CS559: Computer Graphics

Lecture 17: Shading in OpenGL, FLTK

Li Zhang

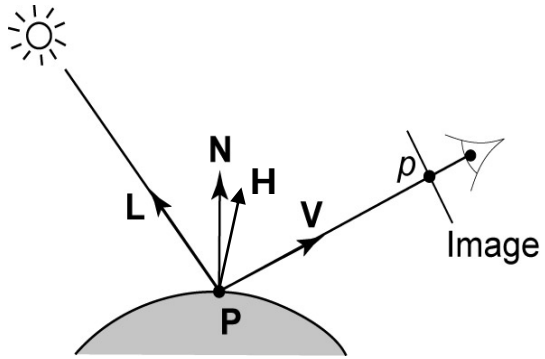
Spring 2008

Today

- shading in OpenGL
- FLTK

- Reading
  - Red book, Ch 4&5 (except color index mode)
  - http://pages.cs.wisc.edu/~cs559-1/tutorials.htm

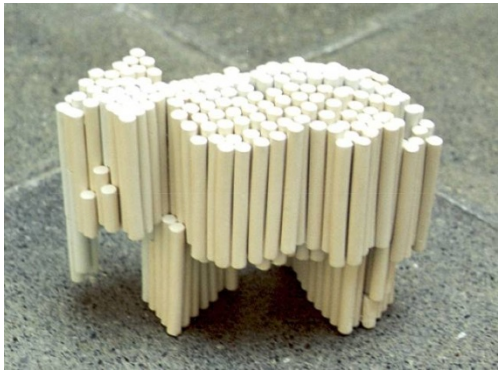
Shading problem

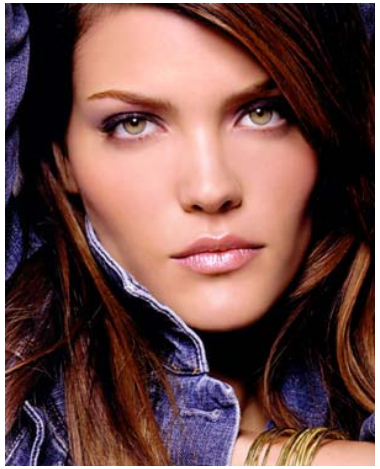


$$\|\mathbf{H}\| = \|\mathbf{L}\| = \|\mathbf{N}\| = \|\mathbf{V}\| = 1$$

Phong Shading Model

$$I = k_e + k_a L_a + k_d L_d \cdot \max(0, \mathbf{L} \cdot \mathbf{N}) + k_s L_s \cdot (\mathbf{H} \cdot \mathbf{N})^{n_s}$$



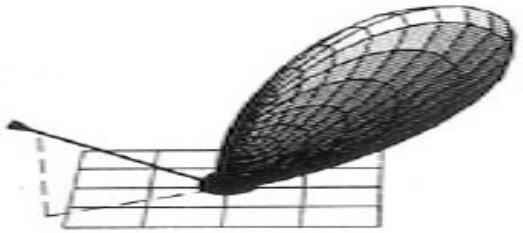


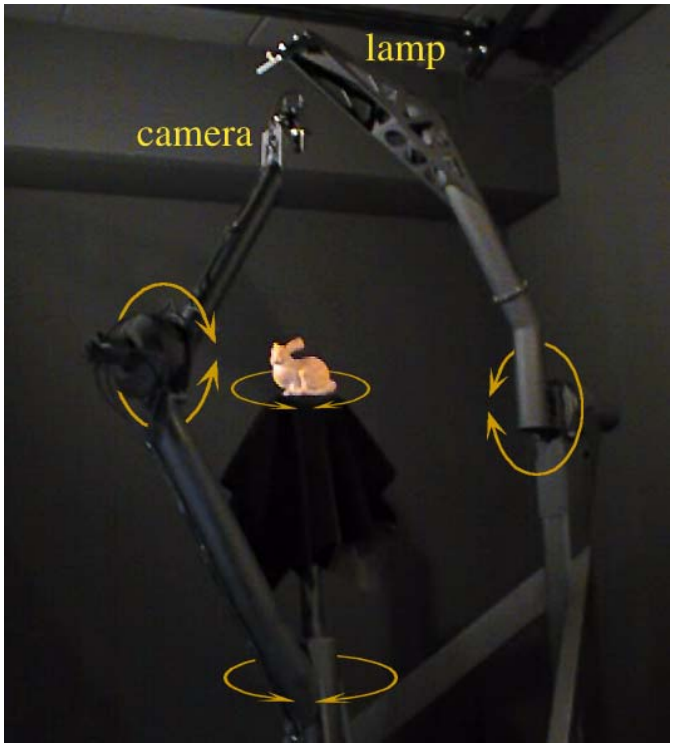
BRDF

- Phong Shading model is a special instance of **Bi-directional Reflectance Distribution Function** (**BRDF**). BRDF's can be quite sophisticated…

$f_r(\omega_{in}, \omega_{out})$

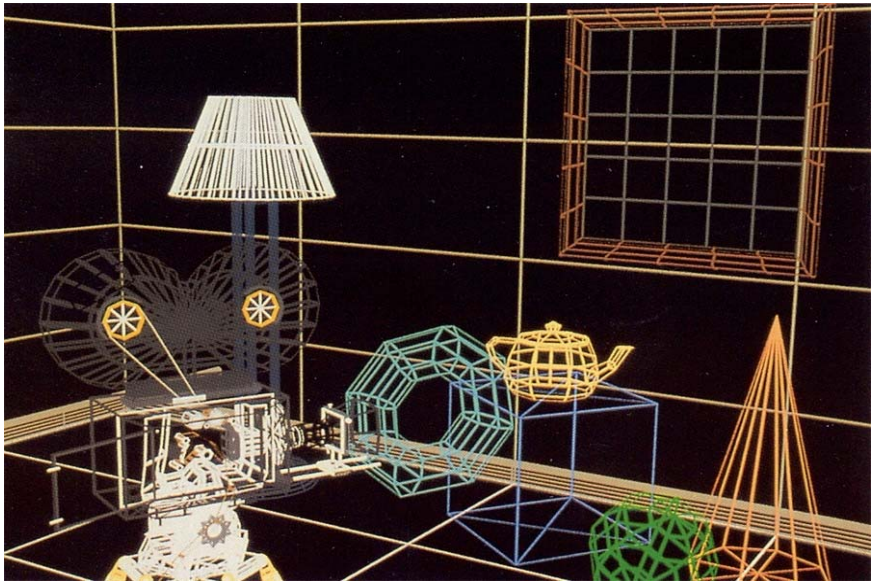
ω_{in}

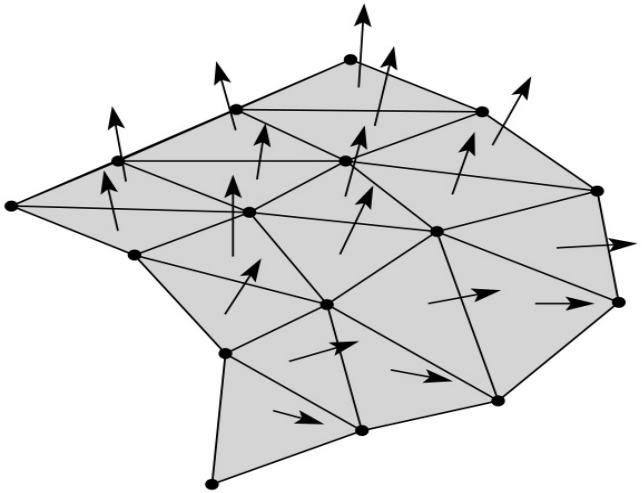


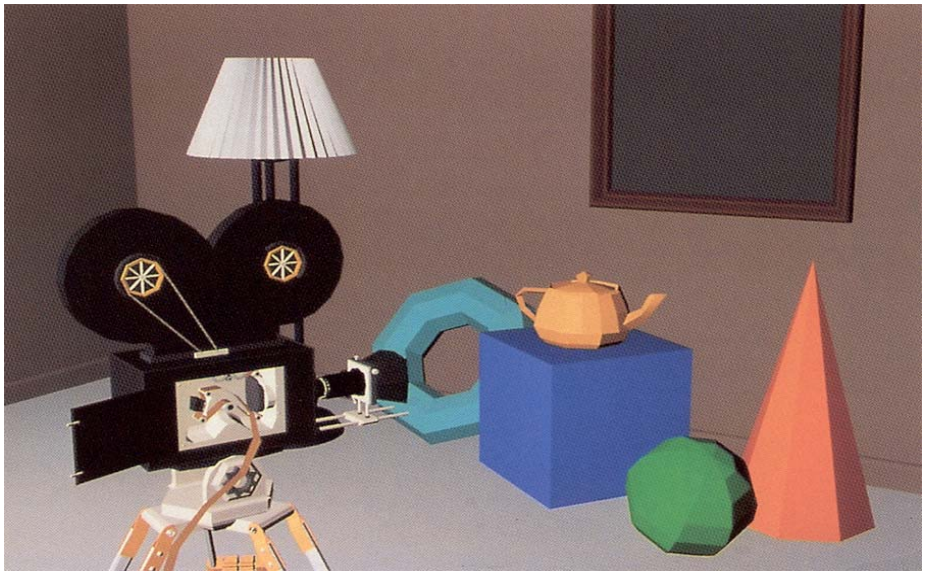


Stanford Graphics Lab

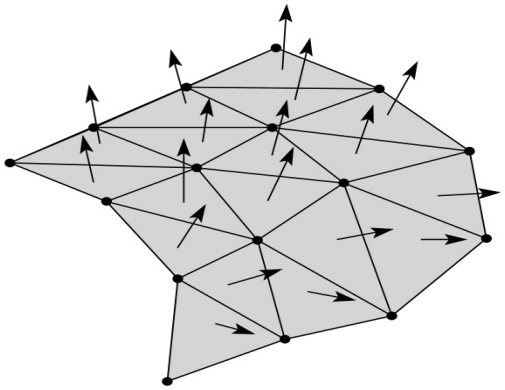
Faceted shading

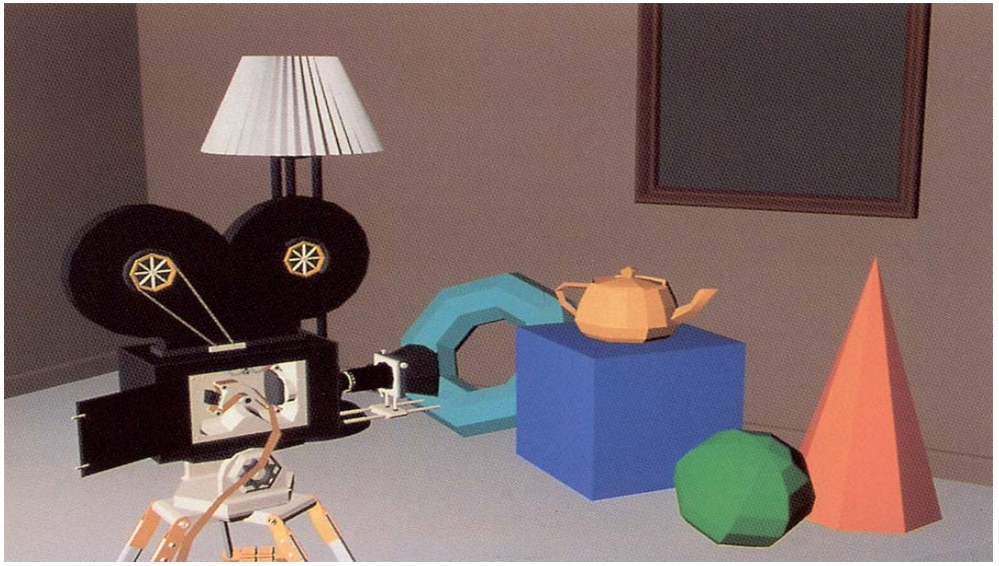


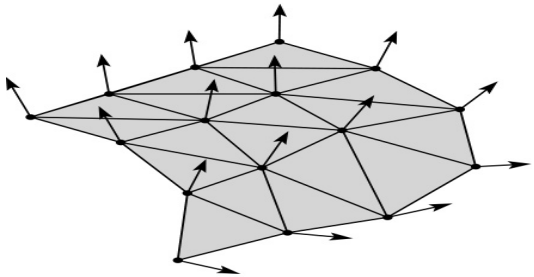


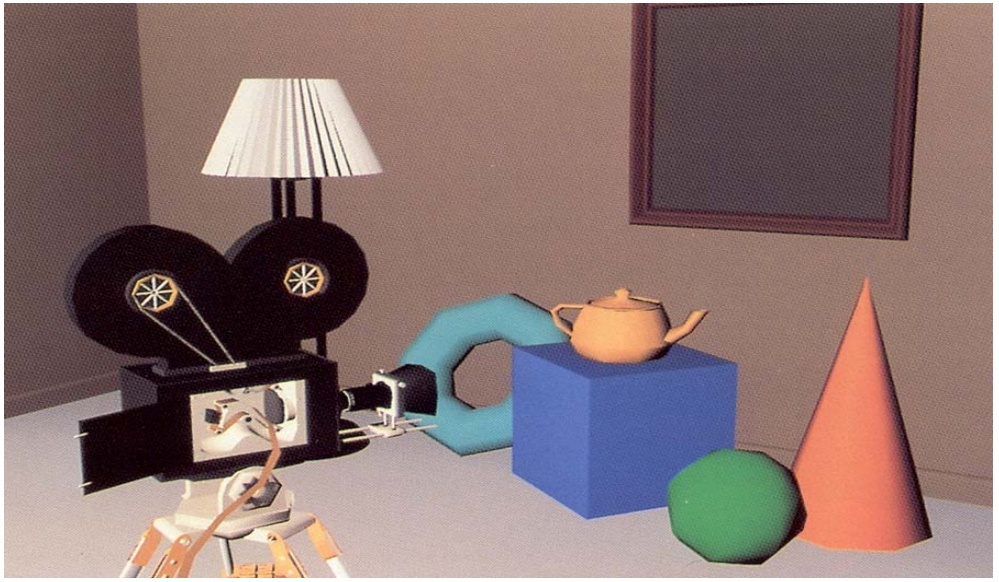


Facted shading vs. Gouraud interpolation

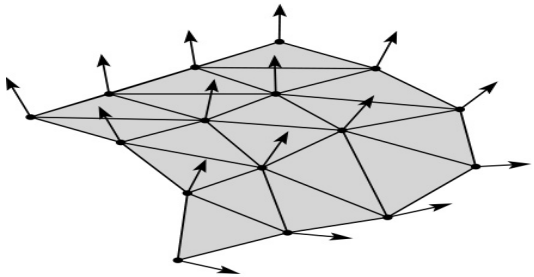


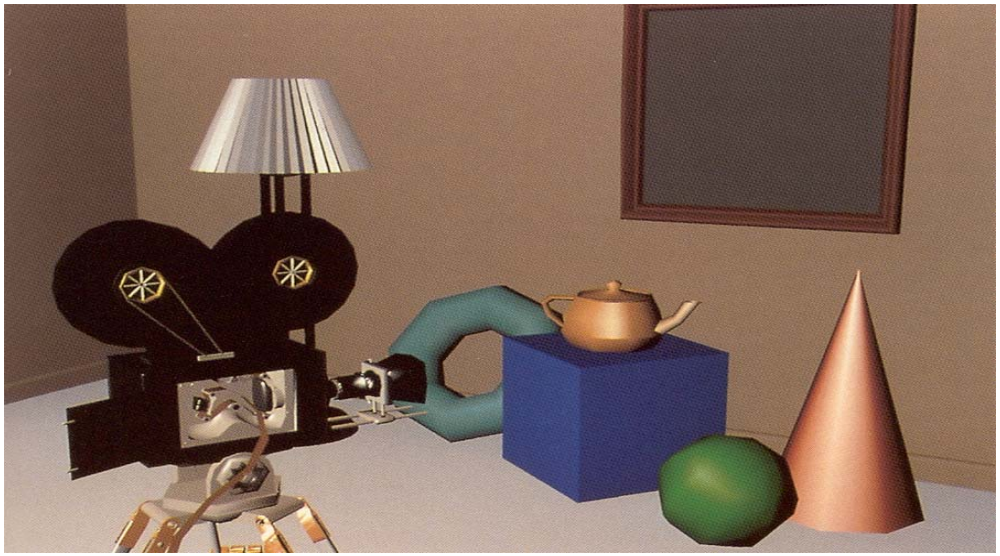






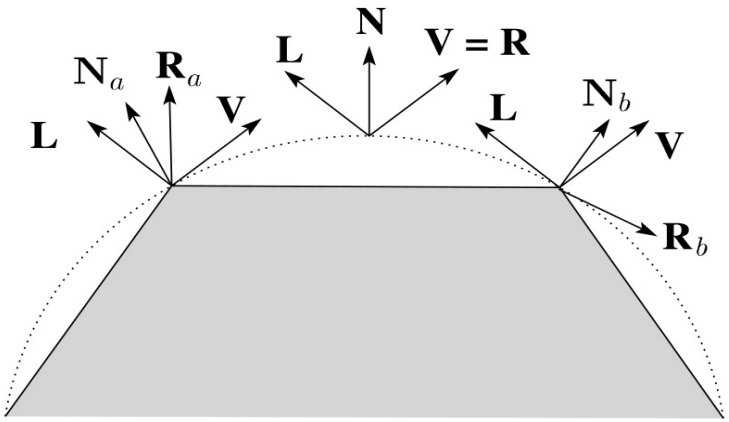
Specular refection artifacts in Gouraud interpolation





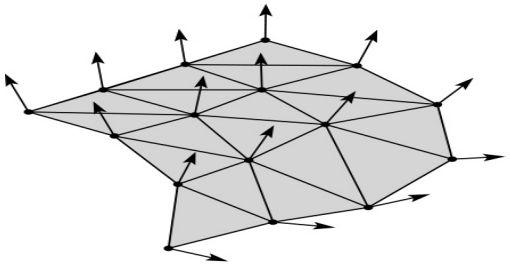
Gouraud interpolation artifacts

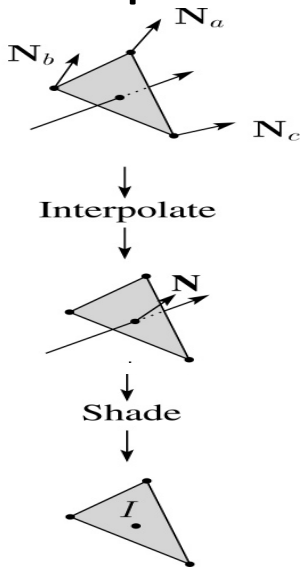
- Gouraud interpolation has significant limitations.
  - If the polygonal approximation is too coarse, we can miss specular highlights.



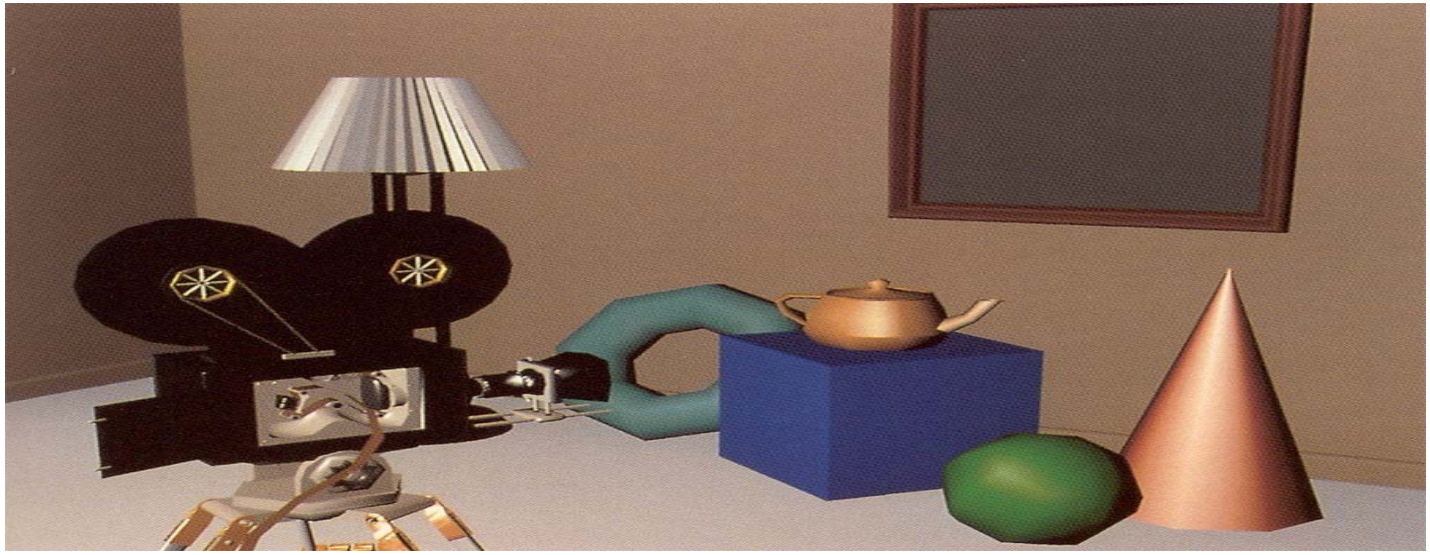
Phong interpolation

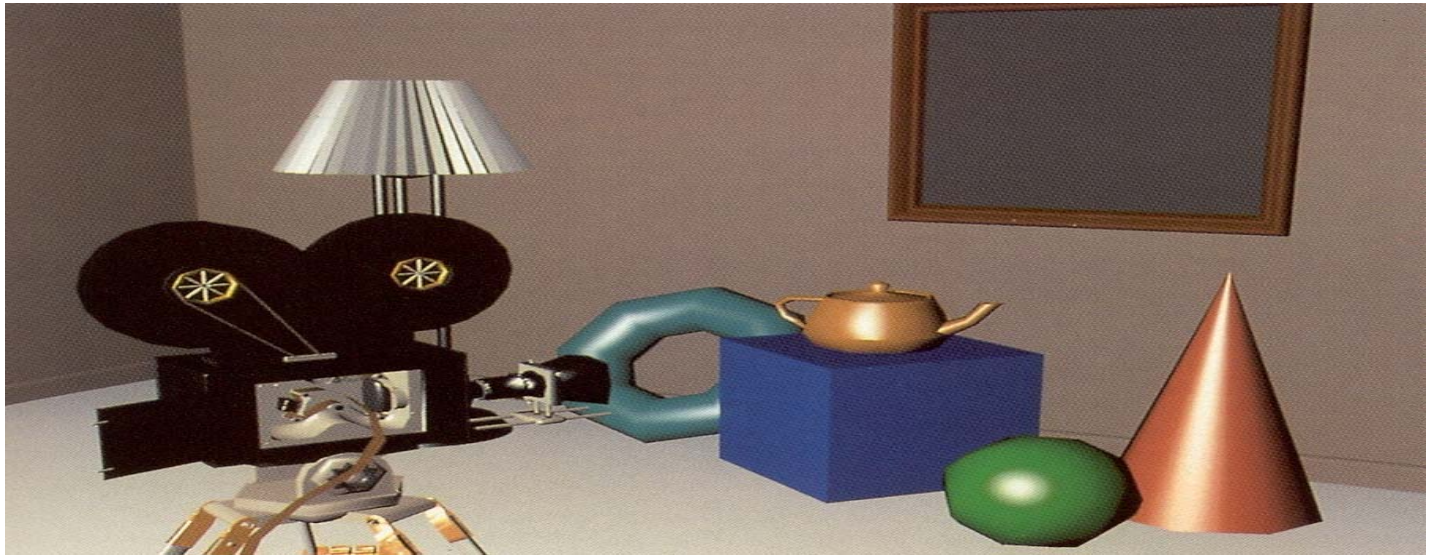
- To get an even smoother result with fewer artifacts, we can perform **Phong *interpolation***.
- Here's how it works:
  1. Compute normals at the vertices.
  2. Interpolate normals and normalize.
  3. Shade using the interpolated normals.





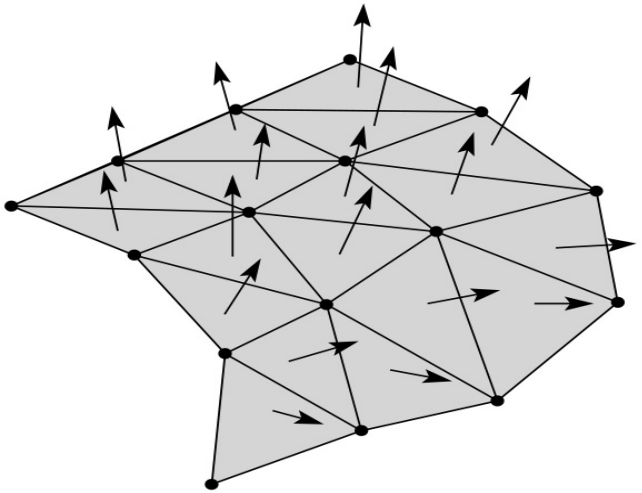
Gouraud vs. Phong interpolation





How to compute vertex normals

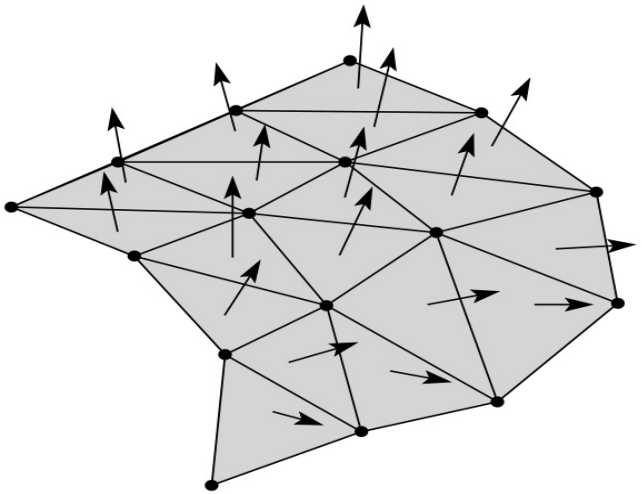
A weighted average of normals of neighboring triangles



$$\mathbf{n}_{vertex} = \frac{\sum_{triangle} area_{triangle}\mathbf{n}_{triangle}}{\sum_{triangle} area_{triangle}}$$

How to compute vertex normals

A weighted average of normals of neighboring triangles



$$\mathbf{n}_{vertex} = \frac{\sum_{triangle} area_{triangle}\mathbf{n}_{triangle}}{\left\| \sum_{triangle} area_{triangle}\mathbf{n}_{triangle} \right\|}$$

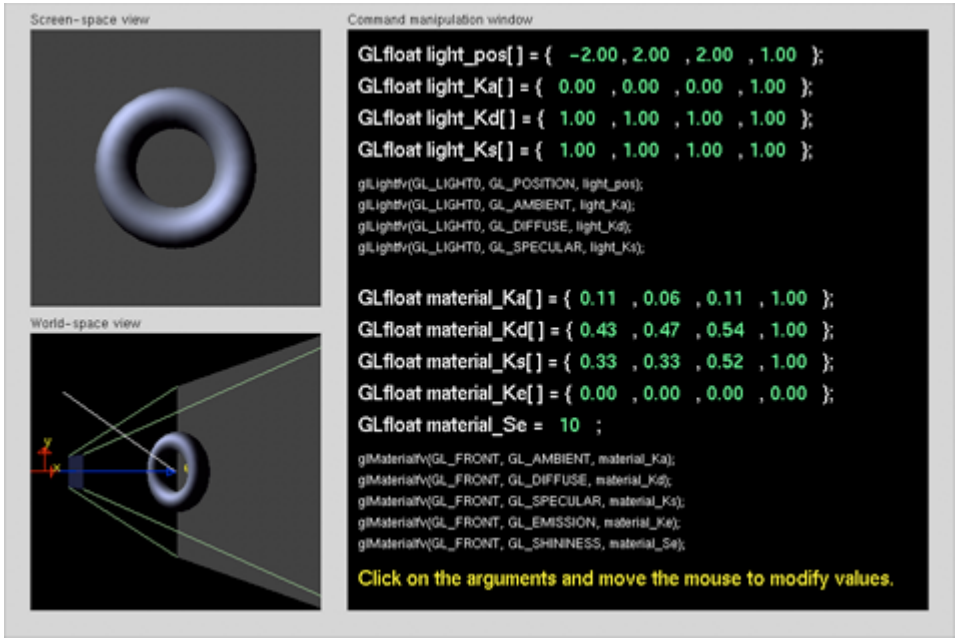
Define a light in OpenGL

```
GLfloat ke[] = { 0.1, 0.15, 0.05, 1.0 };
GLfloat ka[] = { 0.1, 0.15, 0.1, 1.0 };
GLfloat kd[] = { 0.3, 0.3, 0.2, 1.0 };
GLfloat ks[] = { 0.2, 0.2, 0.2, 1.0 };
GLfloat ns[] = { 50.0 };
glMaterialfv(GL_FRONT, GL_EMISSION, ke);
glMaterialfv(GL_FRONT, GL_AMBIENT, ka);
glMaterialfv(GL_FRONT, GL_DIFFUSE, kd);
glMaterialfv(GL_FRONT, GL_SPECULAR, ks);
glMaterialfv(GL_FRONT, GL_SHININESS, ns);
```

$$I = k_e + k_a L_a + k_d L_d \cdot \max(0, \mathbf{L} \cdot \mathbf{N}) + k_s L_s \cdot (\mathbf{H} \cdot \mathbf{N})^{n_s}$$

```
GLfloat light_ambient[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

Demo

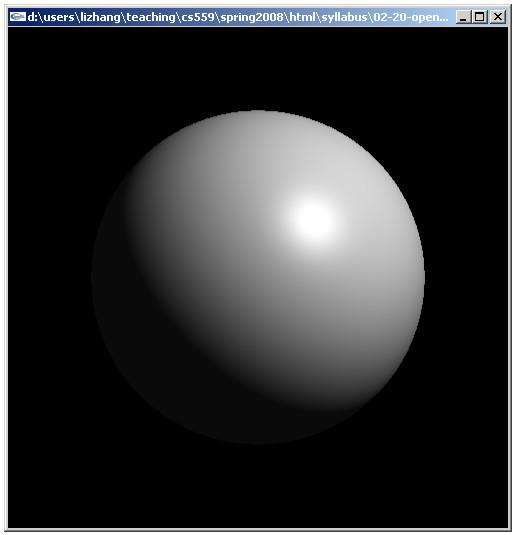

Screen-space view
World-space view
Command manipulation window
GLfloat light_pos[ ] = { -2.00 , 2.00 , 2.00 , 1.00 };
GLfloat light_Ka[ ] = { 0.00 , 0.00 , 0.00 , 1.00 };
GLfloat light_Kd[ ] = { 1.00 , 1.00 , 1.00 , 1.00 };
GLfloat light_Ks[ ] = { 1.00 , 1.00 , 1.00 , 1.00 };
glLightfv(GL_LIGHT0, GL_POSITION, light_pos);
glLightfv(GL_LIGHT0, GL_AMBIENT, light_Ka);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_Kd);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_Ks);
GLfloat material_Ka[ ] = { 0.11 , 0.06 , 0.11 , 1.00 };
GLfloat material_Kd[ ] = { 0.43 , 0.47 , 0.54 , 1.00 };
GLfloat material_Ks[ ] = { 0.33 , 0.33 , 0.52 , 1.00 };
GLfloat material_Ke[ ] = { 0.00 , 0.00 , 0.00 , 0.00 };
GLfloat material_Se = 10 ;
glMaterialfv(GL_FRONT, GL_AMBIENT, material_Ka);
glMaterialfv(GL_FRONT, GL_DIFFUSE, material_Kd);
glMaterialfv(GL_FRONT, GL_SPECULAR, material_Ks);
glMaterialfv(GL_FRONT, GL_EMISSION, material_Ke);
glMaterialfv(GL_FRONT, GL_SHININESS, material_Se);
Click on the arguments and move the mouse to modify values.

Light.c

```
void init(void) {
     GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
     GLfloat mat_shininess[] = { 50.0 };
     GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
     glClearColor (0.0, 0.0, 0.0, 0.0);
     glShadeModel (GL_SMOOTH);
     glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
     glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
     glLightfv(GL_LIGHT0, GL_POSITION, light_position);
     glEnable(GL_LIGHTING);
     glEnable(GL_LIGHT0);
     glEnable(GL_DEPTH_TEST);
}
```

```
void display(void) {
     glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
     glutSolidSphere (1.0, 20, 16);
     glFlush ();
}
```

Light.c


d:\users\lizhang\teaching\cs559\spring2008\html\syllabus\02-20-open...

Complex lighting

- How to have multiple lights?
- How to change lighting positions?
- How to change color material?

Multiple lights

$$I = k_e + k_a L_a + \sum_j \frac{f_{spot_j}}{a_j + b_j D + c_j D^2} \left[k_a L_a + k_d L_d \cdot \max(0, \mathbf{L}_d \cdot \mathbf{N}) + k_s L_s \cdot (\mathbf{H} \cdot \mathbf{N})^{n_s} \right]_j$$

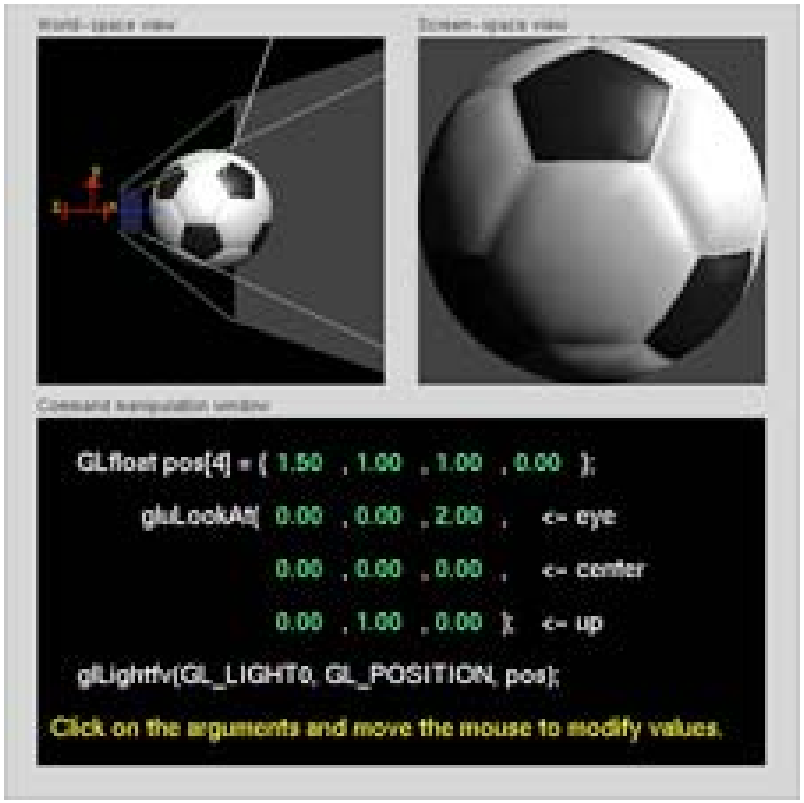
```
GLfloat light1_ambient[] = { 0.2, 0.2, 0.2, 1.0 };
GLfloat light1_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light1_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light1_position[] = { -2.0, 2.0, 1.0, 1.0 };
GLfloat spot_direction[] = { -1.0, -1.0, 0.0 };

glLightfv(GL_LIGHT1, GL_AMBIENT, light1_ambient);
glLightfv(GL_LIGHT1, GL_DIFFUSE, light1_diffuse);
glLightfv(GL_LIGHT1, GL_SPECULAR, light1_specular);

glLightfv(GL_LIGHT1, GL_POSITION, light1_position);
glLightf(GL_LIGHT1, GL_CONSTANT_ATTENUATION, 1.5);
glLightf(GL_LIGHT1, GL_LINEAR_ATTENUATION, 0.5);
glLightf(GL_LIGHT1, GL_QUADRATIC_ATTENUATION, 0.2);

glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, spot_direction);
glLightf(GL_LIGHT1, GL_SPOT_CUTOFF, 45.0);
glLightf(GL_LIGHT1, GL_SPOT_EXPONENT, 2.0);

glEnable(GL_LIGHT1);
```

Moving light source

- Method 1:
  - Use glLightfv(GL_LIGHT1, GL_POSITION, light1_position);
- Method 2:
  - Use transformation

Moving a light source

- Use glLightfv(GL_LIGHT1, GL_POSITION, light1_position);



Moving light source

- Use transformation

```
static GLdouble spin;
void display(void) {
     GLfloat light_position[] = { 0.0, 0.0, 1.5, 1.0 };
     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
     glPushMatrix();
          gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
          glPushMatrix();
               glRotated(spin, 1.0, 0.0, 0.0);
               glLightfv(GL_LIGHT0, GL_POSITION, light_position);
          glPopMatrix();
          glutSolidTorus (0.275, 0.85, 8, 15);
     glPopMatrix();
     glFlush();
}
```

- If we fix spin and change gluLookAt, the lighting will move together with object.
- If we change spin value, light will change independently with the object.

Demo

- Rotating a light source demo.

glColorMaterial

```
glEnable(GL_COLOR_MATERIAL);

glColorMaterial(GL_FRONT, GL_DIFFUSE);
/* now glColor* changes diffuse reflection */

glColor3f(0.2, 0.5, 0.8);
/* draw some objects here */

glColorMaterial(GL_FRONT, GL_SPECULAR);
/* glColor* no longer changes diffuse reflection */
/* now glColor* changes specular reflection */

glColor3f(0.9, 0.0, 0.2);
/* draw other objects here */

glDisable(GL_COLOR_MATERIAL);
```

glColorMaterial

- Demo

```
void mouse(int button, int state, int x, int y) {
     switch (button) {
          case GLUT_LEFT_BUTTON:
               if (state == GLUT_DOWN) {
                    /* change red */
                    diffuseMaterial[0] += 0.1;
                    if (diffuseMaterial[0] > 1.0) diffuseMaterial[0] = 0.0;
                    glColor4fv(diffuseMaterial);
                    glutPostRedisplay();
               }
               break;
          case GLUT_MIDDLE_BUTTON:
               if (state == GLUT_DOWN) {
                    /* change green */
                    diffuseMaterial[1] += 0.1;
                    if (diffuseMaterial[1] > 1.0) diffuseMaterial[1] = 0.0;
                    glColor4fv(diffuseMaterial);
                    glutPostRedisplay();
               }
               break;
          case GLUT_RIGHT_BUTTON:
               if (state == GLUT_DOWN) {
                    /* change blue */
                    diffuseMaterial[2] += 0.1;
                    if (diffuseMaterial[2] > 1.0) diffuseMaterial[2] = 0.0;
                    glColor4fv(diffuseMaterial);
                    glutPostRedisplay();
               }
               break;
          default:
               break;
     }
}
```

Shading in OpenGL, cont’d

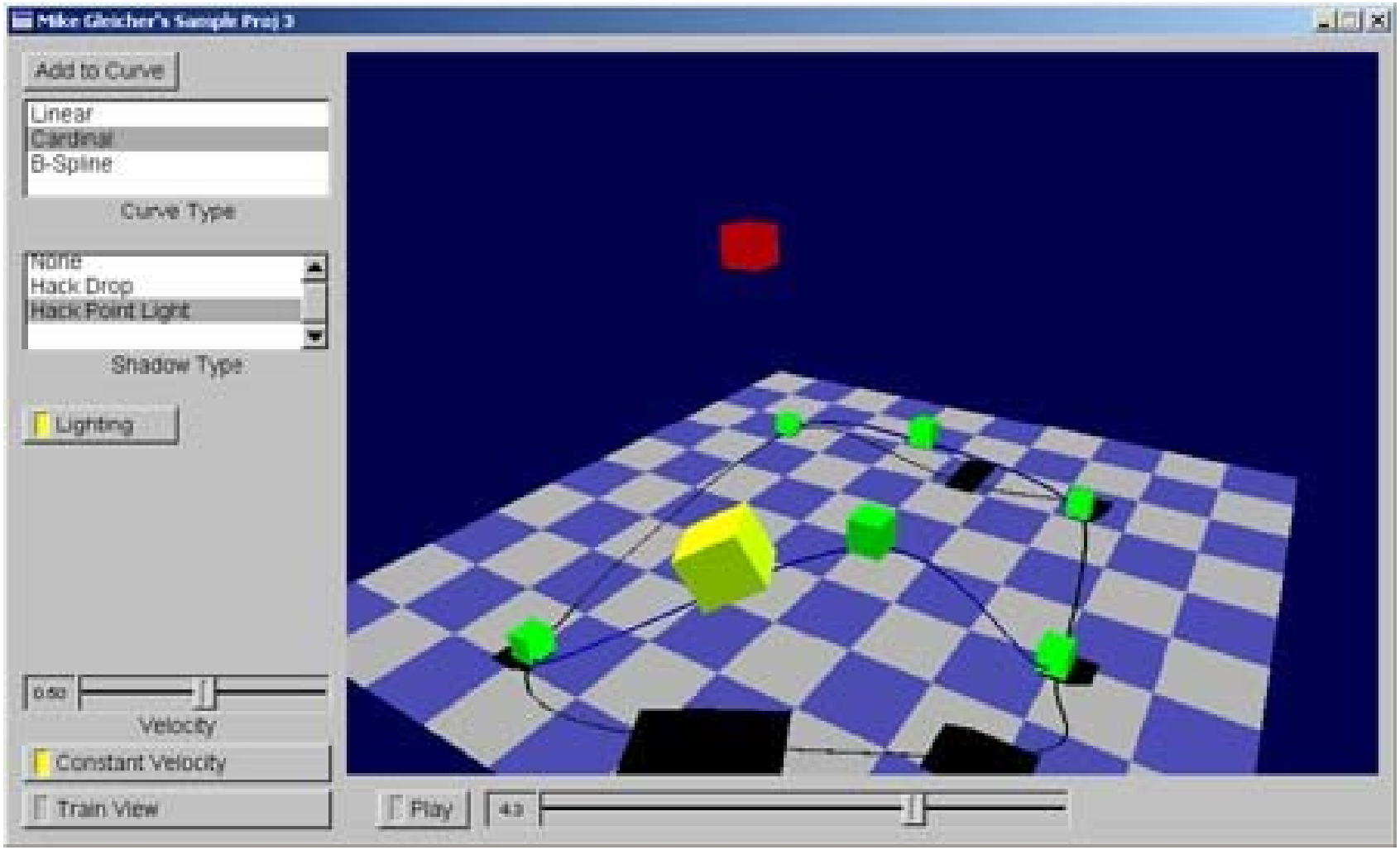
Notes:

You can have as many as GL_MAX_LIGHTS lights in a scene. This number is system-dependent.

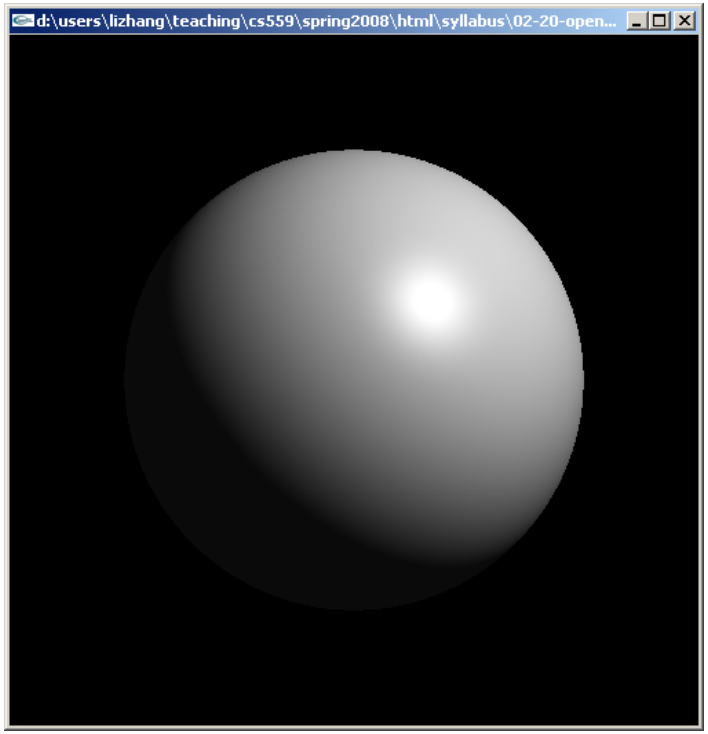
For directional lights, you specify a light direction, not position, and the attenuation and spotlight terms are ignored.

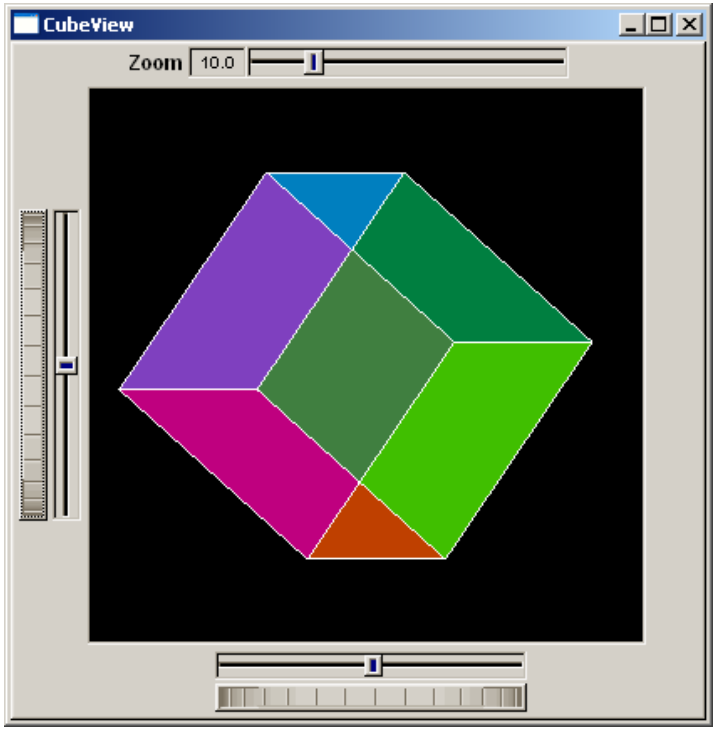
The directions of directional lights and spotlights are specified in the world coordinate systems, not the object coordinate system.

3D User Interface



FLUT vs FLTK

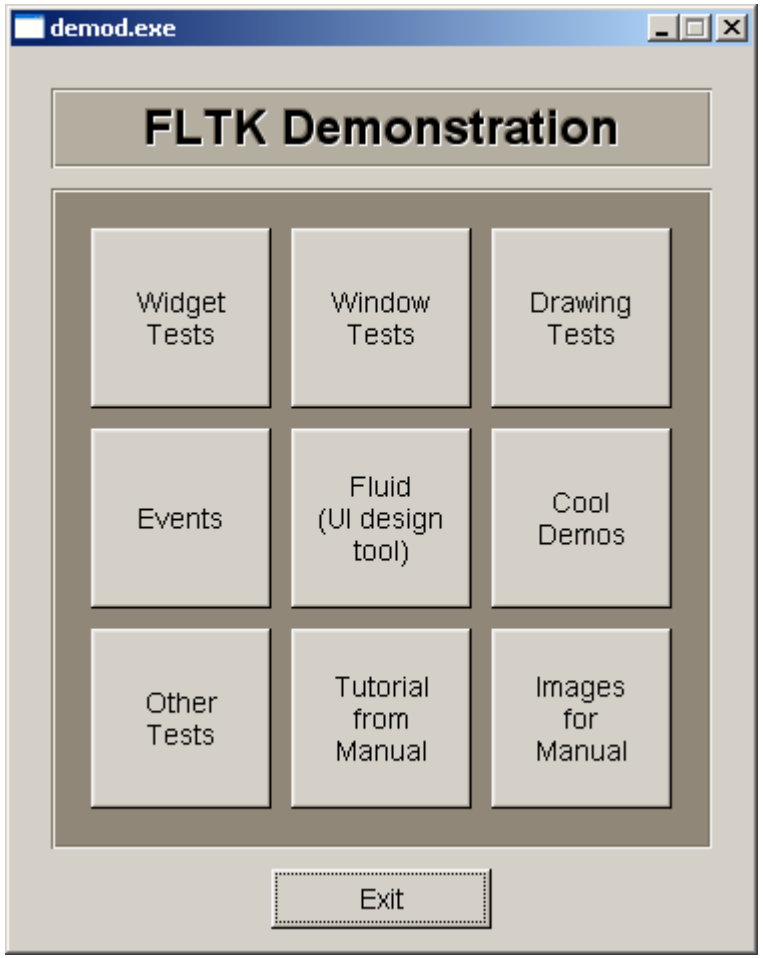

d:\users\lizhang\teaching\cs559\spring2008\html\syllabus\02-20-open...


CubeView
Zoom
10.0

FLTK

- FLTK stands for "The Fast Light Toolkit“
- A cross-platform C++ GUI toolkit for UNIX®/Linux® (X11), Microsoft® Windows®, and MacOS® X.

FLTK Demo


demod.exe
FLTK Demonstration
Widget Tests
Window Tests
Drawing Tests
Events
Fluid (UI design tool)
Cool Demos
Other Tests
Tutorial from Manual
Images for Manual
Exit

FLTK class hierarchy

- Fl
- Fl_End
- Fl_File_Icon
- Fl_Image
  - Fl_Bitmap
    - Fl_XBM_Image
  - Fl_Pixmap
    - Fl_GIF_Image
    - Fl_XPM_Image
  - Fl_RGB_Image
    - Fl_BMP_Image
    - Fl_JPEG_Image
    - Fl_PNG_Image
    - Fl_PNM_Image
  - Fl_Shared_Image
  - Fl_Tiled_Image
- Fl_Menu_Item
- Fl_Preferences
- Fl_Text_Buffer
- Fl_Tooltip
- Fl_Widget
  - Fl_Box
  - Fl_Browser_
    - Fl_Browser
      - Fl_File_Browser
      - Fl_Hold_Browser
      - Fl_Multi_Browser
      - Fl_Select_Browser
    - Fl_Check_Browser

- Fl_Button
  - Fl_Check_Button
  - Fl_Light_Button
  - Fl_Repeat_Button
  - Fl_Return_Button
  - Fl_Round_Button
- Fl_Chart
- Fl_Clock
- Fl_Free
- Fl_Group
  - Fl_Color_Chooser
  - Fl_File_Chooser
  - Fl_Help_Dialog
  - Fl_Help_View
  - Fl_Input_Choice
  - Fl_Pack
  - Fl_Scroll
  - Fl_Slider
  - Fl_Tabs
  - Fl_Text_Display
    - Fl_Text_Editor
  - Fl_Tile
  - Fl_Window
    - Fl_Double_Window
    - Fl_Gl_Window
    - Fl_Menu_Window
    - Fl_Overlay_Window
    - Fl_Single_Window
  - Fl_Wizard

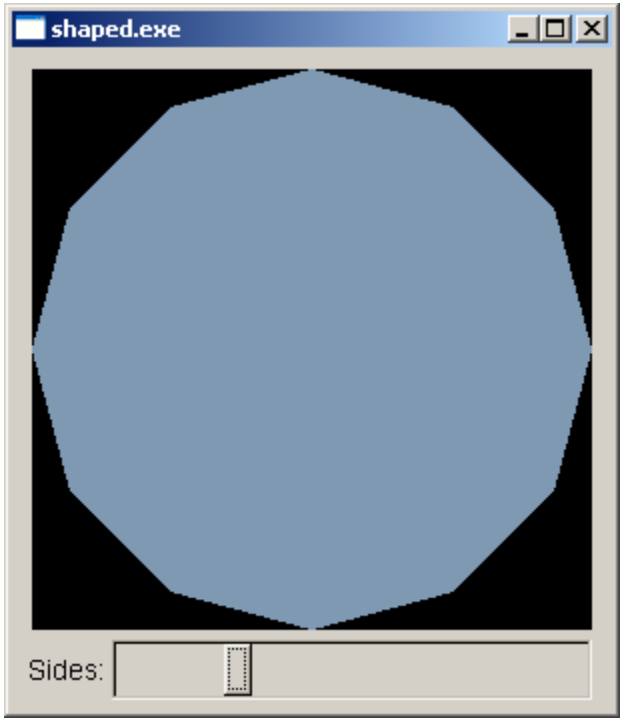
- Fl_Input_
  - Fl_Input
    - Fl_Float_Input
    - Fl_Int_Input
    - Fl_Multiline_Input
    - Fl_Secret_Input
  - Fl_Output
    - Fl_Multiline_Output
- Fl_Menu_
  - Fl_Choice
  - Fl_Menu_Bar
  - Fl_Menu_Button
- Fl_Positioner
- Fl_Progress
- Fl_Timer
- Fl_Valuator
  - Fl_Adjuster
  - Fl_Counter
  - Fl_Dial
  - Fl_Roller
  - Fl_Slider
    - Fl_Scrollbar
    - Fl_Value_Slider
  - Fl_Value_Input
  - Fl_Value_Output

Hello world

Hello world


```
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>
int main(int argc, char **argv)
{
     Fl_Window *window = new Fl_Window(300,180);

     Fl_Box *box = new Fl_Box(20,40,260,100,"Hello, World!");
     box->box(FL_UP_BOX);
     box->labelsize(36);
     box->labelfont(FL_BOLD+FL_ITALIC);
     box->labeltype(FL_SHADOW_LABEL);

     window->end();
     window->show(argc, argv);
     return Fl::run();
}
```

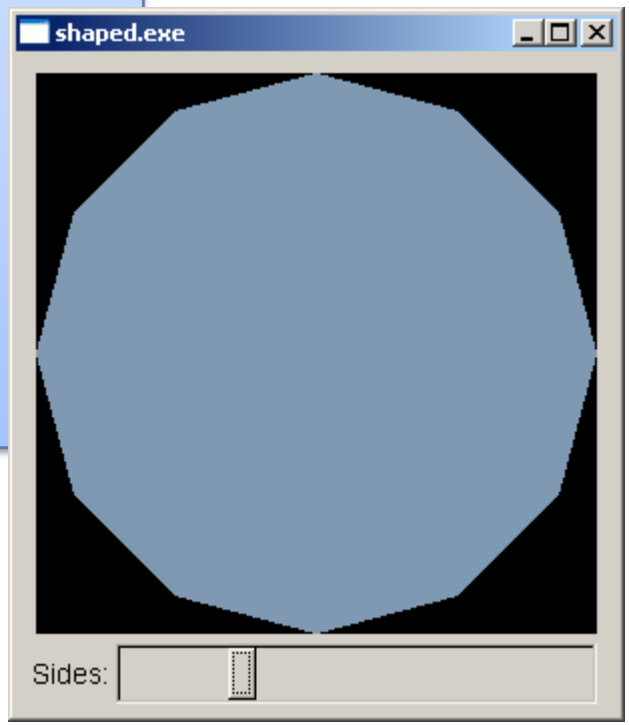
Shape Control



Shape Control

```
int main(int argc, char **argv) {

     Fl_Window window(300, 330);

     shape_window sw(10, 10, 280, 280);

     window.resizable(&sw);

     Fl_Hor_Slider slider(50, 295, window.w()-60, 30, "Sides:");
     slider.align(FL_ALIGN_LEFT);
     slider.callback(sides_cb,&sw);
     slider.value(sw.sides);
     slider.step(1);
     slider.bounds(3,40);

     window.end();
     window.show(argc,argv);

     return Fl::run();
}
```

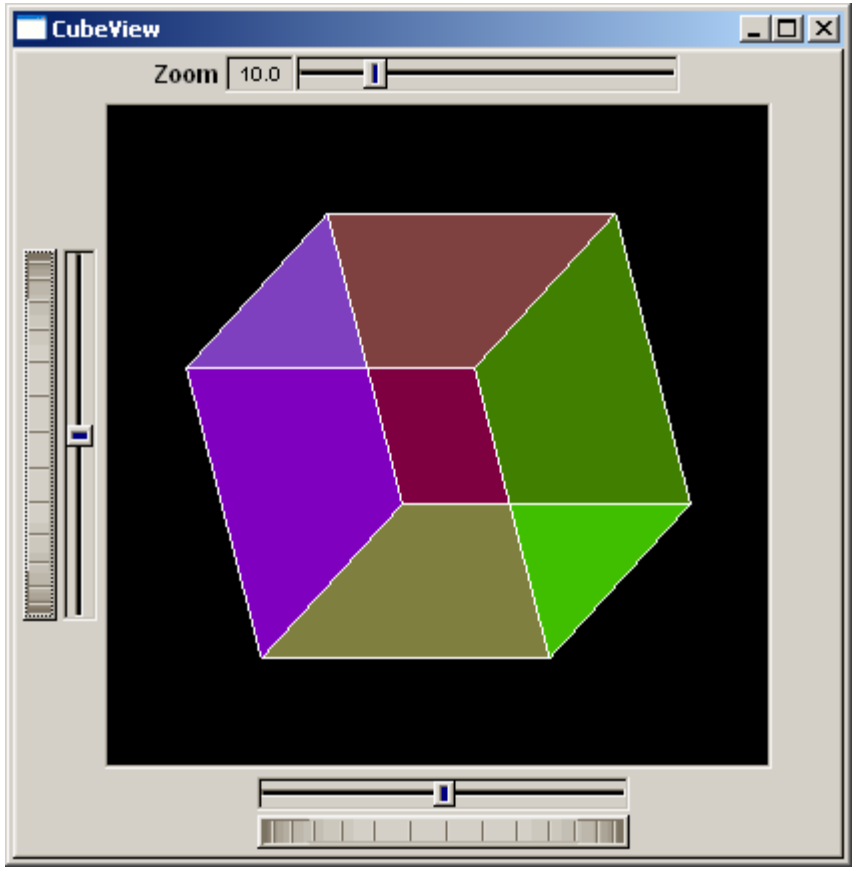
```
void sides_cb(Fl_Widget *o, void *p) {
     shape_window *sw = (shape_window *)p;
     sw->sides = int(((Fl_Slider *)o)->value());
     sw->redraw();
}
```



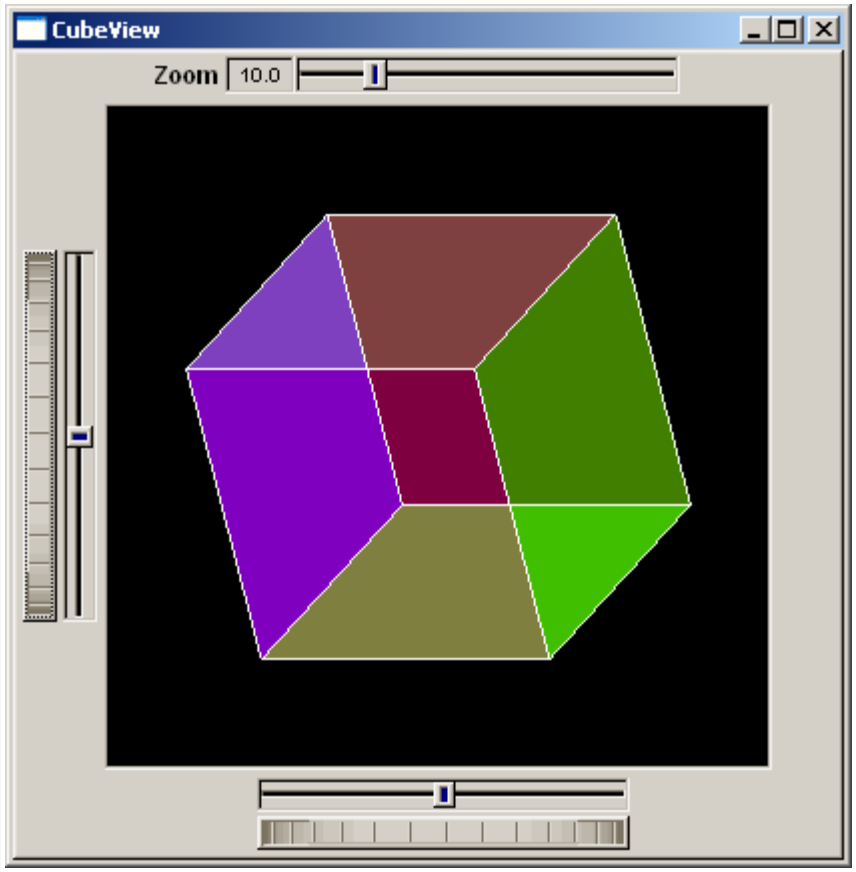
```
class shape_window : public Fl_Gl_Window {
     void draw();
public:
     int sides;
     shape_window(int x,int y,int w,int h,const char *l=0);
};

shape_window::shape_window(int x,int y,int w,int h,const char *l) :
Fl_Gl_Window(x,y,w,h,l) {
     sides = 3;
}

void shape_window::draw() {
     if (!valid()) {
          valid(1);
          glLoadIdentity();
          glViewport(0, 0, w(), h());
     }

     glClear(GL_COLOR_BUFFER_BIT);
     glColor3f(.5,.6,.7);
     glBegin(GL_POLYGON);
     for (int i=0; i<sides; i++) {
          double ang = i*2*M_PI/sides;
          glVertex3f(cos(ang),sin(ang),0);
     }
     glEnd();
}
```

The Cube example

```
#include "config.h"
#include <FL/Fl.H>
#include "CubeViewUI.h"

int main(int argc, char **argv) {

    CubeViewUI *cvui=new CubeViewUI;

    Fl::visual(FL_DOUBLE|FL_RGB);

    cvui->show(argc, argv);

    return Fl::run();
}
```

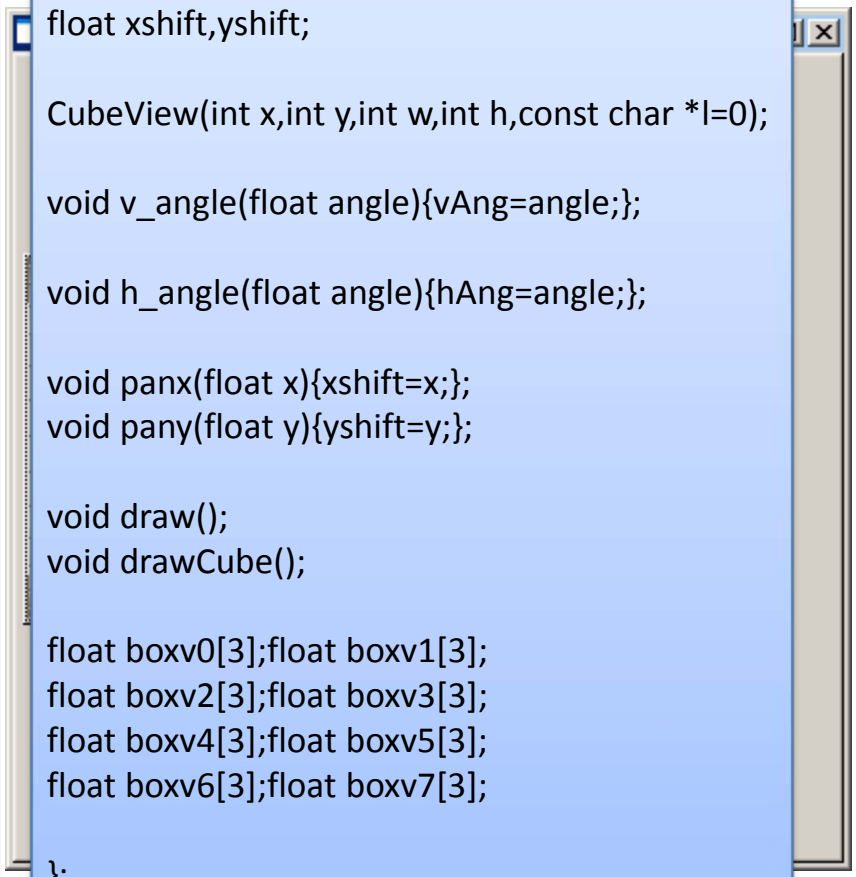


```
class CubeViewUI {
public:
  CubeViewUI();
private:
  Fl_Double_Window *mainWindow;

public:
  Fl_Roller *vrot;
  Fl_Slider *ypan;
private:
  void cb_vrot_i(Fl_Roller*, void*);
  void cb_ypan_i(Fl_Slider*, void*);

public:
  Fl_Slider *xpan;
  Fl_Roller *hrot;
private:
  void cb_xpan_i(Fl_Slider*, void*);
  void cb_hrot_i(Fl_Roller*, void*);

public:
  CubeView *cube;
  Fl_Value_Slider *zoom;
private:
  void cb_zoom_i(Fl_Value_Slider*, void*);

public:
  void show(int argc, char **argv);
};
```



```
class CubeViewUI {
public:
  CubeViewUI();
private:
  Fl_Double_Window *mainWindow;

public:
  Fl_Roller *vrot;
  Fl_Slider *ypan;
private:
  void cb_vrot_i(Fl_Roller*, void*);
  void cb_ypan_i(Fl_Slider*, void*);

public:
  Fl_Slider *xpan;
  Fl_Roller *hrot;
private:
  void cb_xpan_i(Fl_Slider*, void*);
  void cb_hrot_i(Fl_Roller*, void*);

public:
  CubeView *cube;
  Fl_Value_Slider *zoom;
private:
  void cb_zoom_i(Fl_Value_Slider*, void*);

public:
  void show(int argc, char **argv);
};
```

```
class CubeView : public Fl_Gl_Window {

public:
double size;
float vAng,hAng;
float xshift,yshift;

CubeView(int x,int y,int w,int h,const char *l=0);

void v_angle(float angle){vAng=angle;};

void h_angle(float angle){hAng=angle;};

void panx(float x){xshift=x;};
void pany(float y){yshift=y;};

void draw();
void drawCube();

float boxv0[3];float boxv1[3];
float boxv2[3];float boxv3[3];
float boxv4[3];float boxv5[3];
float boxv6[3];float boxv7[3];

};
```



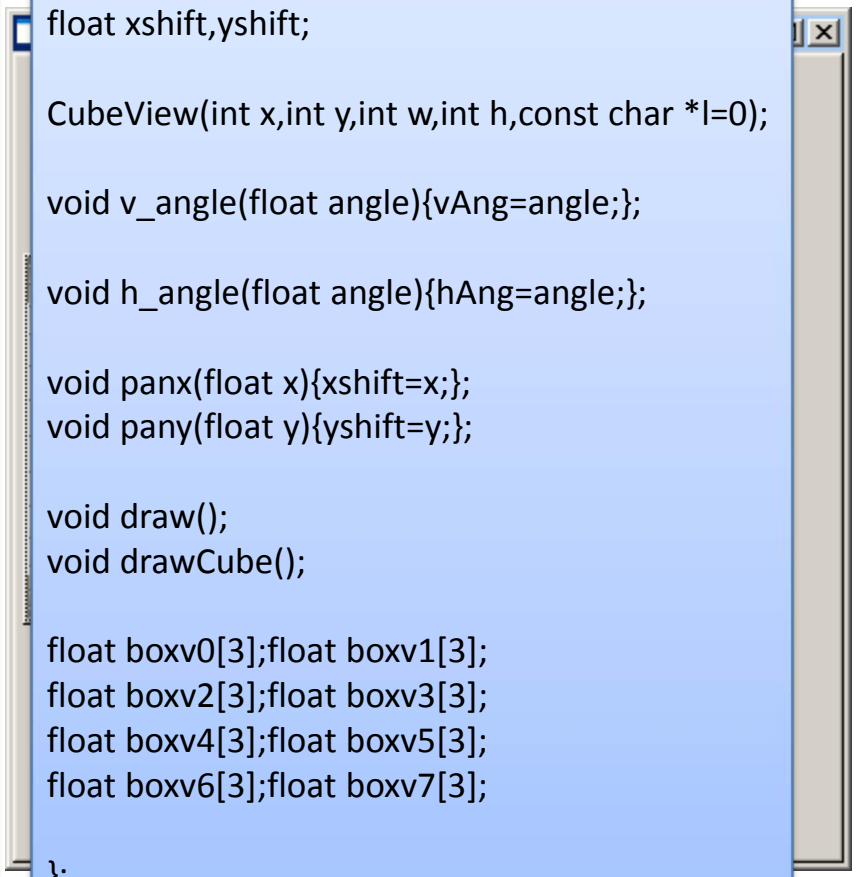
```
class CubeViewUI {
public:
  CubeViewUI();
private:
public:
  void show(int argc, char **argv);
};
```

```
class CubeView : public Fl_Gl_Window {

public:
ng;
ift;

x,int y,int w,int h,const char *l=0);

loat angle){vAng=angle;};

loat angle){hAng=angle;};

t x){xshift=x;};
t y){yshift=y;};

e();

;float boxv1[3];
;float boxv3[3];
;float boxv5[3];
;float boxv7[3];

};
```

```
void CubeView::draw() {
    if (!valid()) {
        glLoadIdentity();
        glViewport(0,0,w(),h());
        glOrtho(-10,10,-10,10,-20050,10000);
        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    }

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glPushMatrix();

    glTranslatef(xshift, yshift, 0);
    glRotatef(hAng,0,1,0); glRotatef(vAng,1,0,0);
    glScalef(float(size),float(size),float(size));

    drawCube();

    glPopMatrix();
}
```

```
class CubeViewUI {
public:
  CubeViewUI();
private:
  Fl_Double_Window *mainWindow;

public:
  Fl_Roller *vrot;
  Fl_Slider *ypan;
private:
  void cb_vrot_i(Fl_Roller*, void*);
  void cb_ypan_i(Fl_Slider*, void*);

public:
  Fl_Slider *xpan;
  Fl_Roller *hrot;
private:
  void cb_xpan_i(Fl_Slider*, void*);
  void cb_hrot_i(Fl_Roller*, void*);

public:
  CubeView *cube;
  Fl_Value_Slider *zoom;
private:
  void cb_zoom_i(Fl_Value_Slider*, void*);

public:
  void show(int argc, char **argv);
};
```

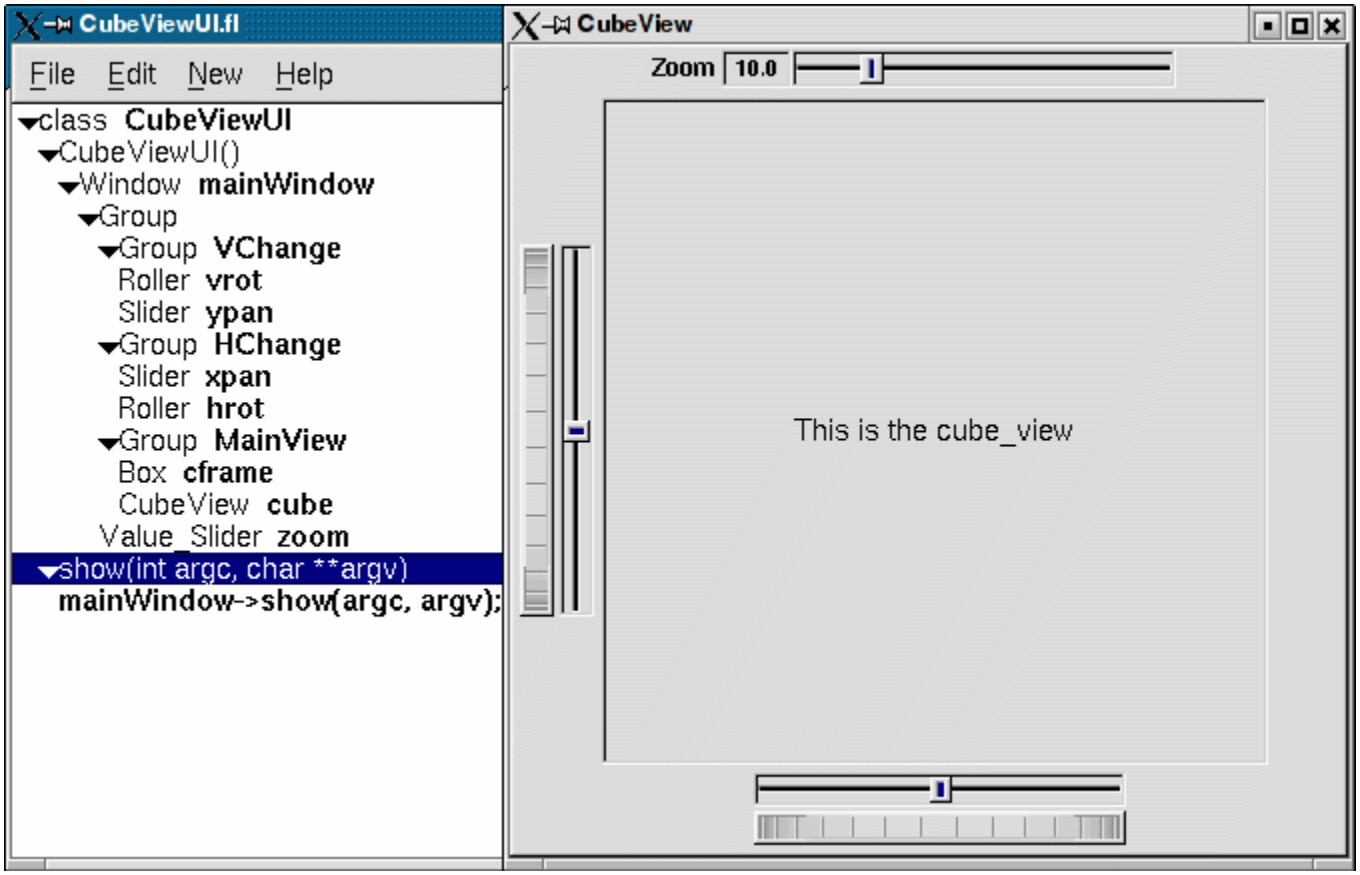
```
class CubeView : public Fl_Gl_Window {

public:
double size;
float vAng,hAng;
float xshift,yshift;

CubeView(int x,int y,int w,int h,const char *l=0);

void v_angle(float angle){vAng=angle;};

void h_angle(float angle){hAng=angle;};

void panx(float x){xshift=x;};
void pany(float y){yshift=y;};

void draw();
void drawCube();

float boxv0[3];float boxv1[3];
float boxv2[3];float boxv3[3];
float boxv4[3];float boxv5[3];
float boxv6[3];float boxv7[3];

};
```



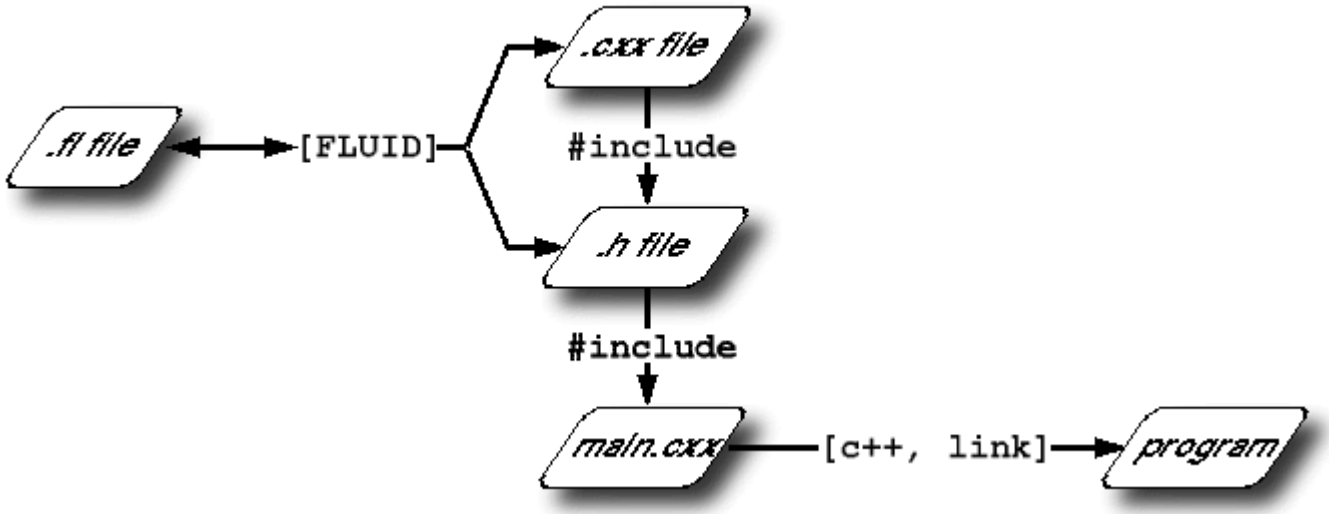
Initialize window layout

```
CubeViewUI::CubeViewUI() {
  Fl_Double_Window* w;
  { Fl_Double_Window* o = mainWindow = new Fl_Double_Window(415, 405, "CubeView");
    w = o;
    o->box(FL_UP_BOX);
    o->labelsize(12);
    o->user_data((void*)(this));
    { Fl_Group* o = new Fl_Group(5, 3, 374, 399);
      { Fl_Group* o = VChange = new Fl_Group(5, 100, 37, 192);
        { Fl_Roller* o = vrot = new Fl_Roller(5, 100, 17, 186, "V Rot");
          o->labeltype(FL_NO_LABEL);
          o->labelsize(12);
          o->minimum(-180);
          o->maximum(180);
          o->step(1);
          o->callback((Fl_Callback*)cb_vrot);
          o->align(FL_ALIGN_WRAP);
        }
        { Fl_Slider* o = ypan = new Fl_Slider(25, 100, 17, 186, "V Pan");
          o->type(4);
          o->selection_color(FL_DARK_BLUE);
          o->labeltype(FL_NO_LABEL);
          o->labelsize(12);
          o->minimum(-25);
          o->maximum(25);
          o->step(0.1);
          o->callback((Fl_Callback*)cb_ypan);
          o->align(FL_ALIGN_CENTER);
        }
        o->end();
      }
      { Fl_Group* o = HChange = new Fl_Group(120, 362, 190, 40);
        { Fl_Slider* o = xpan = new Fl_Slider(122, 364, 186, 17, "H Pan");
          o->type(5);
          o->selection_color(FL_DARK_BLUE);
          o->labeltype(FL_NO_LABEL);
          o->labelsize(12);
          o->minimum(25);
```

FLUID


CubeViewUI.fl
File Edit New Help
class CubeViewUI
CubeViewUI()
Window mainWindow
Group
Group VChange
Roller vrot
Slider ypan
Group HChange
Slider xpan
Roller hrot
Group MainView
Box cframe
CubeView cube
Value_Slider zoom
show(int argc, char **argv)
mainWindow->show(argc, argv);
CubeView
Zoom
10.0
This is the cube_view

FLUID


.fl file
[FLUID]
.cxx file
#include
.h file
#include
main.cxx
[c++, link]
program

Demo