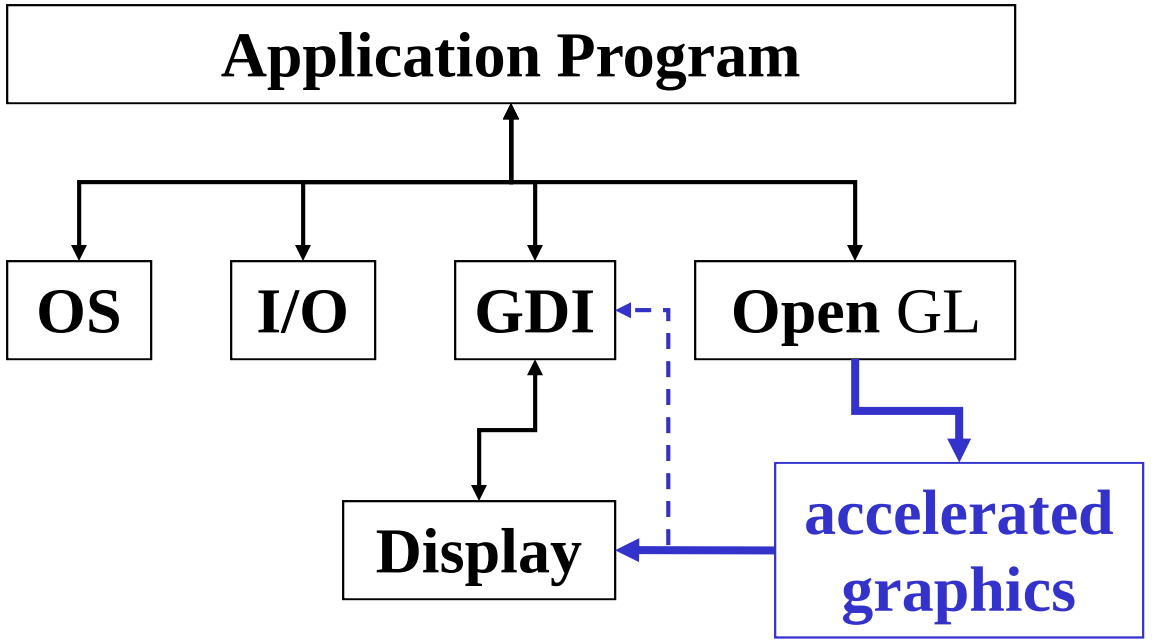
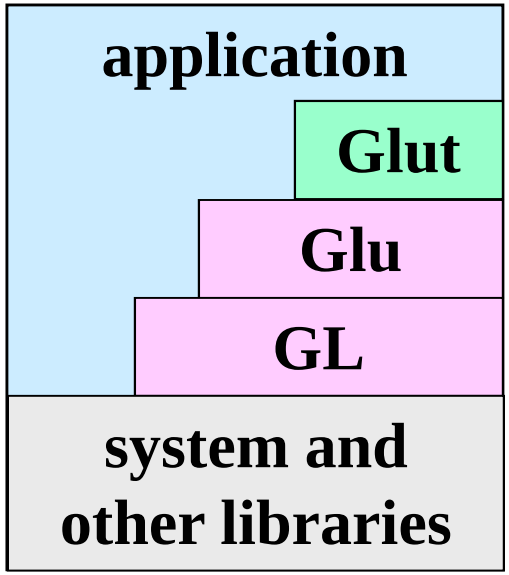
OpenGL

An 3D graphics API.





Utility Toolkit, system independent (3rd party)

Utilities: shapes, matrices, nurbs part of Open GL graphics primities

Graphics Library – primitives

on-line glut manual
http://www.opengl.org/resources/libraries/glut/spec3/spec3.html

API Description

State Machine – Stack architecture

Current state (mode): ModelView, Projection, Selection, …

```
glMatrixMode(GL_Projection)
```

Each state has a stack -- Current matrix is top of that stack

Current attribute: (top of attribute stack) polygon, color,

```
glPopMatrix(…)
glPushAttrib(…)
```

Functions set current state variables: polygon mode, culling,

```
glEnable(GL_CULL_Face)
glCullFace(GL_Back)
```

Function naming convention:

```
// RGBA color
float aColor[4] = {…}
…
glColor4fv(aColor)
```

```
glColor3f(0.0, 1.0, 0.0)
```

library

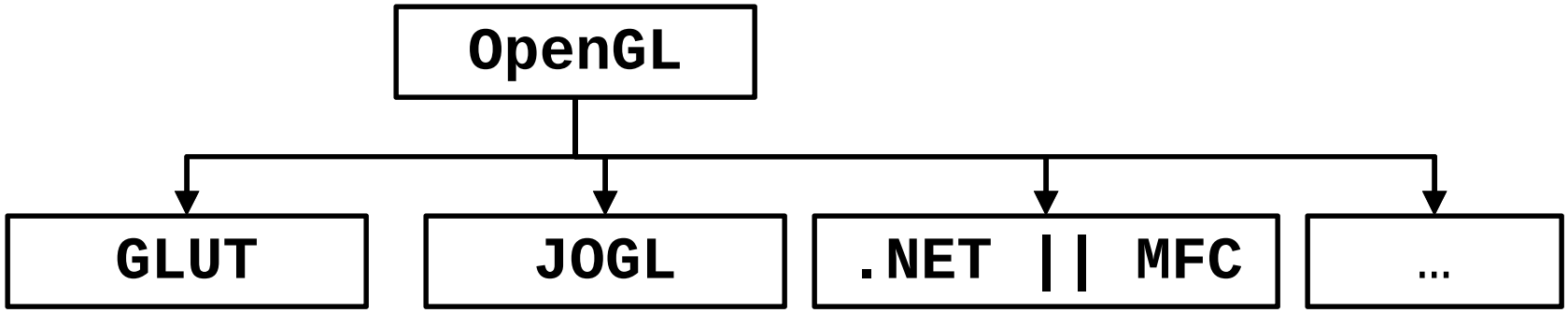
command

agr count

arg type

Programming

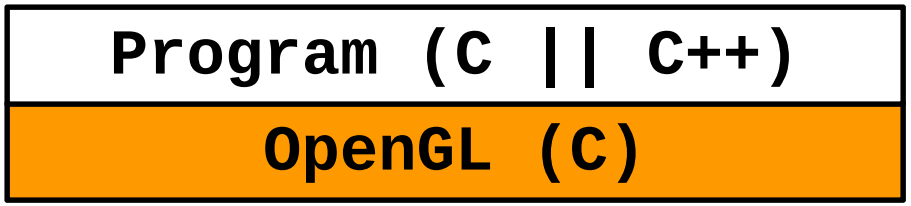
OpenGL and host system window graphics

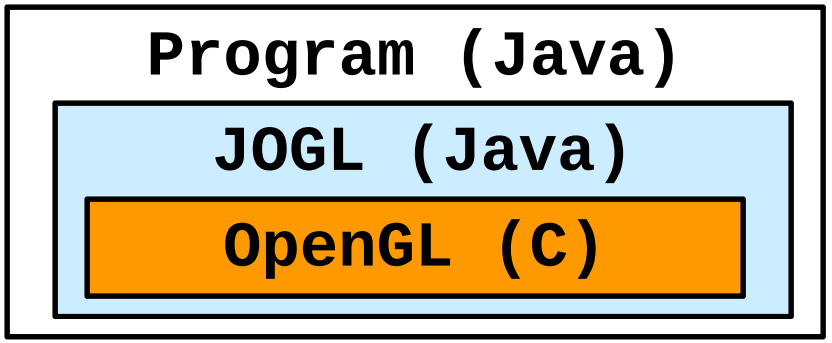


OpenGL and OOP

C++ classes for application objects, functions for OpenGL

Java JOGL lite class wrapper for OpenGL calls





Overview of a C, GLUT, GL program

www.opengl.org

M.J. Kilgard, OpenGL Programming for the X Windows System, Addison Wesley, 1996.

```
# include <GL\glut.h>      // could be <gl/glut.h> on system

void render() {...}        // callback for display fn
void initialize(){...}     // background, clips, ...
void resize() {...}        // callback viewport changes
void callbacks() {...)     // many user callback fns
void miscFns() {...}       // many user fns...

int main(int argc, char ** argv) {
   glutInit(argc, argv);
   glutInitDisplay(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
   glutCreateWindow("title");
   glutReshapeFunc(resize);  // add resize callback
   glutDisplayFunc(render);  // add render callback
   initialize();
   // pass control to GLUT event handling routine
   glutMainLoop();
   }
```

```
void resize(int width, int height)
   glViewport(xOrigin, yOrigin, width, height);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective( FieldOfView, (float) width / (float)
      height, near, far);
   // Have gluLookAt(...) in the render function
   // Consistently sequence view changes, model changes
```

```
void render()
   // set light values
   ...
   // clear color and depth buffers
   glClearColor(1, 1, 1, 1);  // white background
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glEnable(GL_DEPTH_TEST);
   // set up lights and materials
   ...
   // set model view
   glMatixMode(GL_MODELVIEW);
   glLoadIdentity();
   // set up camera viewpoint w/ gluLookAt(....)
   ...
   // do transformation and drawing
   ...
   glutSwapBuffers();
```

Overview of a C++, Glut, GL program

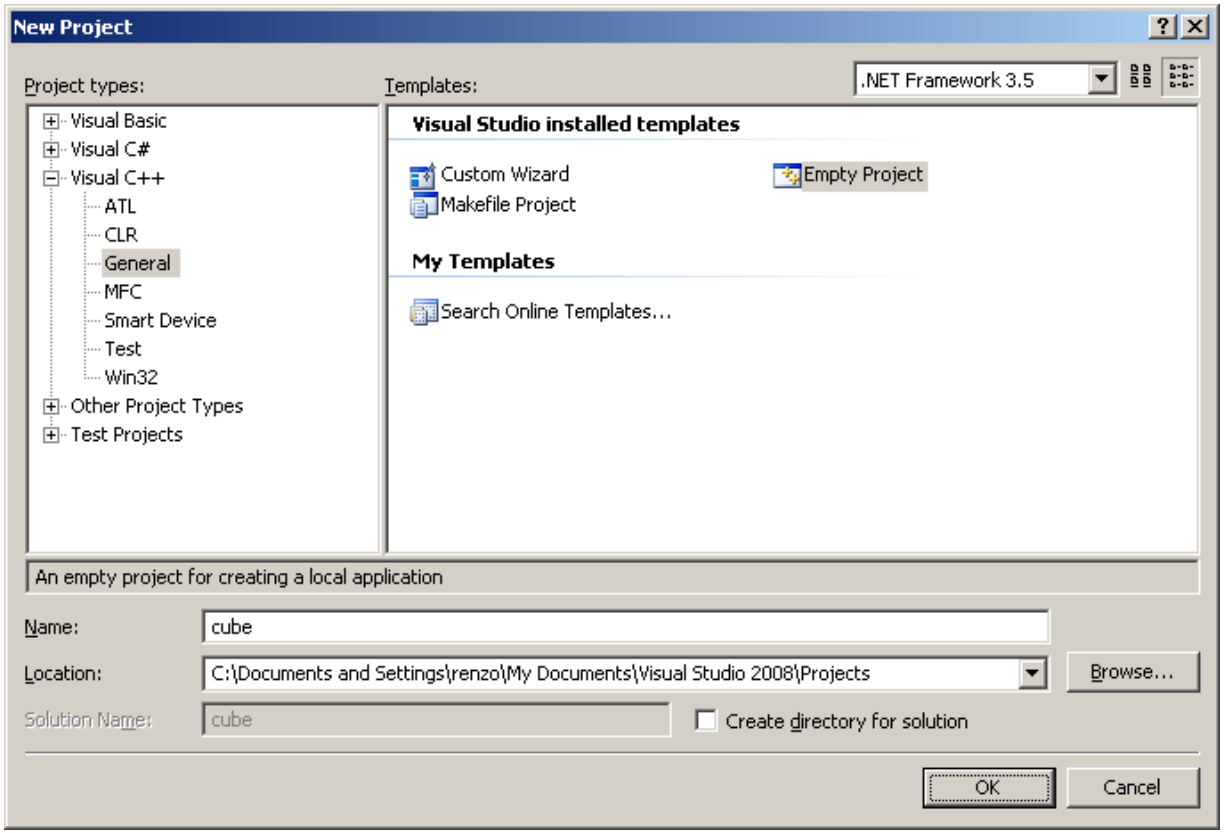
```
// Separate Object Oriented from C / OpenGL

class cube {   ...
   public void draw() {...}
   };

Cube * aCube;  // pointer (reference) to a Cube

public void draw() { ...
   cube -> draw();
   }

// Program entry
int main(int  argc, char *argv[])  {
   aCube = new Cube();   // create objects
   ...  // create glut display and window
   glutDisplayFunc(draw);  // set callback functions
   ...
   glutMainLoop();
   ... }
```

Build Application

Developing Open GL programs w/ Visual Studio .NET 2008

menu select **File | New | Project**
in New Project Dialog
select Visual C++ Projects
select **General Empty Project**
enter **Name:** cube
enter or browse [...] for **Location:**
click **OK** button

<< show example
use cube.cpp >>



Now in the Visual Studio IDE for development.

If you have downloaded a copy of the cube.cpp file from the class page and placed the copy in your project subdirectory

Project | **Add new item** to add a file say, cube.cpp to project

Project | Add Existing item to add class examples – like cube.cpp to your project

Else create an empty file and copy the source into that file or edit

Project | File | **New File...** create a C++ (*.cpp) file

---

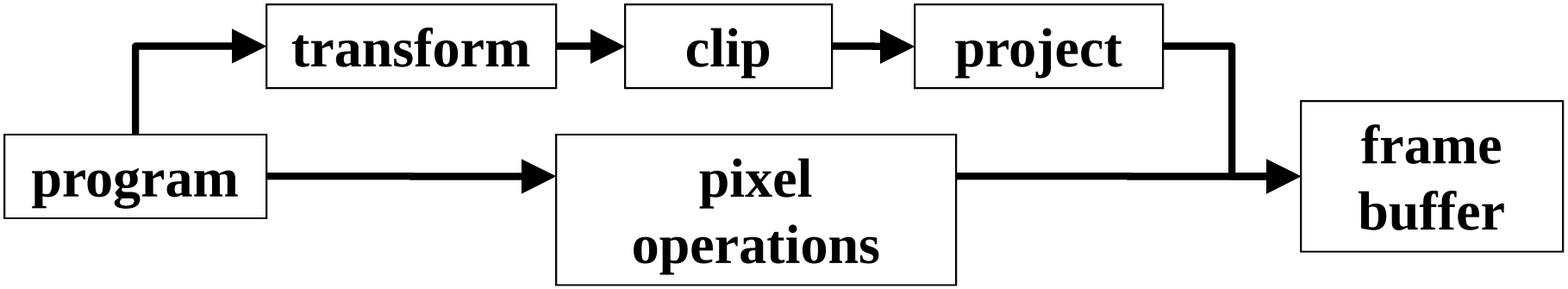
use **Build** | **Build project** or **Rebuild project** to compile

use **Debug** | **Start without debugging** to execute

example Mac Xcode, Linux and Gnu gcc, and Java and JOGL builds, makefiles, look at the posted cube examples.

cube.cpp

Simplified OpenGL pipeline



Walk through cube.cpp for C++

(JOGL OpenGL overview posted on class page)

Include files specified in source – not project (portable)

Makefiles for linux (java ?) approaches posted on class page

GLUT Callbacks

Callbacks are a indirect function calls.
Connect user code with pre-built library routines.

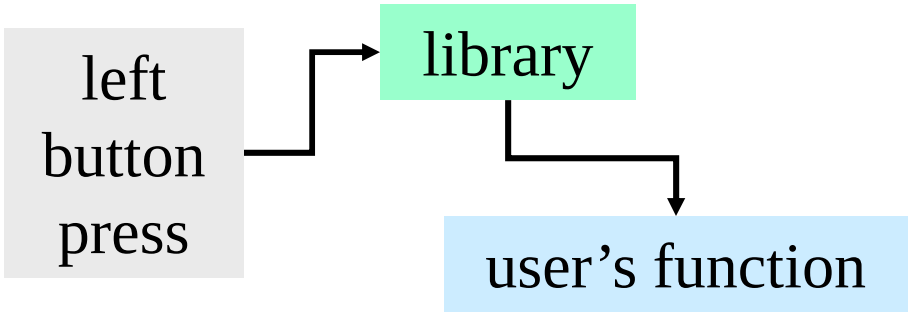
- library has a callback resource (*entry in array*...) for user behavior
- user writes function, “registers / adds” function to callback resource
- executing program can call user function from library function

compiled library code

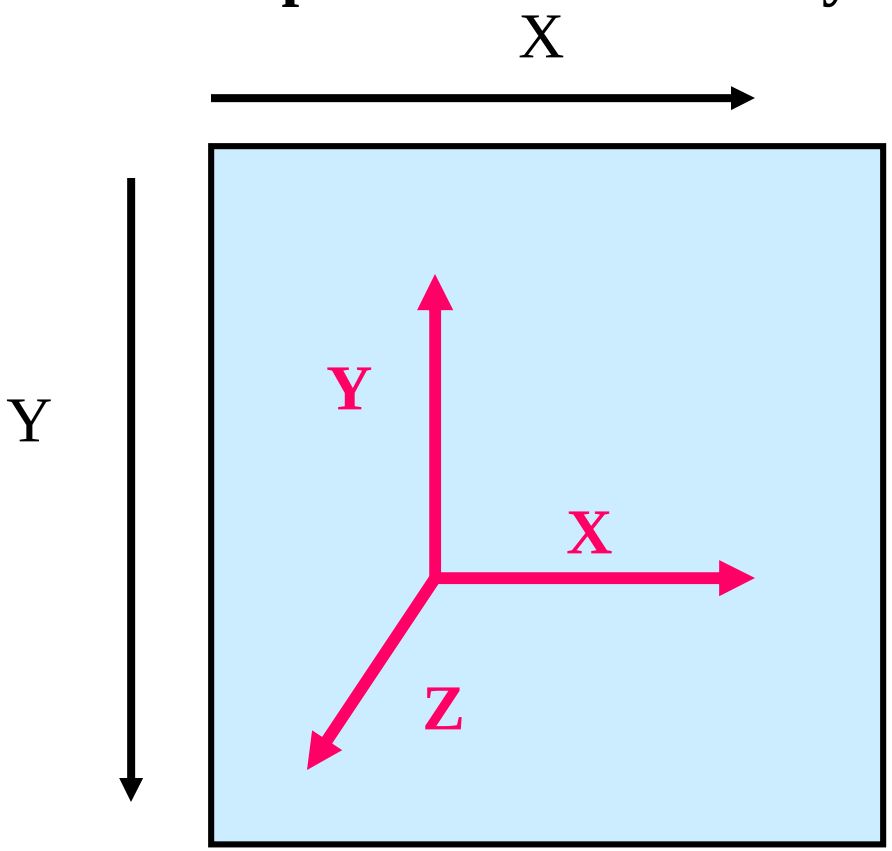
```
…
typedef struct {
   void (* fn)(...);
   ...; } Callback;
Callback cb[n];
…
void addCallback(
   void (* fn) (...), int cbId){...}
…
void leftButtonPress {} {
   cb[LBN]-> fn(...); }
…
```

user’s code

```
…
void myPress(...) {...}
…
int main () {
   addCallback(myPress, LBN);
   …
}
```



Glut Vs. openGL coordinate systems



Glut uses a window manager's coordinate system.

openGL uses a right handed coordinate system.

GLUT mouse drag (x,y) values are in the GLUT coordinate system....

http://www.opengl.org/documentation/specs/glut/spec3/spec3.html

Viewport

```
glViewport(x, y, width, height);  // origin, offsets
```

Initially the size of the window created.
Place in resize function to adjust accordingly.

Glut windows

```
int glutCreateWindow(char *name);
```

The initial window is at -1, -1 with a size of 300 by 300.

```
void glutInitWindowSize(int width, int height);

void glutInitWindowPosition(int x, int y);

void glutSetWindowTitle(char *name);

void glutSetIconTitle(char *name);

void glutFullScreen(void);

void glutSetCursor(int cursor);
```

GLUT display

`glutPostRedisplay()` explicitly calls the rendering function set with **`glutDisplayFunc(...)`**.

`glutSwapBuffers()` causes double buffered output to be displayed -- usually last stmt in rendering function.

the rendering function is also called on window changes after the call to the resize function set with **`glutReshapeFunc(...)`**

the render function has (int width, int height) args.

glutTimerFunc(...) sets the timer function to be called after an interval (msec) occurs. A user specified int value can be passed to the timer function. (animation)

glutIdleFunc(...) sets the idle function to be called when there are no other events pending. (background sensor reading...)

Glut keyboard

X and Y arguments are mouse position in window coordinates

```
// set handler
glutKeyboardFunc(void (*func)
   (unsigned char key, int x, int y));  // set handler

// define handler
void keyboardFuncName(unsigned char key, int x, inty) {
   if (key == 'q' || key == 'Q') exit(); }

// set handler
glutSpecialFunc(void (*func) (int key, int x, int y));

// define handler
void specialFuncName(int key, int x, int y) {
   if ( key == GLUT_KEY_F1) // do something;
   }
```

Glut mouse

Mouse movement w/ no buttons pressed is passive motion.

Need to keep old values of mouse position: say, mtX, mtY
Then relative mouse movement is :

`x - mtX` and **`mtY - y`**

```
// set handler
glutMotionFunc(void (*func)(int x, int y));

// define handler
void motionFuncName(int x, int y) {...}

// set handler
glutPassiveMotionFunc(void (*func)(int x, int y));

// define handler
void passiveMotionFuncName(int x, int y) {...}
```

Glut shapes


```
void glutSolidSphere(GLdouble radius,
   GLint slices, GLint stacks);

void glutWireSphere(GLdouble radius,
   GLint slices, GLint stacks);
```

radius The radius of the sphere.
slices Subdivisions around the Z axis (longitude).
stacks Subdivisions along the Z axis (latitude).

```
void glutSolidCube(GLdouble size);
void glutWireCube(GLdouble size);
void glutSolidCone(GLdouble base, GLdouble height,
   GLint slices, GLint stacks);
void glutWireCone(GLdouble base, GLdouble height,
   GLint slices, GLint stacks);
void glutSolidTorus(GLdouble innerRadius,
   GLdouble outerRadius, GLint nsides, GLint rings);
void glutWireTorus(GLdouble innerRadius,
   GLdouble outerRadius, GLint nsides, GLint rings);
```

... more... see documentation

GLUT menus

OpenGL and GLUT have states.
There is the current menu, the current color, ...
(minimize use of menus for interactive "game" programs)

```
glutSetMenu(int menu);
int glutGetMenu();
int glutCreateMenu(menuFn);   // create menu, set handler
glutDestroyMenu(int menu);
glutAddMenuEntry(“string”, int index);
glutRemoveMenuItem(int index);
glutAddSubMenu(“string”, int menu); // to current menu
glutAttachMenu(int button);
   GLUT_RIGHT_BUTTON, GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON
glutDetachMenu(int button);
glutChangeToSubMenu(int entry, “string”, int menu);
   changes menu entry to submenu
glutChangetoMenuEntry(int entry, “string”, int value);
```

Glu Quadrics


```
GLUquadricObj * gluNewQuadric()
void gluDeleteQuadric(GLUquadricObj * obj)

// slices are lines longitude  -- lengthwise, horizontal
// stacks are lines latitude  -- vertical

void gluSphere(GLUquadricObj * obj, GLdouble radius,
   GLint slices, GLint stacks);

// base in plane z = 0, base and top radii,
// stacks in z, slices in y
gluCylinder(GLUquadric * obj, GLdouble base,
   GLdouble top, GLdouble height, GLdouble slices,
   GLdouble stacks);

// plane z = 0,
gluDisk(GLUquadricObj * obj, GLdouble inner,
   GLdouble outer, GLint slices, GLint rings)
// remove wedge from disk, start angle + angle degrees
gluPartialDisk(GLUquadricObj * obj, ... GLdouble start,
   GLdouble angle)
```

Transformations

GL transformation commands use float or double arguments to change coordinate values (x, y, z)

Translate – set values for the coordinates

```
glTranslatef( x, y, z);// float, move on coordinates
```

Scale – multiply the x, y, and z values

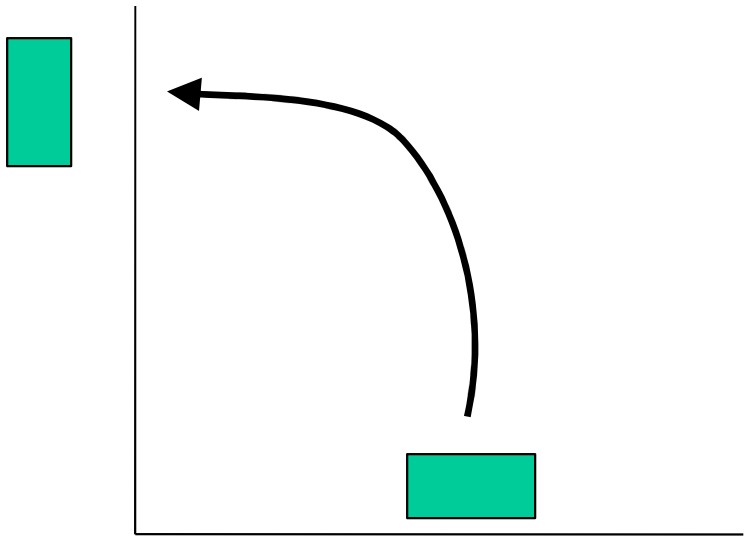
```
glScalef(x, y, z);
```

Rotate – change orientation (position) of coordinates

```
glRotatef(angle, x, y, z) // angle in degrees on axis
```

Transformations order sensitive

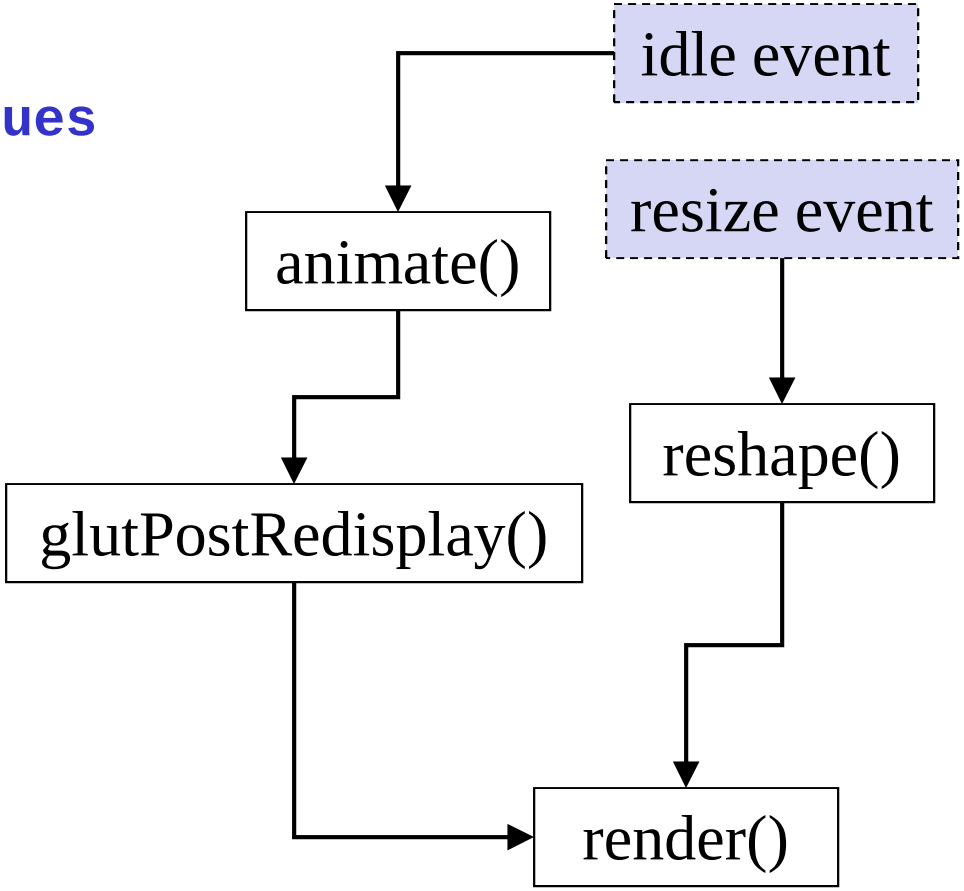
```
glTranslatef(10.0, 2.0, 0.0)
glRotatef(90.0,  0.0, 0.0, 1.0)

// "in place"
glTranslate(-10.0, -2.0, 0.0)
glRotatef(90.0, 0.0, 0.0, 1.0)
glTranslatef(10.0, 2.0, 0.0)
```

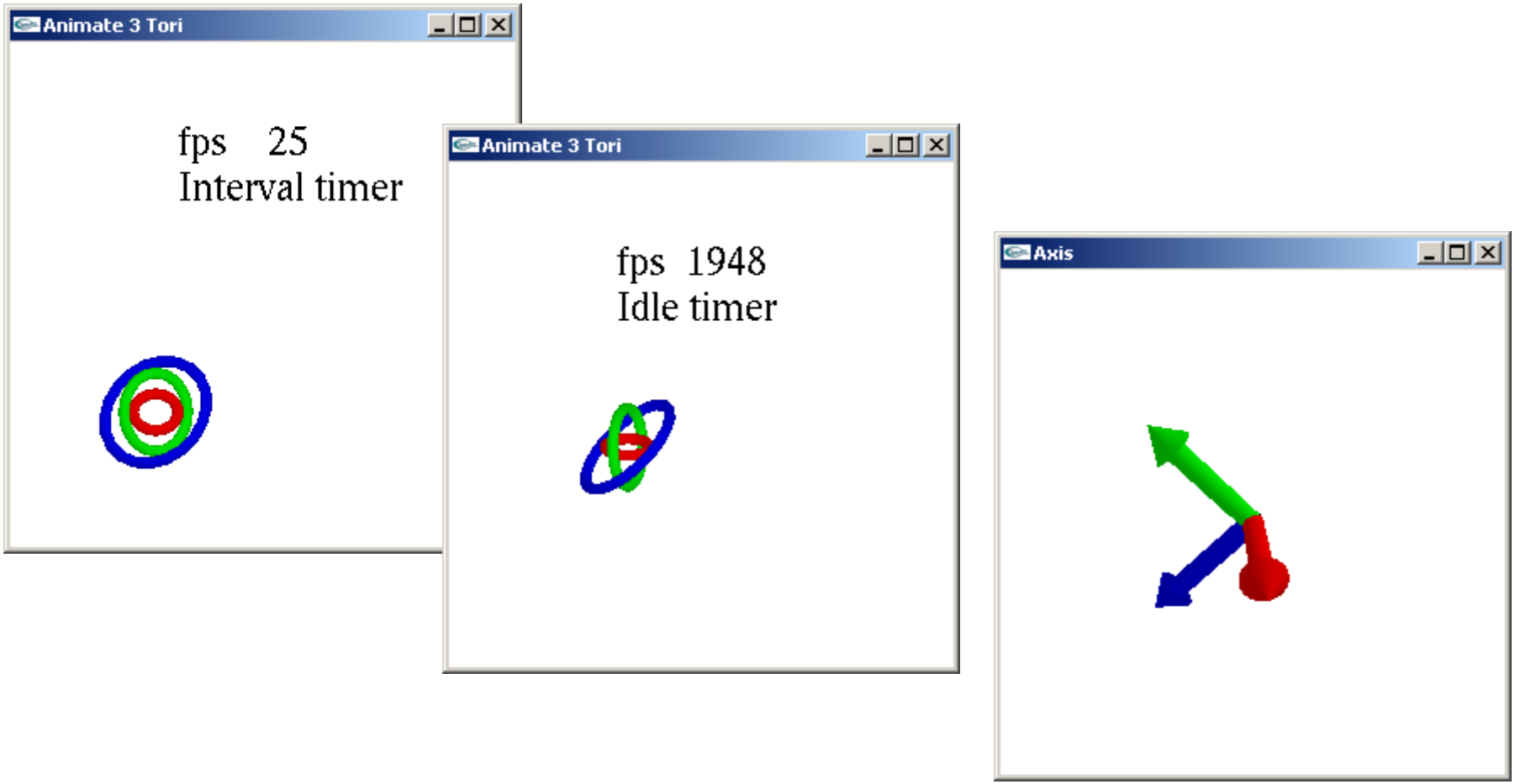


Animation

Uses the GLUT Idle callback function(or timer) to set values for transformation matrices and/or render states for objects

```
// void glutIdleFunc (void  (*func) (void))

void animate(void) {
    // set up animation values
    glutPostRedisplay();
    }
...
void render (GLenum mode){
    // draw
    }
...
void main () {
    ...
    glutIdleFunc(animate);
    }
```

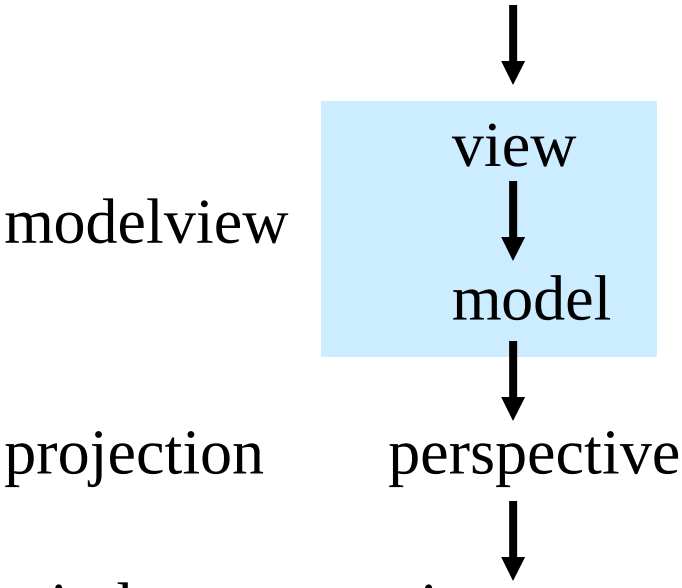




animate.cpp : timerFunc, idleFunc, display lists, glutBitmapCharacter

axis.cpp: Glu quadrics, menu, mouse drag events

OpenGL matrices



from where the scene is looked at

mainpulations of objects in the scene

3D ⇒ 2D how depth is displayed

scale / clip to window dimensions

ModelView matrix is used for both view and model -- these operations are not independent.

ModelView operations "*move the origin*" of the next draw request

Set the current matrix, initialize the current matrix

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

Matrix Stack

OpenGL maintains a separate stack of modelview and projection matrix operations.

The current matrix is the top of the stack.

Each level in the matrix stack is “group” or “local” composite matrix

Matrix commands are applied to the current stack.

The last transformation is first one applied (matrix postmultiplication)

A GL *"Transform group”*

```
  glPushMartrix();      // save current matrix on stack
3 glTranslatef( tx, ty, tz);     // update w/ translation
2 glRotatef( angle, rx, ry, rz); // update w/ rotation
1 glScalef( sx, sy, sz);    // update w/ scale
  glutSolidCube(20);        // draw  at the new center
  glPopMatrix();            // restore the previous matrix
```

GL is state specific. It draws in the current color state with the current attribute state using the current matrix ....

Matrix state is preserved by popping and pushing

Changing and restoring field of view

// set up the original projection matrix

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPespective(45.0, width/height, 50.0, 300.0);
glPostRedisplay();  // render the scene
... // assume switch to GL_MODELVIEW matrix
glMatrixMode(GL_PROJECTION);
glPushMatrix(); // save initial perspective
glLoadIdentity();
gluPespective(90.0, width/height, 50.0, 300.0);
glPostRedisplay();
...
glMatrixMode(GL_PROJECTION);
glPopMatrix();   // restore initial perspective
```

Conceptual Orientation & Translation matrix

	right	up	at	origin
X	R_x	U_x	A_x	T_x
Y	R_y	U_y	A_y	T_y
Z	R_z	U_z	A_z	T_z
H	0	0	0	1

Successive OpenGL transformation calls generate a concateated ModelView matrix.

This matrix contains orientation and position for the current reference frame".

OpenGL matrices contain 16 GLdouble or GLfloat values

```
GLfloat  m[16];  // or GLdouble * m;
```

$$\begin{bmatrix} m[0] & m[4] & m[8] & m[12] \\ m[1] & m[5] & m[9] & m[13] \\ m[2] & m[6] & m[10] & m[14] \\ m[3] & m[7] & m[11] & m[15] \end{bmatrix}$$

conceptual and openGL matrices revisited in Transformations notes

GL matrix functions


```
// current matrix (CM),  current matrix stack (CMS)

glGetGLTypev(GLType, GLType);    // get matrix values
// glGetFloatv(GL_MODELVIEW_MATRIX,  aFloatMatrix);

glLoadIdentity()     // CM set to indentity

glLoadMatrix?(const GLType * m)  // CM = m

glMatrixMode(...)    // set matrix type
// types: GL_MODELVIEW, GL_PROJECTION, GL_TEXTURE

glMultMatrix?(const GLType * m)  // CM' = CM*m

glPopMatrix()        // CM = CMS
glPushMatrix()       // CMS += CM   CM == top of CMS
// each cms initially has identity matrix (IM)
// error to pop stack w/ IM
```

Projection

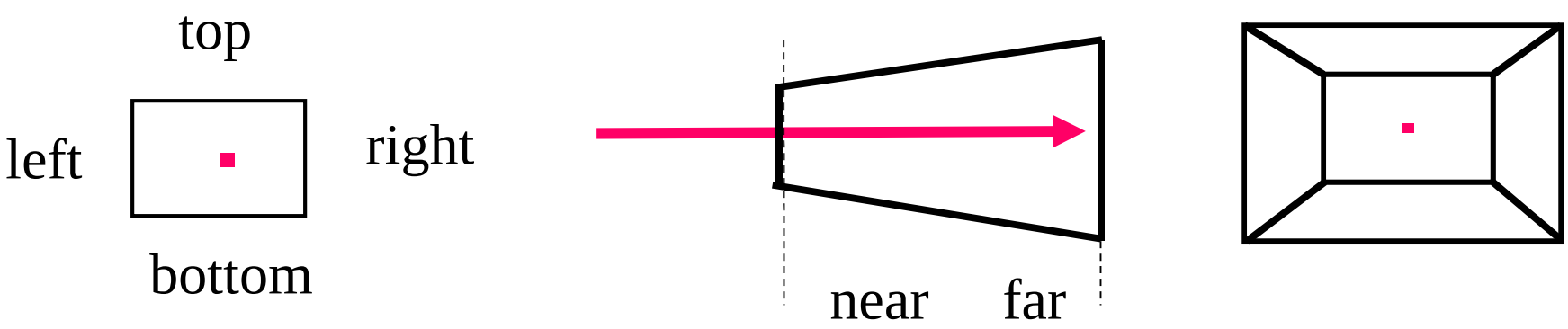
Orthographic project is a parallel projection of 3D into 2D (disregard the z axis values...). Useful for CAD... see text examples.

```
glOrtho(left, right, bottom, top, near, far)
gluOrtho(left, right, bottom, top);
gluOrtho(-100.0, 100, -100.0, 100.0);
```

Perspective projection defines a “vanishing point” on +Z and scales point position by their distance from the viewer.

```
gluPerspective(FieldOfViewAngleY, aspectXY, near, far);
gluPerspective(45.0, width/height, 5.0, 300.0);
```

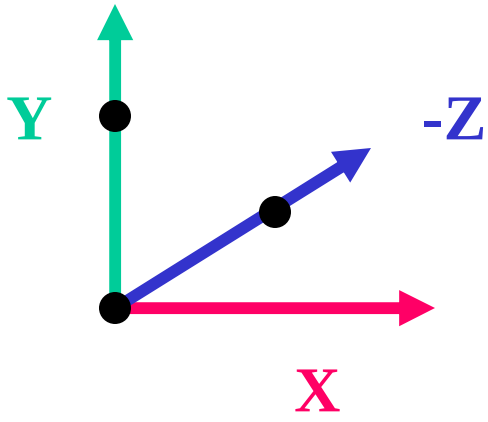
Projection defines a viewing volume or frustum:



Viewpoint

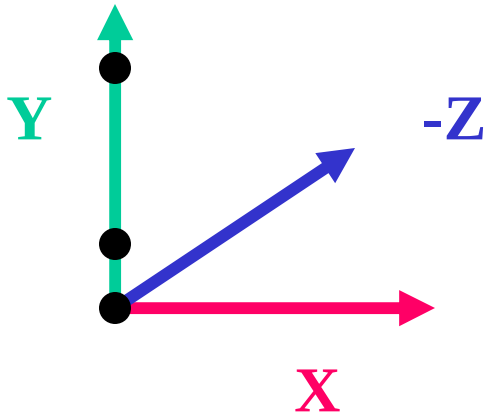
Define a “sight” vector from the eye to the center of the scene.
Define an “up” vector.

```
gluLookAt(  eyeX, eyeY, eyeZ,
            centerX, centerY, centerZ,
            upX, upY, upZ);

gluLookAt ( 0,   0,    0,
            0,   0, -100,
            0, 100,    0);
```



Can’t have the sight and up vector be the same. Won’t see anything.
Lighting incorrect (later)

```
gluLookAt(  0,   0, 0,
            0,  10, 0,
            0, 100, 0);
```



<< show Robin's Tutors projection >>

This doesn't work for changing values!

```
double eye[3] = {0.0,   0.0,    0.0};
double  at[3] = {0.0,   0.0, -100.0};
double  up[3] = {0.0, 100.0,    0.0};
double vx, vy, vz, rvx, rvy, rvz;

...

gltranslatef(vx, vy, vz);
glrotatef(rvx, rvy, rvz);
glBegin(GL_POINTS);
   glVertex3dv(eye);    // glVertex3dv(const double*)
   glVertex3dv(at);     // values for eye, at, up
   glVertex3dv(up);     // are not changed
glEnd();
gluLookAt(eye[0], eye[1], eye[2],  // 0.0, 0.0, 0.0
          at[0],  at[1],  at[2],
          up[0],  up[1],  up[2]);
```

Drawing

```
Vertices   glVertex3f(xcoord, ycoord, zcoord)
           glVertex3f(100.0, 0.0, 200);

           glVertex3fv(ptArray)
           double aPt[3] = {100.0, 0.0, 200);
           glVertex3fv(aPt);
```

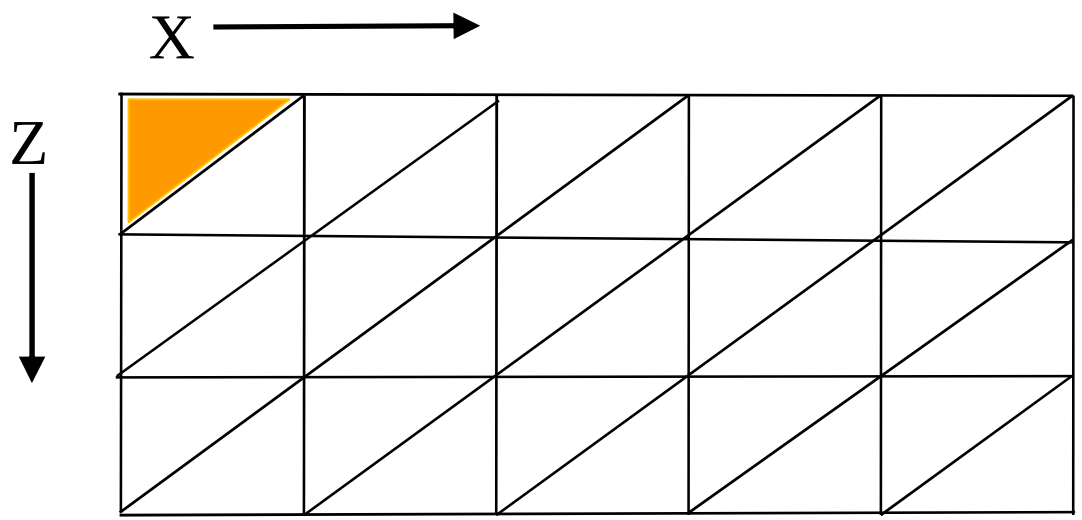
Drawing a shape from points requires "shape" info.: `GL_POINTS, GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP,GL_POLYGON, GL_TRIANGLE, GL_QUADS, GL_TRIANGLE_STRIP, GL_QUAD_STRIP, GL_TRIANGLE_FAN`

```
glBegin(GL_LINE);
   glVertex3f(...);
   glVertex3f(...);
glEnd();
```

Mesh

A mesh (grid) can be drawn with triangles.
Triangle surfaces are always planar – they have a single normal

```
glPolygonMode(GL_FRONT, GL_FILL);
…
   glBegin(GL_TRIANGLE);  // counter clockwise
      glVertex3f(0.0, 0.0, 0.0);
      glVertex3f(0.0, 0.0, 1.0);
      glVertex3f(1.0, 0.0, 0.0);
   glEnd();
```



Mesh indices

```
float GridMesh[XDim][ZDim][3]  // X Z vertex pts
int Indices[6 * (XDim - 1) * (ZDim – 1)] [2] // X Z
…
glBegin(GL_TRIANGLE);
   for(i = 0; i < 6* (XDim-1) * (ZDim-1), i++)
      glVertex3fv(
         GridMesh[Indices[i][0]] [Indices[i][1]])
   glEnd();
…
```

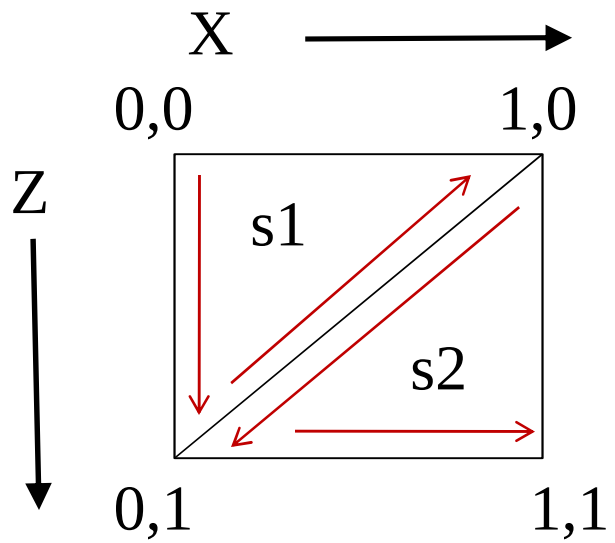
// from the perspective of points

GM == { (0,0), (0,1), (1,0), (1,1)}

I == { 0, 1, 2, 2, 1, 3}

surface 1 = = (0,0), (0,1),(1,0)

surface 2 == (1,0), (0,1), (1, 1)



Surface normal

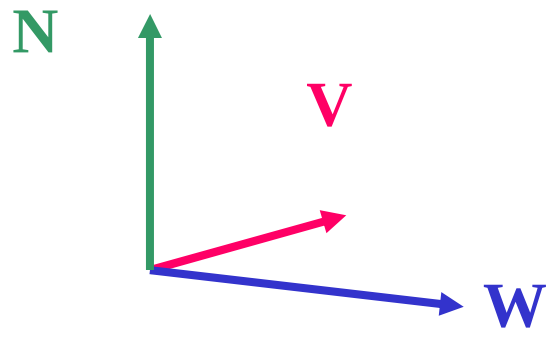
`glNormal3*(x, y, z)` can be used to set the current normal for use with graphics primitives: `GL_TRIANGLES, GL_QUADS, GL_POLYGON`

`glScale` can make unit normals un-normalized.

Can use `GL_NORMALIZE` to make all normals unit normals on each rendering – has performance hit

Normal is the cross product of 2 non parallel vectors

$$N = W \times V$$



Normal = cross(W, V)

$$\boldsymbol{N} = \boldsymbol{W} \times \boldsymbol{V} = \begin{bmatrix} W_y V_z - W_z V_y \\ W_z V_x - W_x V_z \\ W_x V_y - W_y V_x \end{bmatrix}$$

Length of normal

$$\boldsymbol{L} = \sqrt{N_x^2 + N_y^2 + N_z^2}$$

Unit normal

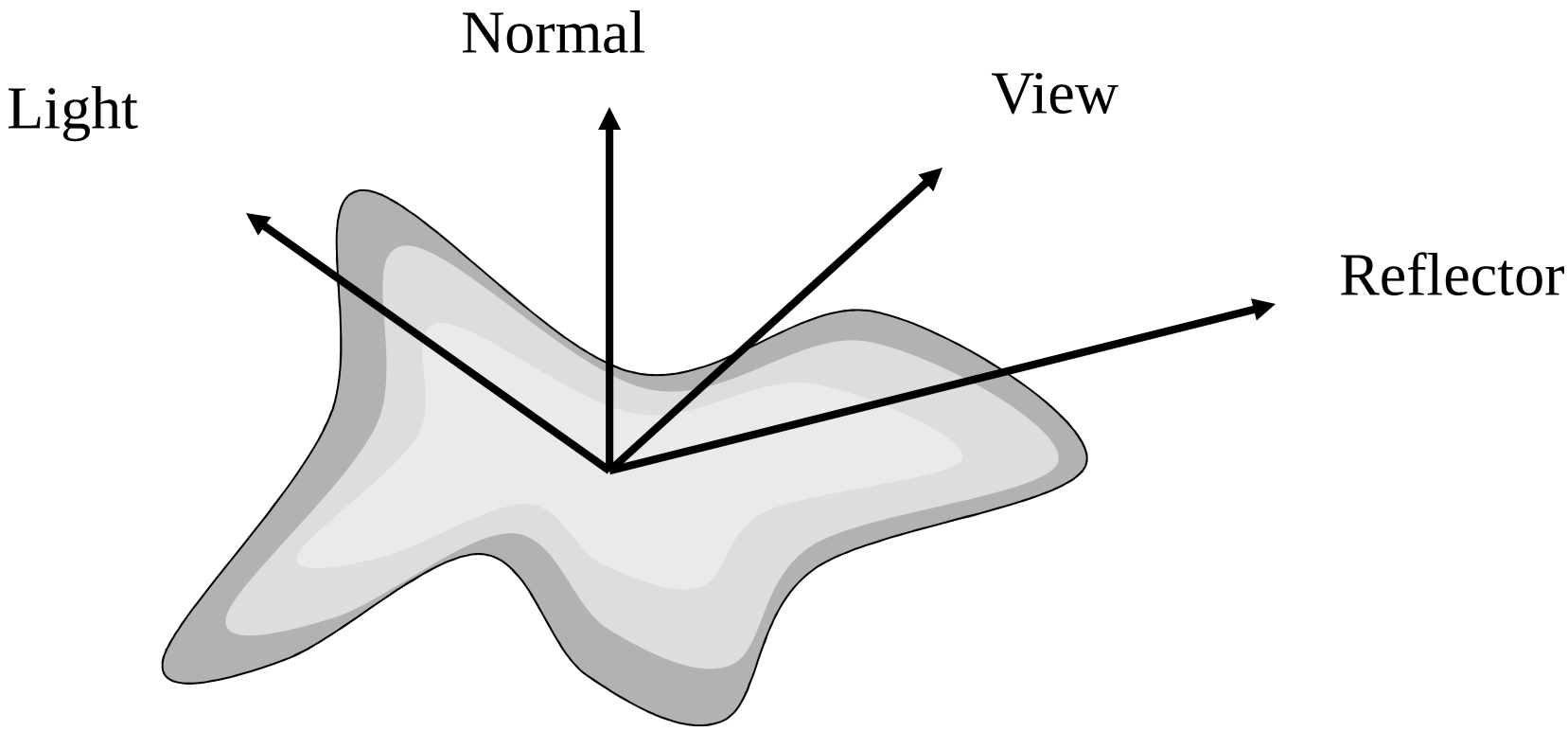
$$\boldsymbol{N}_{unit} = \begin{bmatrix} \frac{N_x}{L} + \frac{N_y}{L} + \frac{N_z}{L} \end{bmatrix}$$

Lights

A point on a surface is seen as reflected light from a view.

The light can have an intensity and a color that illuminates the surface.

The surface has color and reflective properties.



Phong model

4 contributions to the shading of a point

diffuse	light reflected in all directions (!V)
specular	shininess of reflections from smooth surface (R V)
ambient	sum of all light source (intesity of light, !R !V)
emissive	glowing – light source (!L, lights ! visible)

Unit normals must be provided for every face at the vertex if openGL lighting is to be used.

Glu and Glut objects have normals.

```
glEnable(GL_NORMALIZE);     // makes normals unit normals
```

OpenGL light sources: point, spotlight, and ambient

each has: diffuse, specular and ambient RGB parameters

Imagine a white light in room with green walls

diffuse and specular parameters are white

reflections are green – so the ambient parameter is green

```
// declare for each light
GLfloat lightPos0[]  = {10.0, 20.0, 50.0, 1.0};
GLfloat diffuse0[]   = {1.0, 0.0, 0.0, 1.0};    // red
GLfloat specular0[]  = {1.0, 1.0, 1.0, 1.0};    // white
GLfloat ambient0[]   = {0.1, 0.1, 0.1, 1.0};    // grey

glEnable(GL_LIGHTING);          // have lights
glEnable(GL_LIGHT0);            // first light, numbered 0

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
// do for each light
glLightfv(GL_LIGHT0, GL_POSITION, lightPos0);
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse0);
glLightfv(GL_LIGHT0, GL_SPECULAR, specular0);
glLightfv(GL_LIGHT0, GL_AMBIENT, ambient0);
...
```

Could use arrays of structs (classes) to represent lights.

```
// declare for each light
GLfloat lightPos0[]  = {10.0, 20.0, 50.0, 1.0};
GLfloat diffuse0[]   = {1.0, 0.0, 0.0, 1.0};    // red
GLfloat specular0[]  = {1.0, 1.0, 1.0, 1.0};    // white
GLfloat ambient0[]   = {0.1, 0.1, 0.1, 1.0};    // grey

glEnable(GL_LIGHTING);          // have lights
glEnable(GL_LIGHT0);            // first light, numbered 0

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
// do for each light
glLightfv(GL_LIGHT0, GL_POSITION, lightPos0);
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse0);
glLightfv(GL_LIGHT0, GL_SPECULAR, specular0);
glLightfv(GL_LIGHT0, GL_AMBIENT, ambient0);
...
```

Could use arrays of structs (classes) to represent lights.

Light sources

Ambient light has no source.
Each "sourced light" can contribute to the ambient lighting.

Distant (parallel) light is like the sun.
Most efficient lighting calculations.

Point light is like a lightbulb.
Light emits in all directions and has an attenuation.

Spot light is like a flashlight, it has inner and outer cone of intensity

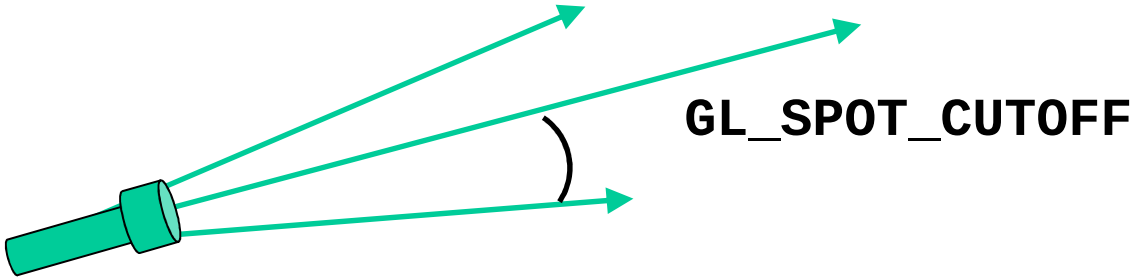
LightPos 4th argument 1 == point source or 0 == distant source

```
GLfloat lightPos0[]  = {10.0, 20.0, 50.0, 0.0};
...
glLightfv(GL_LIGHT0, GL_POSITION, lightPos0);
```

<< lightposition tutor >>

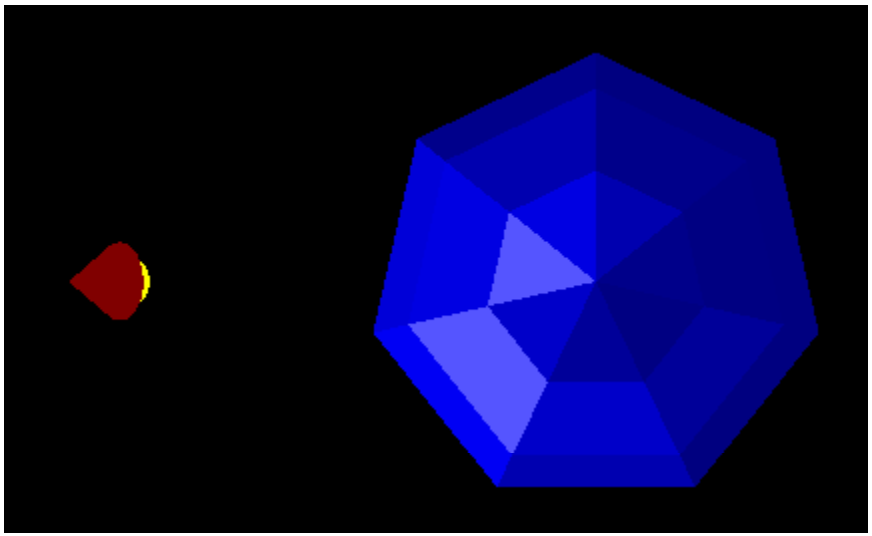
Spot light

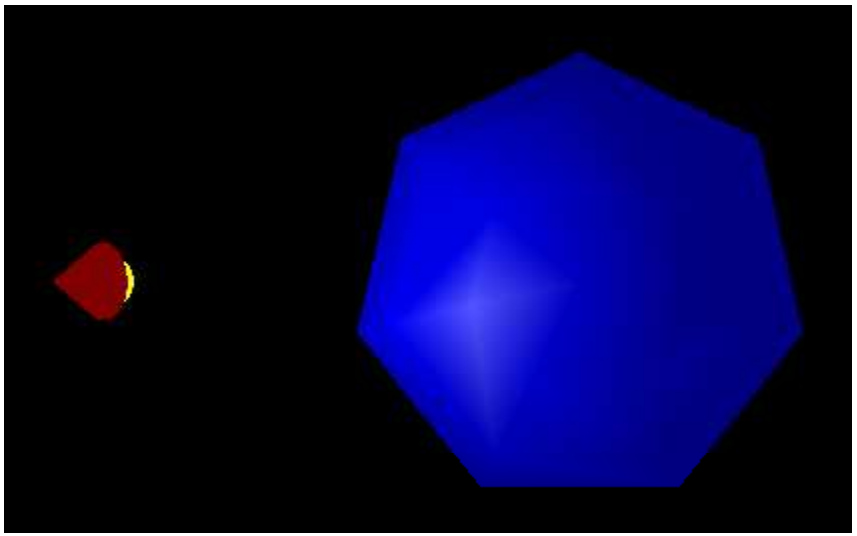

```
// set Spot light position GLfloat
lightPos[]={0.0,0.0,60.0,1.0};
glLightfv(GL_LIGHT0,GL_POSITION,lightPos);
...
// set spot light parameters
// direction of light, default (0, 0, -1)
GLfloat spotDir[]={0.0,1.0,-1.0};
// define spot direction
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, spotDir);
// set cutoff angle, range 90..180, default 180
glLightf(GL_LIGHT0, GL_SPOT_CUTOFF,40.0);
// set focusing strength, higher value more focused,
// range 0..128, default 0
glLightf(GL_LIGHT0, GL_SPOT_EXPONENT,7.0);
```



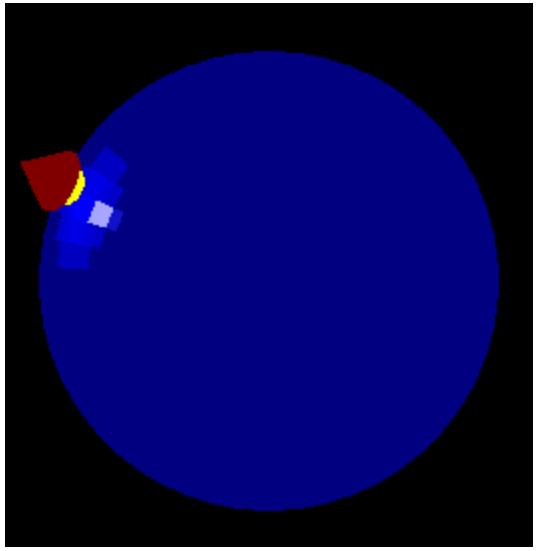
Spot demo

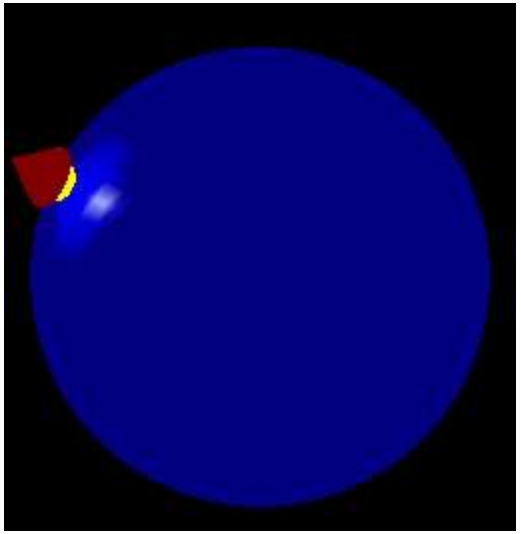
Modified spot demo from SuperBible 4th Ed. pp 212 - 222





low tesselation
flat and smooth





high tesselation
flat and smooth

Materials

Materials have: ambient, diffuse, specular, and shininess values

```
void glMaterial{ifv}(GLenum face, GLenum name, TYPE value)
void glMaterialfv(GLenum face, GLenum name, TYPE * value)
```

There is one current material in the gl state.

```
typedef struct Material {
   GLfloat ambient[4];  GLfloat diffuse[4];
   GLfloat specular[4]; GLfloat shininess;
   } Material;
...
void setMaterial(Material * m, GLenum face) {
   glMaterialfv(face, GL_AMBIENT,   m->ambient);
   glMaterialfv(face, GL_DIFFUSE,   m->diffuse);
   glMaterialfv(face, GL_SPECULAR,  m->specular);
   glMaterialf(face,  GL_SHININESS, m->shininess); }

also GL_AMBIENT_AND_DIFFUSE, GL_EMISSION
```

```
// material values from E. Angel,
// OpenGL A Primer, 2002.

Material brass = {
   {0.33, 0.22, 0.03, 1.0},  //  ambient
   {0.78, 0.57, 0.11, 1.0},   // diffuse
   {0.99, 0.91, 0.81, 1.0},   // specular
   27.8};                     // shininess

Material redPlastic = {
   {0.3, 0.0, 0.0, 1.0}, {0.6, 0.0, 0.0, 1.0},
   {0.8, 0.6, 0.6, 1.0}, 32 };

setMaterials(redPlastic, GL_FRONT_AND_BACK);
```

Also need to select a lighting model:

```
glLightModel{if}(Glenum pname, TYPE param)
   GL_LIGHT_MODEL_AMBIENT            RGBA
   GL_LIGHT_MODEL_LOCAL_VIEWER       0.0 || GL_FALSE
   // false, infinite viewing all specular reflections ==
   GL_LIGHT_MODEL_TWO_SIDE           0.0 || GL_FALSE
```

Color Material

glMaterial*() affects performance

```
void glColorMaterial(GLenum face, GLenum mode);
   defaults GL_FRONT_AND_BACK, GL_AMBIENT_AND_dIFFUSE
```

sets mode material properties of face based on a color

```
glEnable(GL_COLOR_MATERIAL);
glColorMaterial(GL_FRONT, GL_DIFFUSE);
glColor3f(0.2, 0.5, 0.8);
....
//draw
```

Shading

GL_FLAT	color computed by first vertex
GL_SMOOTH	color interpolated between vertices

<< lightlab demo – source ?? >>
<< lightmaterial tutor >>

Enabling GL features

GL features that affect rendering must be enabled, disabled.

stipple(fill), culling, lighting, hidden surface removal, texture mapping

```
glEnable(GL_LINE_STIPPLE);
```

Polygons

Triangle polygons prefered: planar, convex

Polygons have front and back faces.

front face vertices are ordered counterclockwise from view

back face vertices are ordered clockwise from view

```
glPolygonMode(face, mode)
glCullFace(face)     // don't render face
```

face: GL_FRONT, GL_BACK, **GL_FRONT_AND_BACK**

mode: GL_POINT, GL_LINE, **GL_FILL**

Color

color used in rendering is the current color.

Color changes between vertices will be **smooth shaded** (interpolated) between the colors.

Color shading can be set to flat shading (no interpolation across face)

`glShadeModel(shadeMode)`

shadeModel: **GL_SMOOTH** or GL_FLAT

Display Lists

Drawing non changing objects

Immediate mode	model changed and rendered each frame
Retained mode	modeled once (compiled) “graphics file” rendered each frame (w/ no changes)

```
#define RED_SQ 1
...
glNewList(RED_SQ, GL_COMPILE);
glColor3f(1.0, 0.0, 0.0);
glRectf(-1.0, -1.0, 1.0, 1.0)
glEndList();

...

glCallList(RED_SQ);
```

Don't load resources (images) inside display lists

Multiple display lists:

glListBase(GLuint offset) offset used by glCallLists(...), default 0

glCallLists(GLsizei n, GLenum type, GLvoid * lists)

executes **n** display lists from integers + current offset

stored in **lists** of **type**

rendering text strings with display lists

```
int base = glGenLists(256); // 256 consecutive list IDs
...
for(i = 0; i < 256; i++) {  // fill ASCII char lists
   glNewList(base + i, GL_COMPILE);
   glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_10, i);
   glEndList(); }
glListBase(base);  // set list base
...
char text[25];  // or * text with appropriate allocation
... // strcpy(text, "Hi CS465!");
glCallLists((Glint) strlen(text), GL_BYTE, text);
```