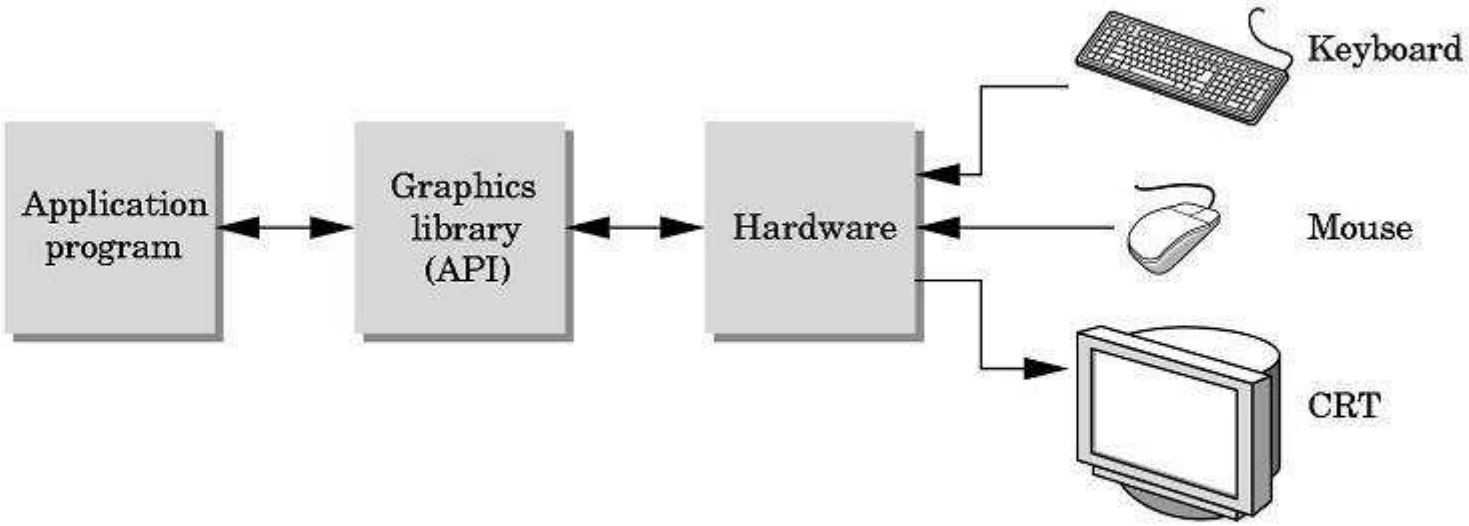
Introduction to

OPEN GRAPHICS LIBRARY (OPENGL)

Graphics API

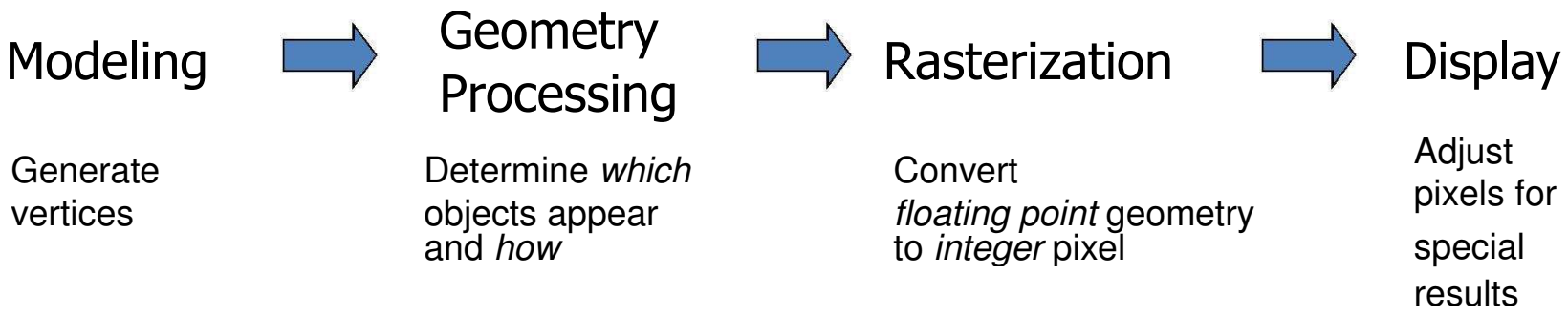
- Interactive CG system that allows programmers to access graphics hardware
  - Easy to use
  - Programs run efficiently
  - Hardware-independent
- Graphics API (Application Programmer's Interface)
  - > A library of functions
  - > Others: DirectX (Microsoft), Java3D
  - > OGL evolved from GL (SGI)

Major Elements of a CG App



Major Elements of a CG App

- Recall the *Viewing Pipeline* …



- Our focus: Modeling and Geometric Processing

  Rasterization & display operations are mostly done for you or allow for special effects

Major Elements of a CG App

- Flow of your basic CG apps will be
  - Initializing functions (os and windowing)
  - Input, interactive functions
  - Specify a set of objects to render
  - Describe properties of these objects
  - Define how these objects should be viewed
  - Termination (os, windowing)

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** (**G**raphics **L**ibrary)
  - Lowest level: vertex, matrix manipulation
  - glVertex3f(point.x, point.y, point.z)
- **GLU (GL U**tilities)
  - Helper functions for shapes, transformations
  - gluPerspective(fovy, aspect, near, far)
- **GLUT** (**GL U**tility **T**oolkit)
  - Highest level: Window and interface management
  - glutSwapBuffers()

OpenGL Implementations

- OpenGL IS an API (think of as collection of .h files):

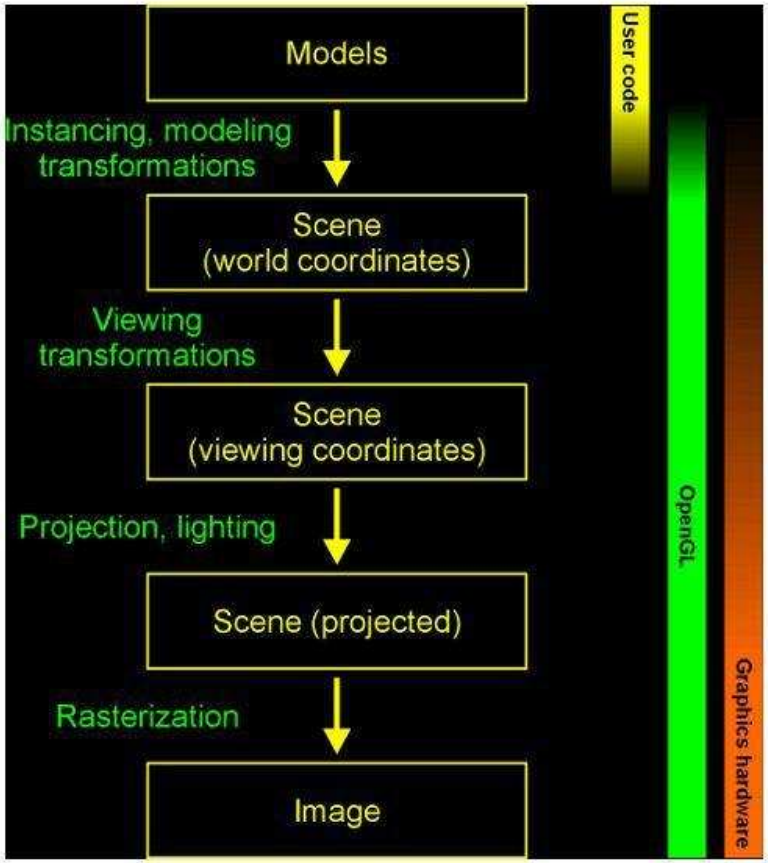
```
#include <GL/gl.h>

#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

Attributes

- Attributes are part of the OpenGL and determine the appearance of objects
  - Color (points, lines, polygons)
  - Size and width (points, lines)
  - Stipple pattern (lines, polygons)
  - Polygon mode
    - Display as filled: solid color or stipple pattern
    - Display edges

Rendering Process

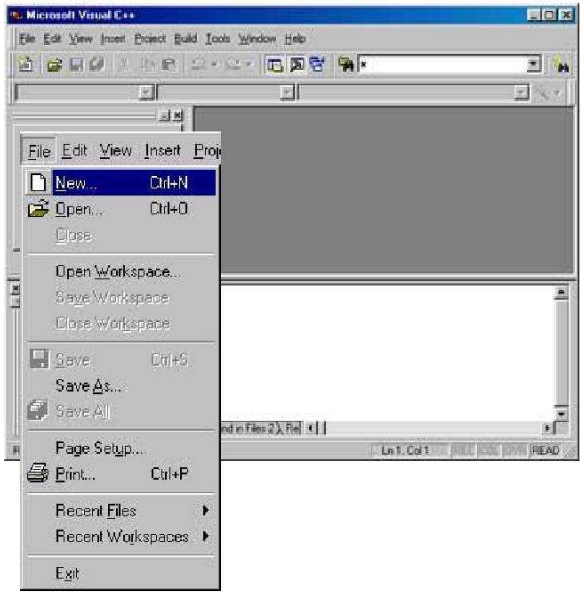


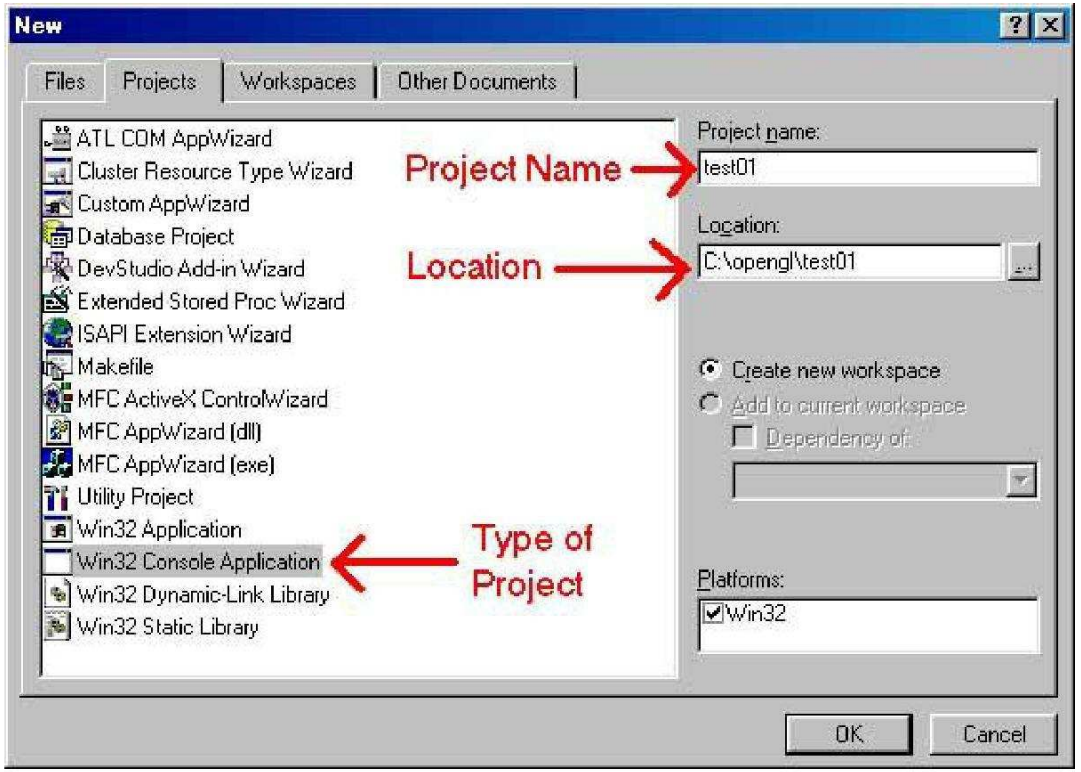
OpenGL: Setup in Windows

- Windows/VC++ 6.0
  - Get glut: http://www.xmission.com/~nate/glut.html
  - An excellent source for the setup of openGL on Windows/VC++ 6.0 is available at
    - http://www.lighthouse3d.com/opengl/glut/index.php

OpenGL: Setup in Windows

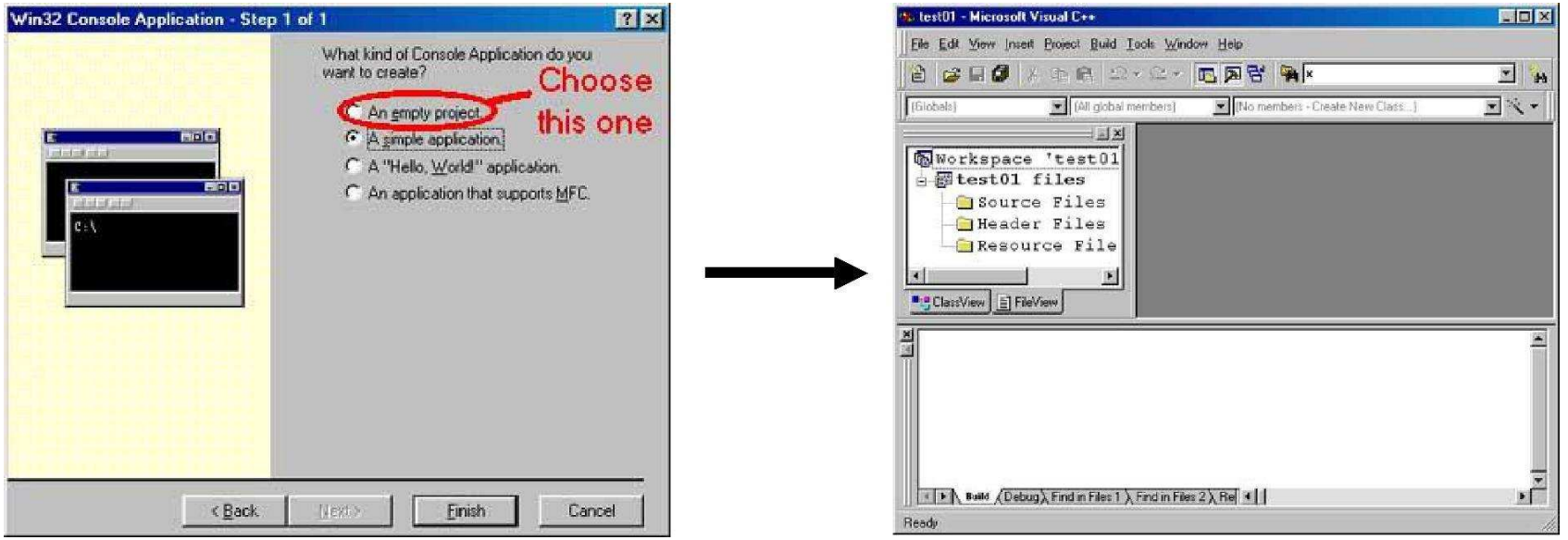
- Windows (step 1: create new project)





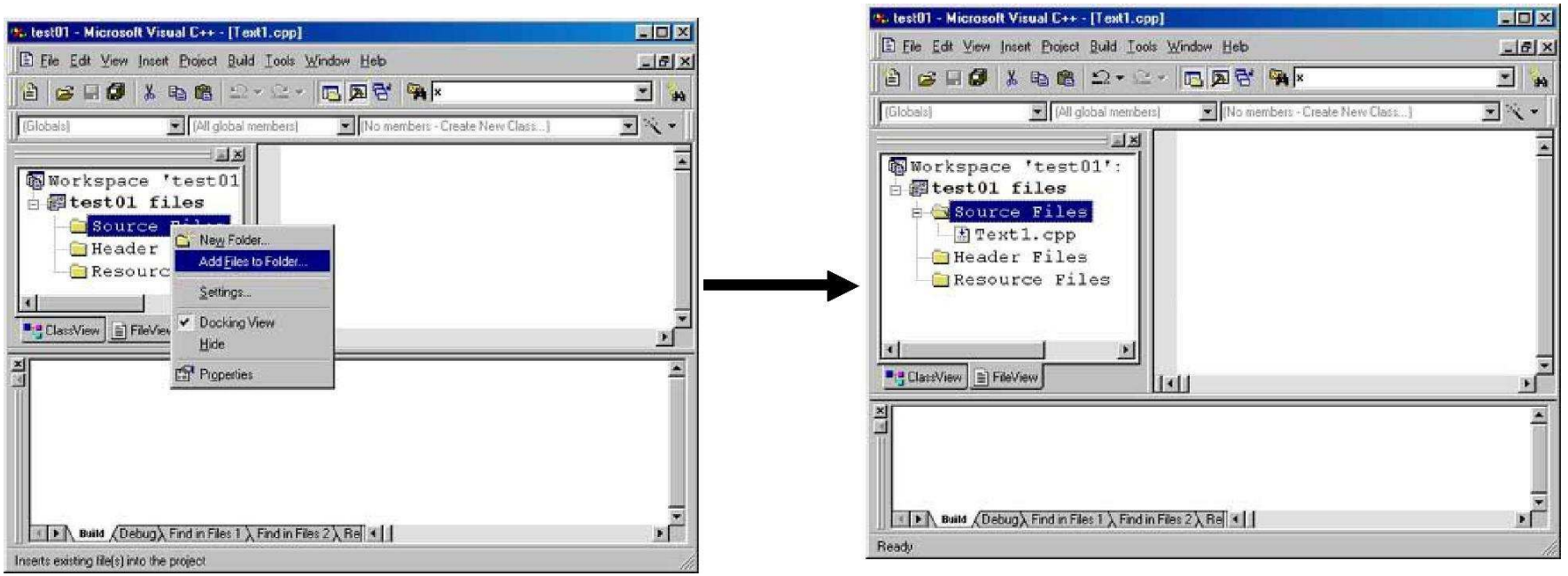
OpenGL: Setup in Windows

- Windows (step 2: create new project)



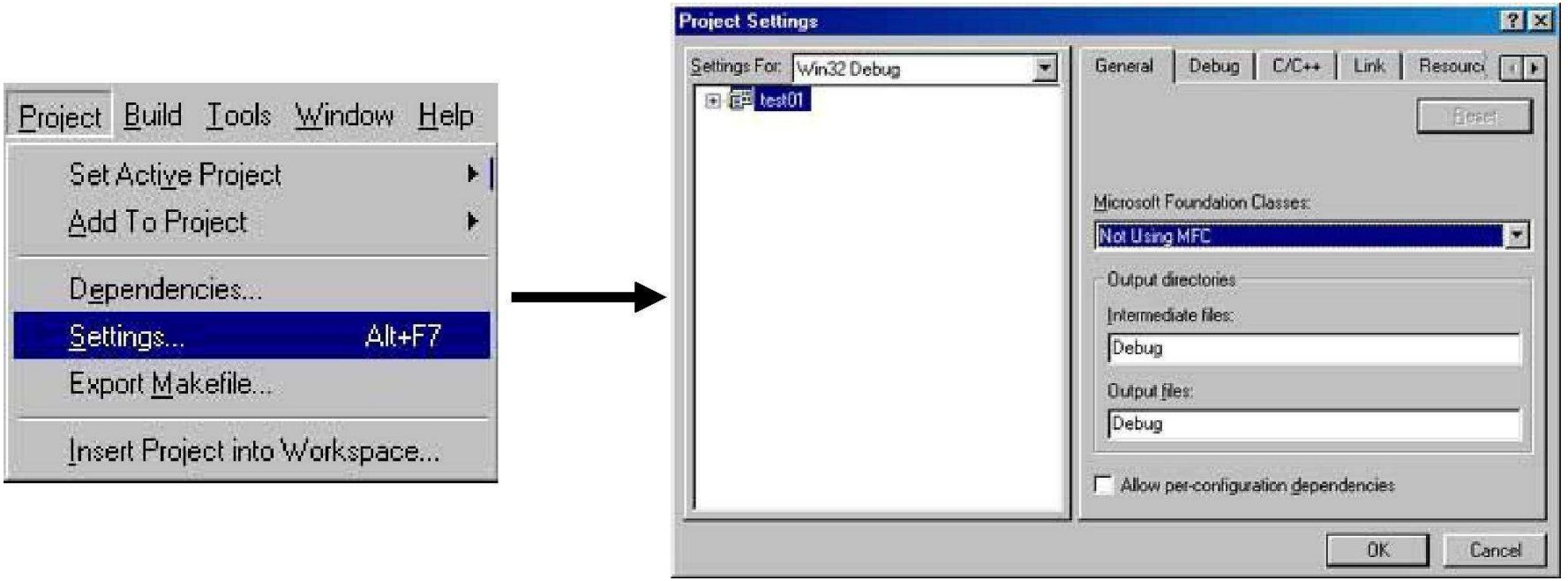
OpenGL: Setup in Windows

- Windows (step 3: insert file)



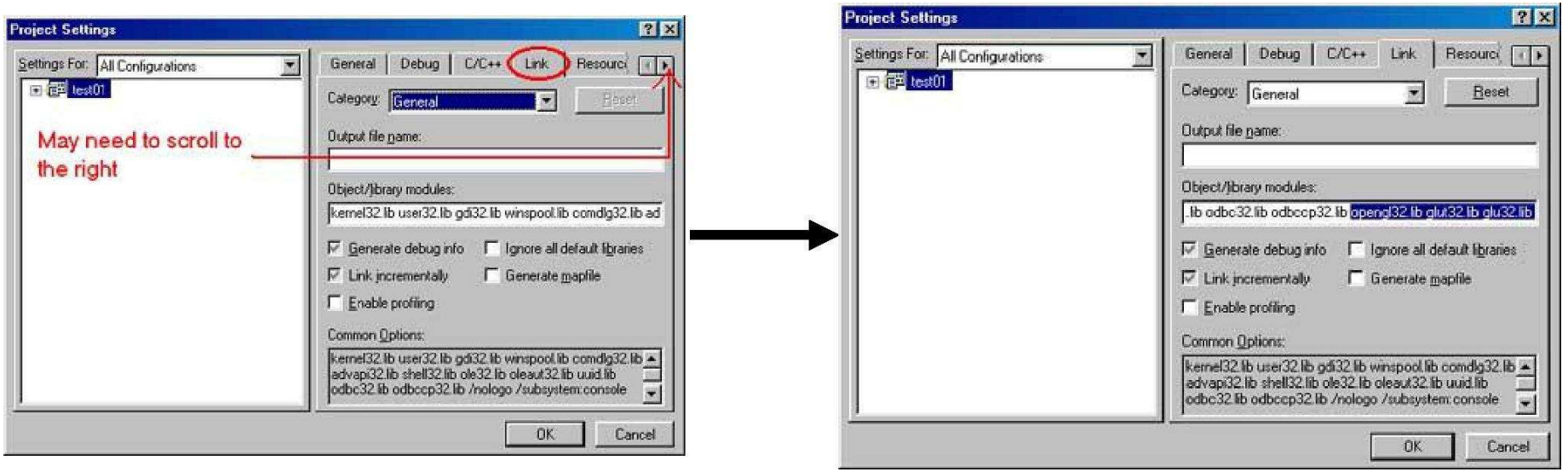
OpenGL: Setup in Windows

- Windows (step 4: project setting)



OpenGL: Setup in Windows

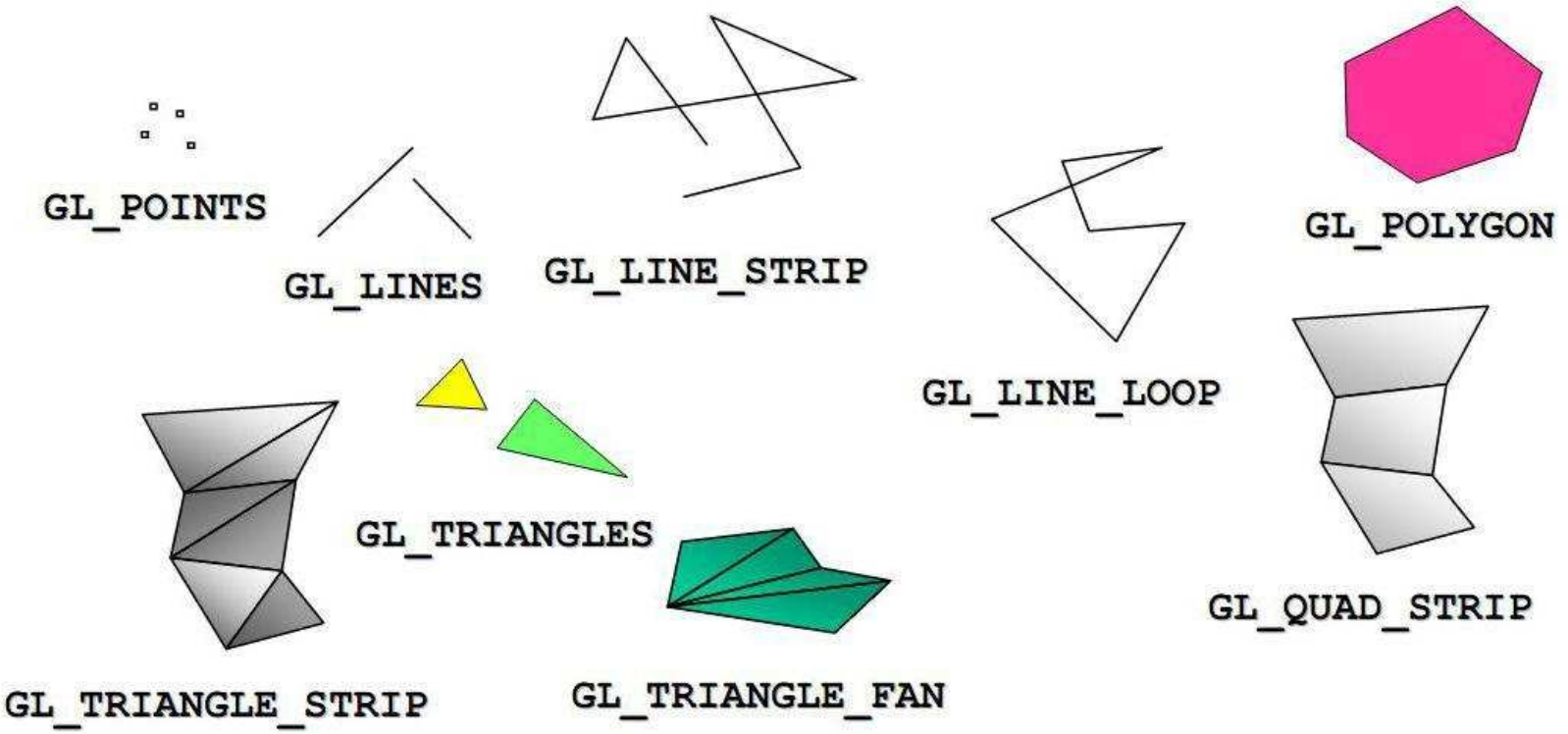
- Windows (step 5: project setting)



Draw

GEOMETRY PRIMITIVE

OpenGL Primitives



Specifying Geometric Primitives

- Primitives are specified using

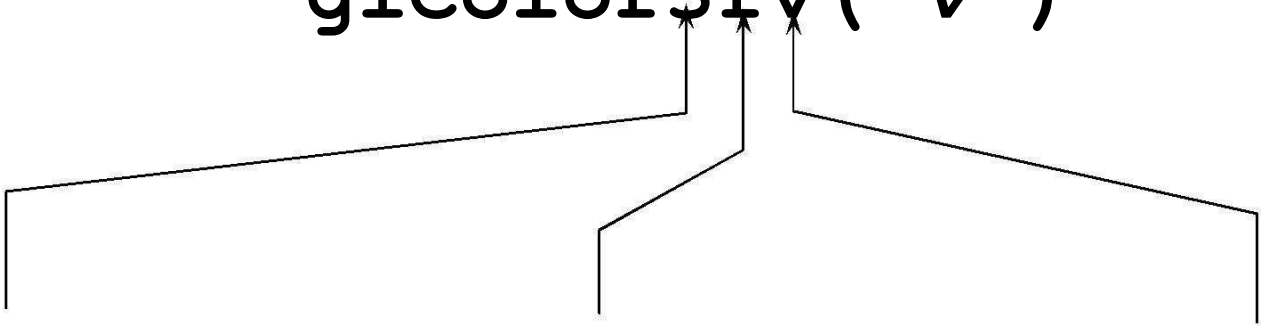
```
glBegin(primType);

...

glEnd();
```

  - *primType* determines how vertices are combined

```
GLfloat red, green, blue;

GLfloat x, y;

glBegin(primType);

for (i = 0; i < nVerts; i++) {
  glColor3f(red, green, blue);
  glVertex2f(x, y);
  ... // change coord. values
}
glEnd();
```

OpenGL Vertex/Color Command Formats

```
glVertex3fv( v )
glColor3fv( v )
```



Number of components

- 2 - (x,y)
- 3 - (x,y,z), (r,g,b)
- 4 - (x,y,z,w), (r,g,b,a)

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-

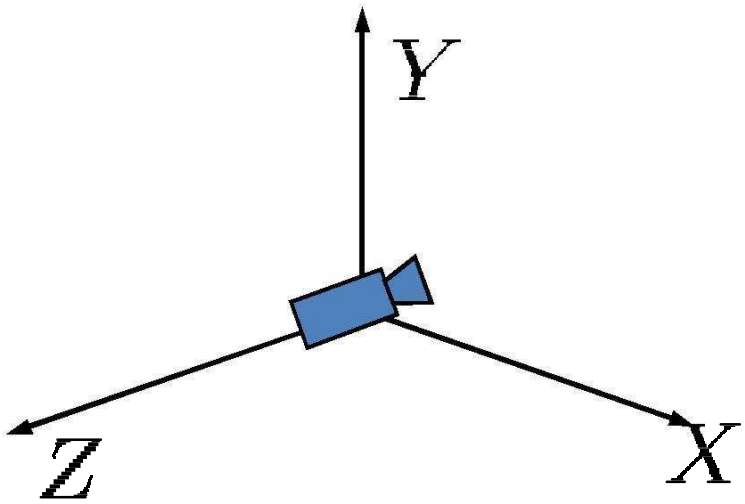
e.g.,
glVertex2f(x, y)
glColor3f(r, g, b)

Introduction

TRANSFORMATIONS IN OPENGL

OpenGL 3-D coordinates

- Right-handed system
- From point of view of camera looking out into scene:
  - +X right, -X left
  - +Y up, -Y down
  - +Z **behind** camera, -Z in front



- Positive rotations are counterclockwise around axis of rotation

Transformations in OpenGl

- Modeling transformation
- Viewing transformation
- Projection transformation

Modeling Transformation

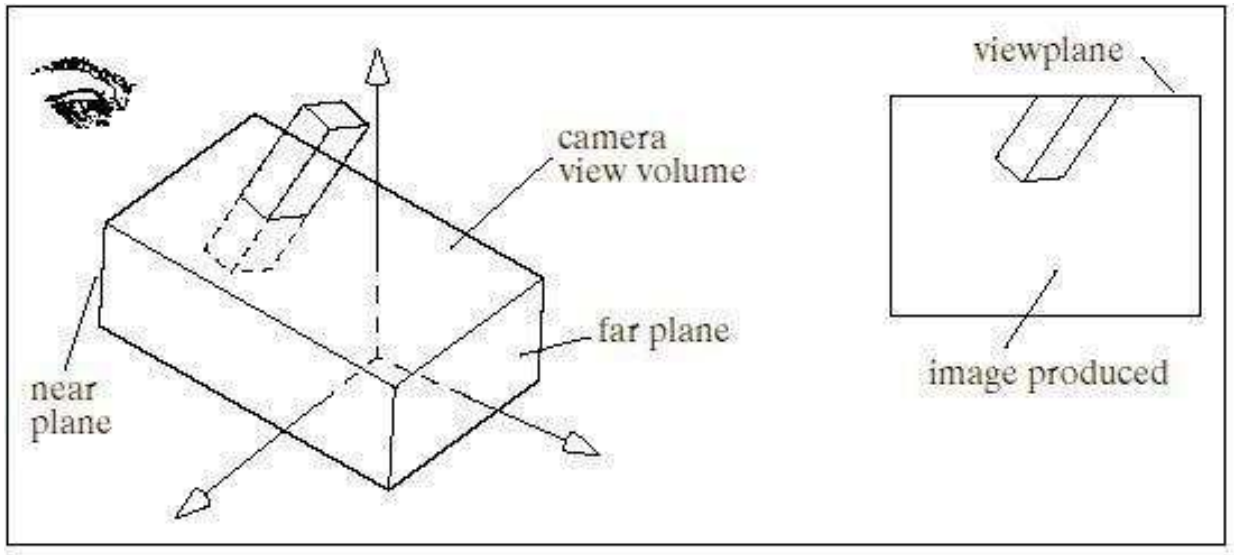
- Refer to the transformation of models (i.e., the scenes, or objects)
- Generally,
  - glMultMatrixf(M_i)
- Some simple transformations
  - Translation: glTranslate(x,y,z)
  - Scale: glScale(sx,sy,sz)
  - Rotation: glRotate(theta, x,y,z)
    - x,y,z are components of vector defining axis of rotation
    - Angle in degrees; direction is counterclockwise

Viewing Transformation

- Refer to the transformation on the camera
- Using glTranslate*() and glRotate*()
- Using gluLookAt()
  - gluLookAt (eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ)
    - **eye** = $(\text{eyeX}, \text{eyeY}, \text{eyeZ})^T$: Desired camera position
    - **center** = $(\text{centerX}, \text{centerY}, \text{centerZ})^T$: Where camera is looking
    - **up** = $(\text{upX}, \text{upY}, \text{upZ})^T$: Camera's "up" vector

Projection Transformation

- Refer to the transformation from scene to image
- Orthographic projection
  - glOrtho (left, right, bottom, top, near, far)



Notes on OpenGL transformations

- Before applying modeling or viewing transformations, need to set

```
glMatrixMode(GL_MODELVIEW)
```

- Before applying projection transformations, need to set

```
glMatrixMode(GL_Projection)
```

- Replacement by either following commands

```
glLoadIdentity();
glLoadMatrix(M);
```

- Multiple transformations (either in modeling or viewing) are applied in **reverse** order