




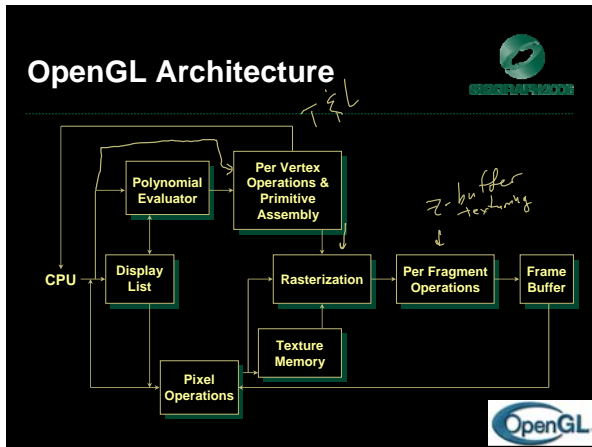

## An Interactive Introduction to OpenGL Programming

Ed Angel  
Dave Shreiner  
Vicki Shreiner


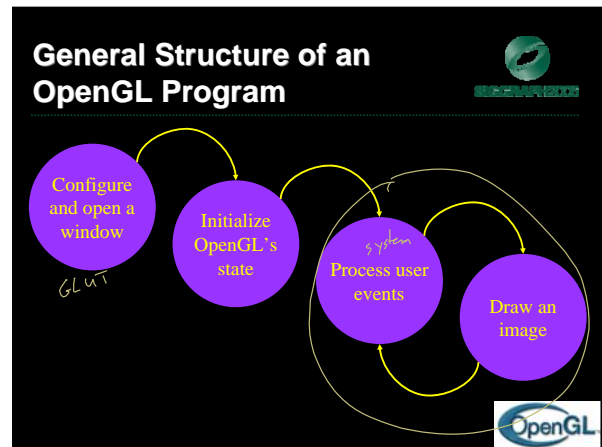
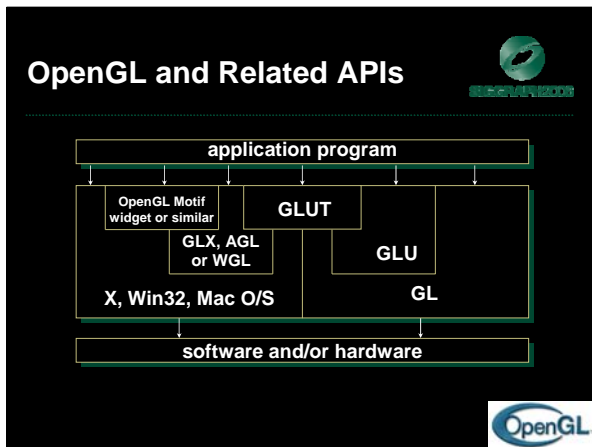
## What Is OpenGL?

- Graphics rendering API
  - high-quality color images composed of geometric and image primitives
  - window system independent ✓
  - operating system independent ✓

## Related APIs


- GLU (OpenGL Utility Library)
  - part of OpenGL
  - NURBS, tessellators, quadric shapes, etc.
- AGL, GLX, WGL
  - glue between OpenGL and windowing systems
- GLUT (OpenGL Utility Toolkit)
  - portable windowing API
  - not officially part of OpenGL


## An OpenGL Program

```
#include <GL/glut.h>
#include "cube.h"

void main( int argc, char *argv[] )
{
    glutInit( &argc, argv );
    glutInitDisplayMode( GLUT_RGBA |
                      GLUT_DEPTH );
    glutCreateWindow( argv[0] );
    init();
    glutDisplayFunc( display );
    glutReshapeFunc( reshape );
    glutMainLoop();
}
```



The main part of the program. GLUT is used to open the OpenGL window, and handle input from the user.




## An OpenGL Program (cont'd.)

```
void init( void )
{
    glClearColor( 0, 0, 0, 1 );
    gluLookAt( 2, 2, 0, 0, 0, 0, 1, 0 );
    glEnable( GL_DEPTH_TEST );
}

void reshape( int width, int height )
{
    glViewport( 0, 0, width, height );
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    gluPerspective( 60, (GLfloat) width / height,
                  1.0, 10.0 );
    glMatrixMode( GL_MODELVIEW );
}
```

Set up some initial OpenGL state


Handle when the user resizes the window



## An OpenGL Program (cont'd.)

```
void display( void )
{
    int i, j;
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    glBegin( GL_QUADS );
    for ( i = 0; i < NUM_CUBE_FACES; ++i ) {
        glColor3fv( faceColor[i] );
        for ( j = 0; j < NUM_VERTICES_PER_FACE; ++j ) {
            glVertex3fv( vertex[face[i][j]] );
        }
    }
    glEnd();
    glFlush();
}
```


Have OpenGL draw a cube from some 3D points (vertices)



## OpenGL Command Formats


### glVertex3fv( v )

- Number of components**
  - 2 - (x,y)
  - 3 - (x,y,z)
  - 4 - (x,y,z,w)
- Data Type**
  - b - byte
  - ub - unsigned byte
  - s - short
  - us - unsigned short
  - i - int
  - ui - unsigned int
  - f - float
  - d - double
- Vector**
  - omit "v" for scalar form
  - glVertex2f( x, y )



## GLUT Basics


- Application Structure:
  - Configure and open window ✓
  - Initialize OpenGL state ✓
  - Register input callback functions
  - Enter event processing loop



## GLUT Callback Functions


- Routine to call when something happens
  - window resize or redraw
  - user input
  - animation
- "Register" callbacks with GLUT
 

```
glutDisplayFunc( display );
glutIdleFunc( idle );
glutKeyboardFunc( keyboard );
```



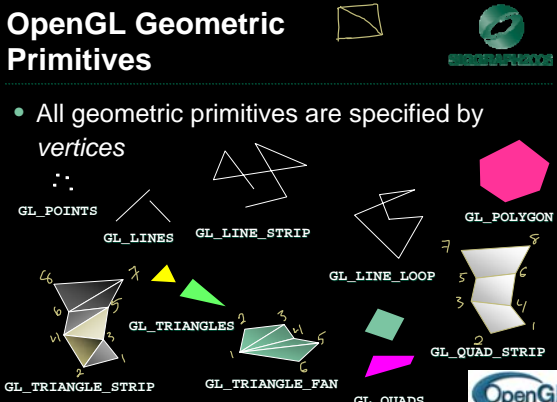

## What can OpenGL Draw?

- Geometric primitives
  - points, lines and polygons
- Image Primitives
  - images and bitmaps
  - separate pipeline for images and geometry
    - linked through texture mapping
- Rendering depends on state
  - colors, materials, light sources, etc.



## OpenGL Geometric Primitives

- All geometric primitives are specified by *vertices*

## Specifying Geometric Primitives

- Primitives are specified using
 


```

vertices
glBegin( primType );
glEnd();
            
```

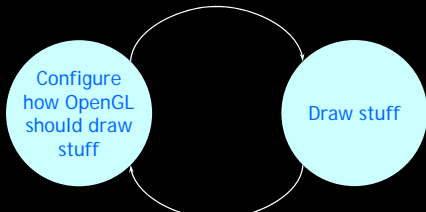

  - primType* determines how vertices are combined

```

glBegin( primType );
for ( i = 0; i < n; ++i ) {
  glColor3f( red[i], green[i], blue[i] );
  glVertex3fv( coords[i] );
}
glEnd();
            
```




## How OpenGL Works: The Conceptual Model

## Controlling OpenGL's Drawing

- Set OpenGL's rendering state
  - State controls how things are drawn
    - shading ✓
    - texture maps ✓
    - polygon patterns ✓
    - lighting ✓
    - line styles (*stipples*) ✓
    - transparency ✓



## The Power of Setting OpenGL State

Appearance is controlled by setting OpenGL's state.




## Setting OpenGL State

- Three ways to set OpenGL state:
  1. Set values to be used for processing vertices
    - most common methods of setting state
      - \* - `glColor()` / `glIndex()`  $(R, G, B, A)$
      - `glNormal()`  $N_x, N_y, N_z, 1$
      - \* - `glTexCoord()`  $(S, T, R, Q)$
    - state must be set before calling `glVertex()`



## Setting OpenGL State (cont'd.)

2. Turning on a rendering mode  
`glEnable()` / `glDisable()`
3. Configuring the specifics of a particular rendering mode
  - Each mode has unique commands for setting its values  
`glMaterialfv()`



## OpenGL and Color

- The OpenGL Color Model
  - OpenGL uses the *RGB* color space
    - There is also a color-index mode, but we do not discuss it
- Colors are specified as floating-point numbers in the range [ 0.0, 1.0 ]
  - for example, to set a window's background color, you would call  
`glClearColor( 1.0, 0.3, 0.6, 1.0 );`



## Shapes Tutorial

The screenshot shows a window titled "Shapes" with two panes. The left pane, "Screen-space view", displays a 2D plot with x and y axes ranging from 0 to 200. A quadrilateral is drawn with vertices at (0,0), (200,0), (175,200), and (50,200). The quadrilateral is filled with a gradient of colors: purple at the bottom-left, green at the top-left, yellow at the top-right, and red at the bottom-right. The right pane, "Command manipulation window", contains the following OpenGL code:

```
glBegin( GL_TRIANGLE_STRIP );
glColor3f( 1.00, 0.00, 1.00 );
glVertex2f( 0.0, 25.0 );
glColor3f( 0.00, 1.00, 1.00 );
glVertex2f( 50.0, 150.0 );
glColor3f( 0.00, 1.00, 0.00 );
glVertex2f( 125.0, 100.0 );
glColor3f( 1.00, 1.00, 0.00 );
glVertex2f( 175.0, 200.0 );
glEnd();
```



## Switch to lighting

