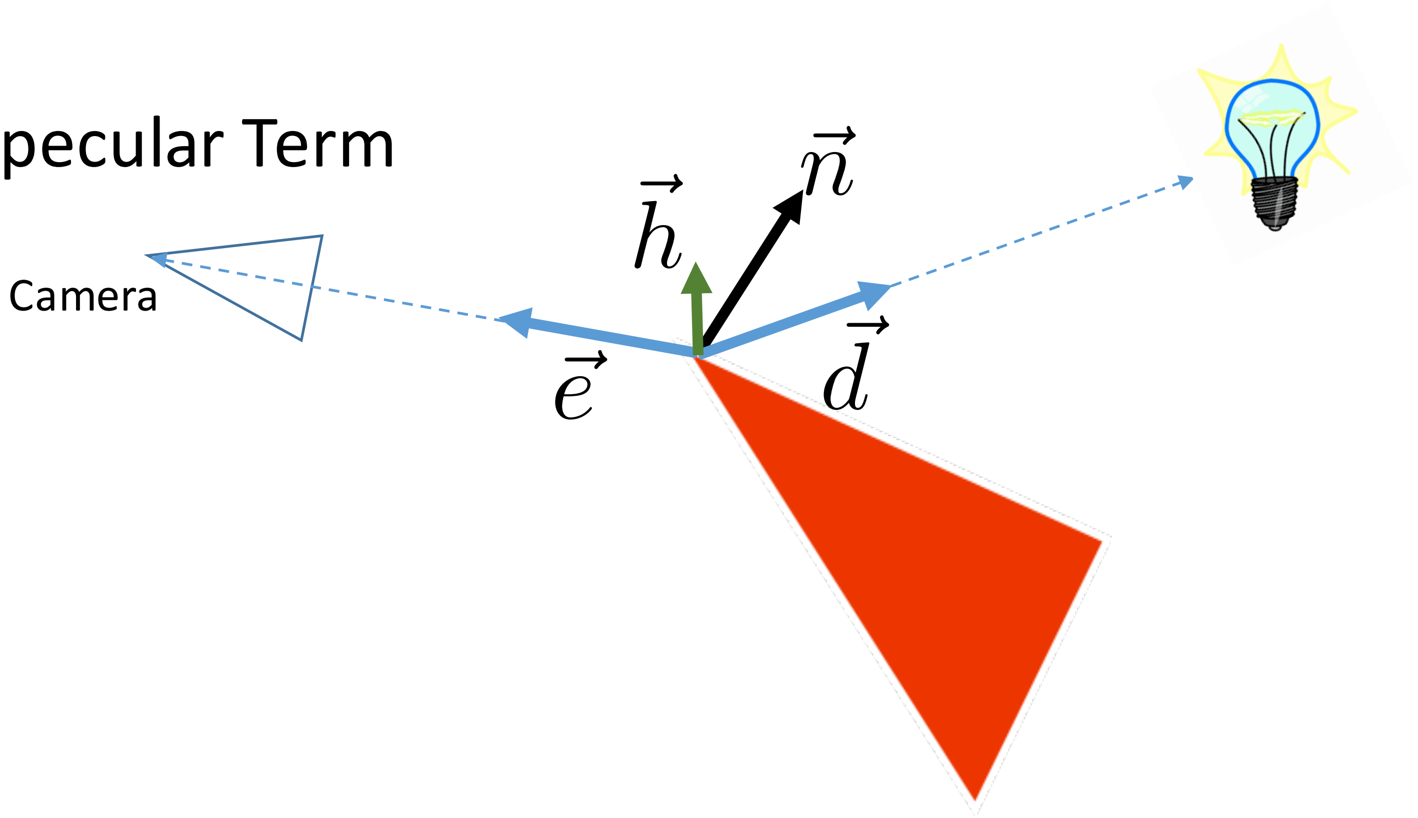
- Diffuse Term



$$\text{diffuseFactor} = \max(\vec{n} \cdot \vec{d}, 0)$$

Vertex Lighting

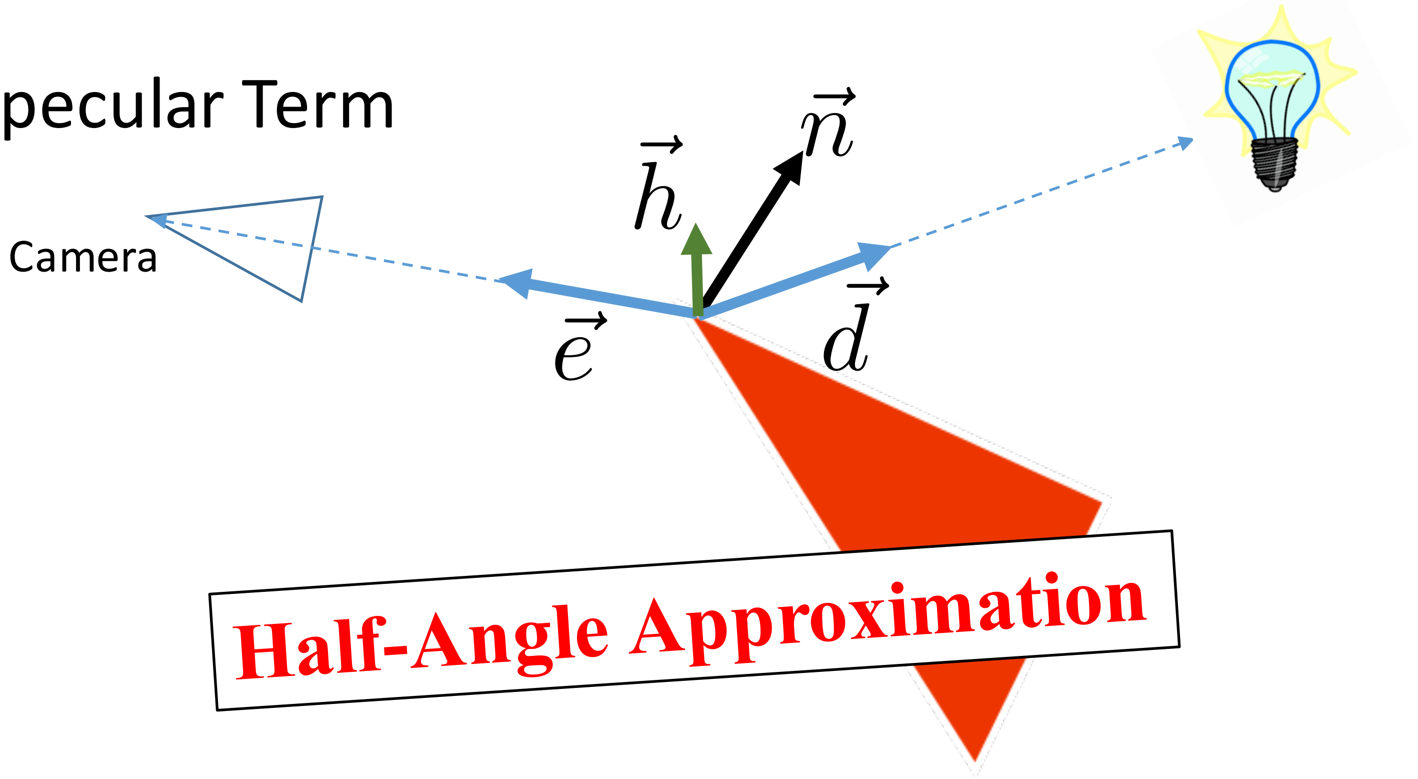
- Specular Term



$$\text{specularFactor} = \left(\vec{h} \cdot \vec{n}\right)^{\text{shininess}}, \quad \vec{h} = \frac{\vec{e} + \vec{d}}{2}$$

Vertex Lighting

- Specular Term

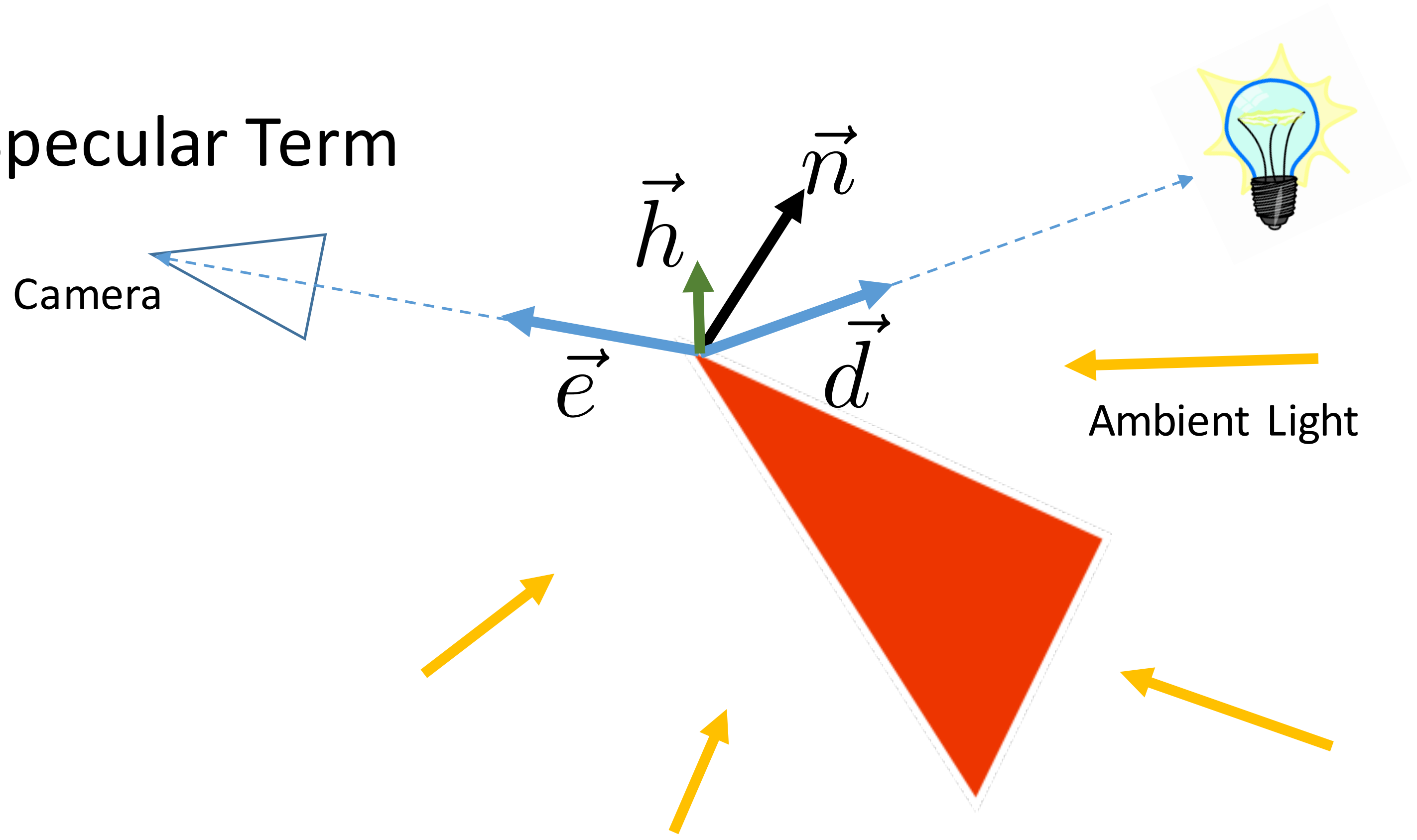


Half-Angle Approximation

$$\text{specularFactor} = \left(\vec{h} \cdot \vec{n}\right)^{\text{shininess}}, \quad \vec{h} = \frac{\vec{e} + \vec{d}}{2}$$

Vertex Lighting

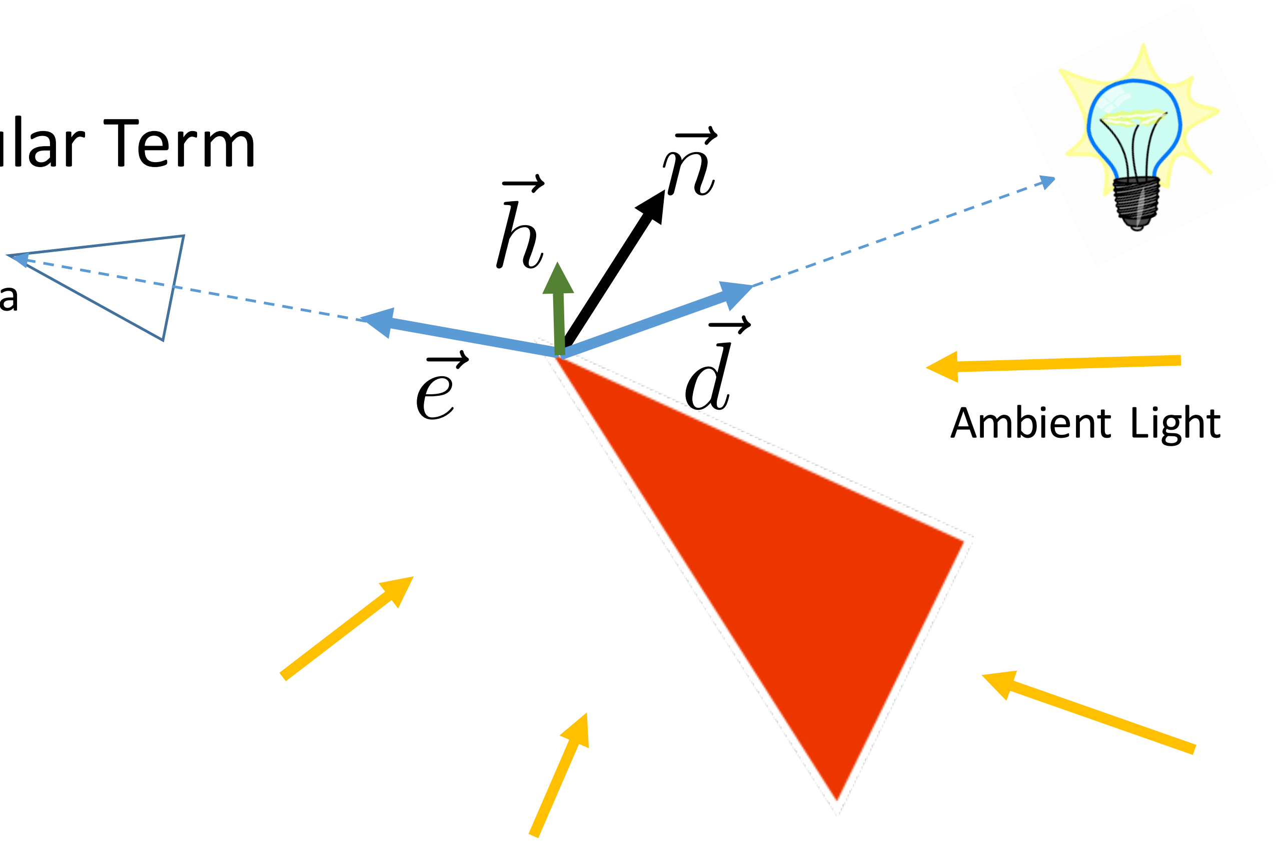
- Specular Term



$$\begin{aligned}\text{Color} = &(\text{DiffuseFactor} \cdot \text{DiffuseColor}) + \\ &(\text{SpecularFactor} \cdot \text{SpecularColor}) + \\ &\text{AmbientColor}\end{aligned}$$

Vertex Lighting

- Specular Term



$$\begin{aligned}\text{Color} = &(\text{DiffuseFactor} \cdot \text{DiffuseColor}) + \\ &\boxed{([\text{DiffuseFactor} > 0]} \cdot \text{SpecularFactor} \cdot \text{SpecularColor}) + \\ &\text{AmbientColor}\end{aligned}$$

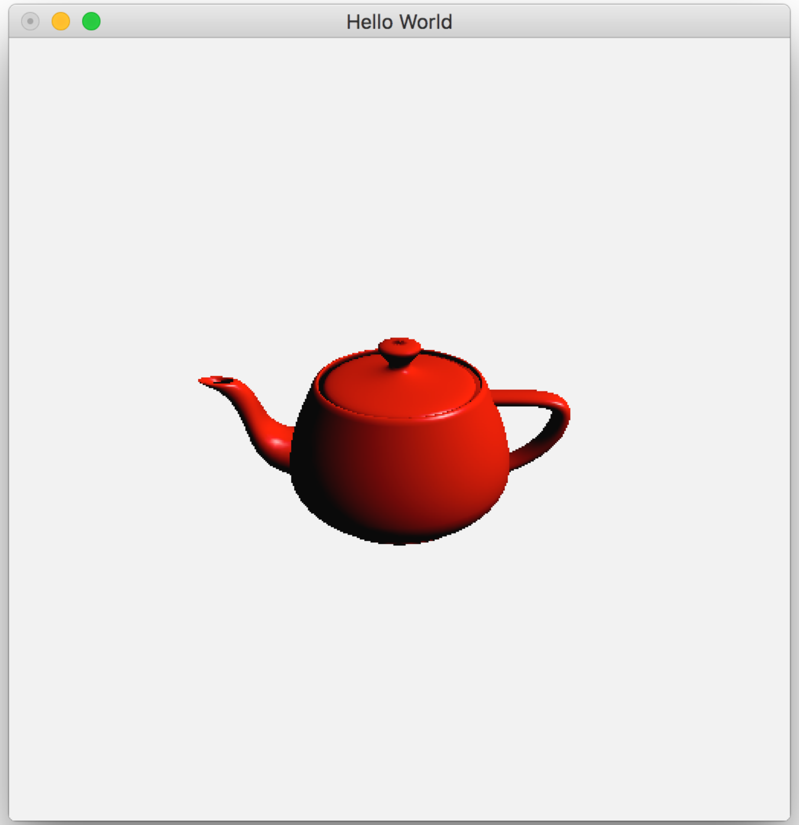
Vertex Lighting

Enable a Light

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

float lightDir[] = {1,1,1,0};
glLightfv(GL_LIGHT0,GL_POSITION,lightDir);
```

Setup material

```
float color[] = {1,0,0,1};
float specular[] = {1,1,1,1};
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE,  color);
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, specular);
glMateriali(GL_FRONT_AND_BACK, GL_SHININESS, 128);

glutSolidTeapot(1);
```



Vertex Lighting

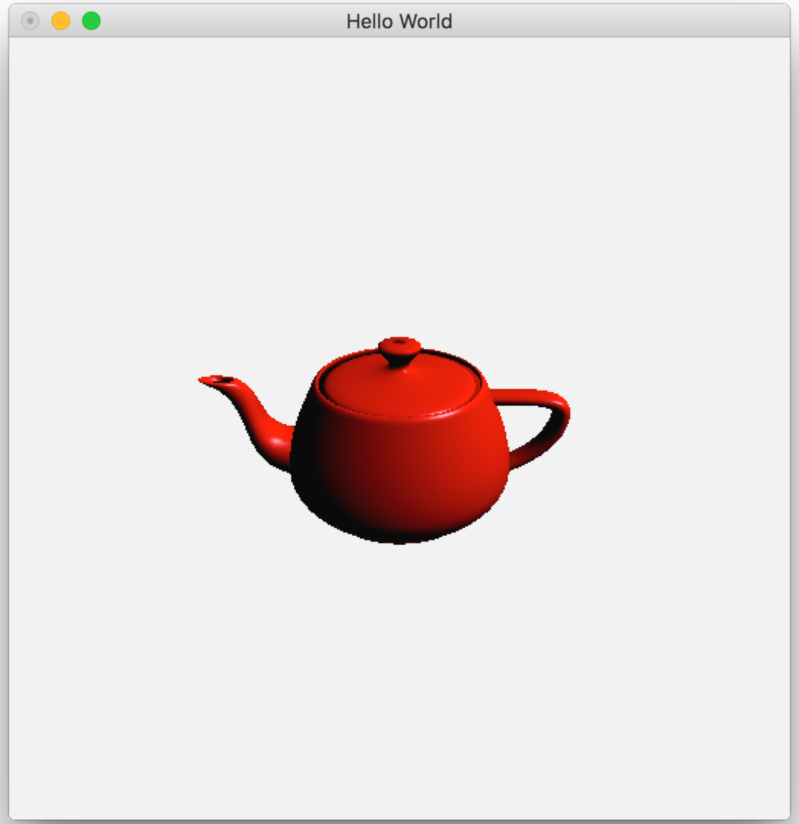
Enable a Light

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

float lightDir[] = {1,1,1,0};
glLightfv(GL_LIGHT0,GL_POSITION,lightDir);
```

Note: w=0 is directional light

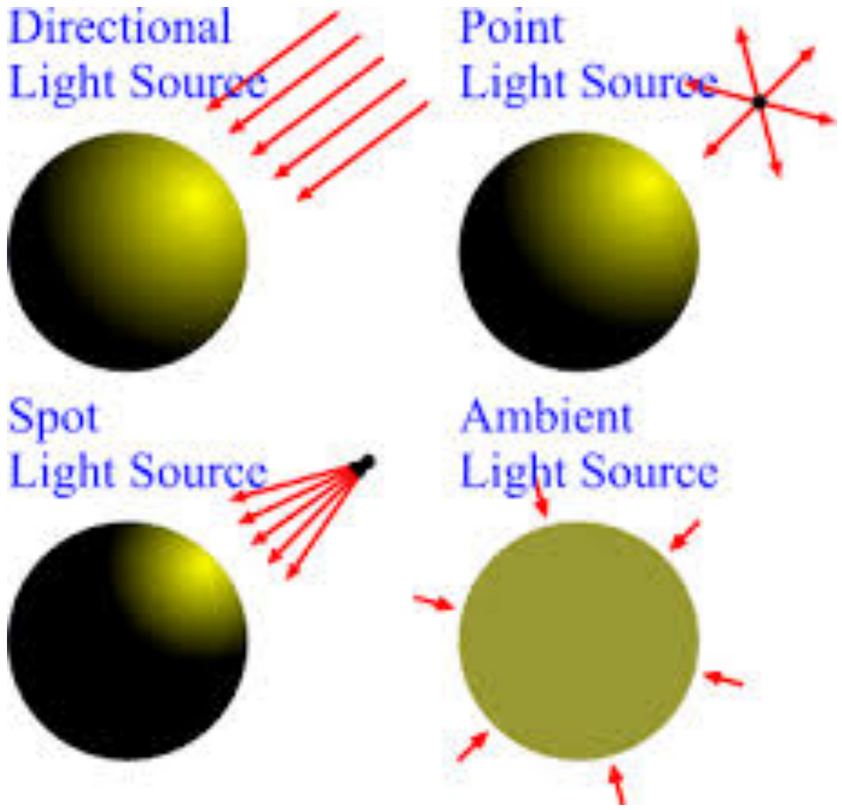
Setup material

```
float color[] = {1,0,0,1};
float specular[] = {1,1,1,1};
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE,  color);
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, specular);
glMateriali(GL_FRONT_AND_BACK, GL_SHININESS, 128);

glutSolidTeapot(1);
```



glutSolidTeapot specifies the normal in this case

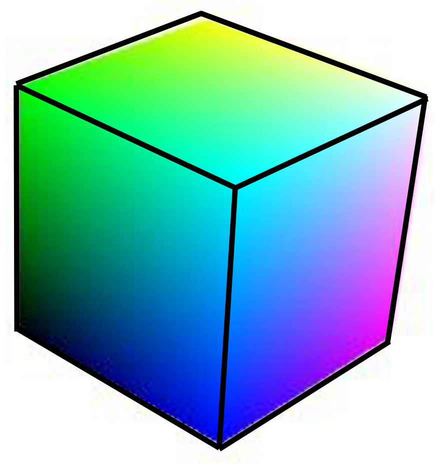
Vertex Lighting: Types



http://www.computing.northampton.ac.uk/~gary/csy3019/images3d/lightSources.gif

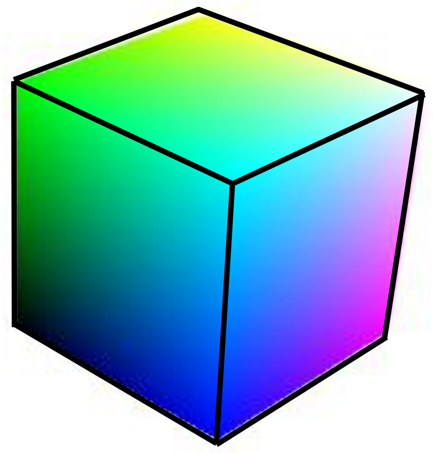
Vertex & Index Buffers

Vertex/Index Buffer



8 vertices only

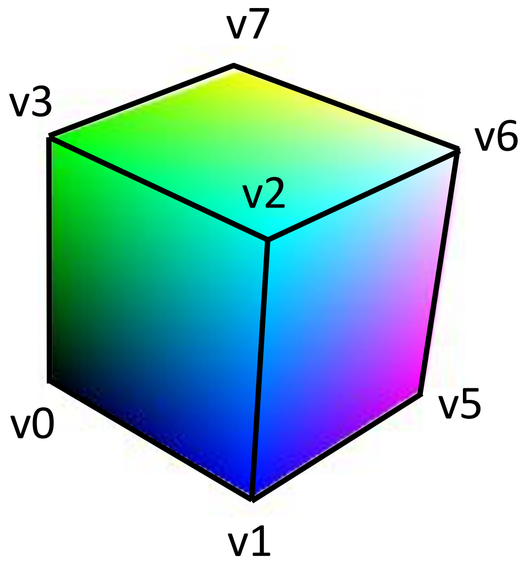
Vertex/Index Buffer



8 vertices only

Drawing with triangles: 36 vertices!

Vertex/Index Buffer

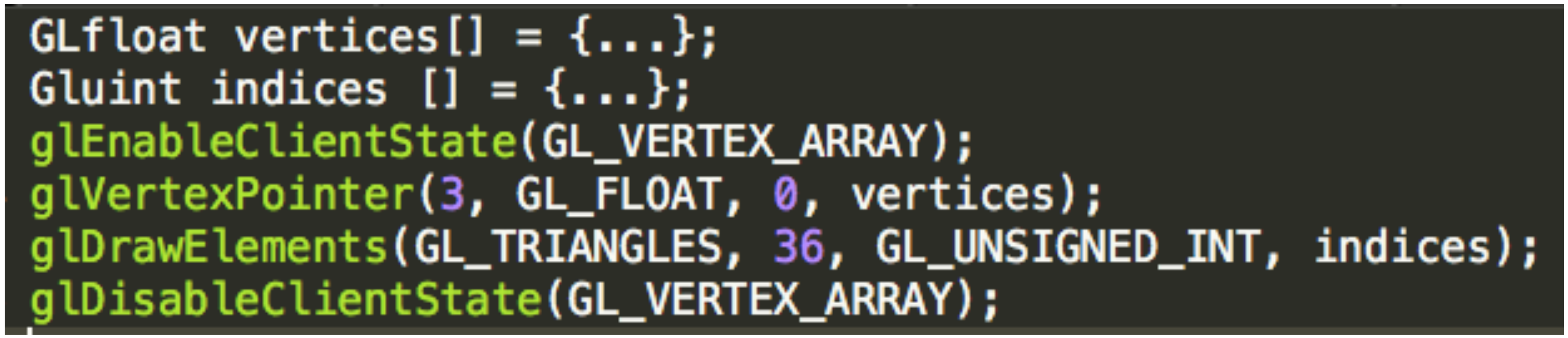


Vertices = { v0,v1,.. V7 }

Indices = { 0,1,2,
0,2,3,
1,5,6,
1,6,2
……
}

8 vertices only

Vertex/Index Buffer



```
GLfloat vertices[] = {...};
Gluint indices [] = {...};
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, 0, vertices);
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_INT, indices);
glDisableClientState(GL_VERTEX_ARRAY);
```

Next Generation APIs

- Vulkan (Nextgen OpenGL)
- DirectX 12 (Microsoft)
- Metal (Apple)

Next Generation APIs

- Vulkan (Nextgen OpenGL)
- DirectX 12 (Microsoft)
- Metal (Apple)

Lower Driver / CPU Overhead !
Support multi-core CPUs

OpenGL

CS 148: Summer 2016
Introduction of Graphics and Imaging
Zahid Hossain

http://www.pling.org.uk/cs/cgv.html