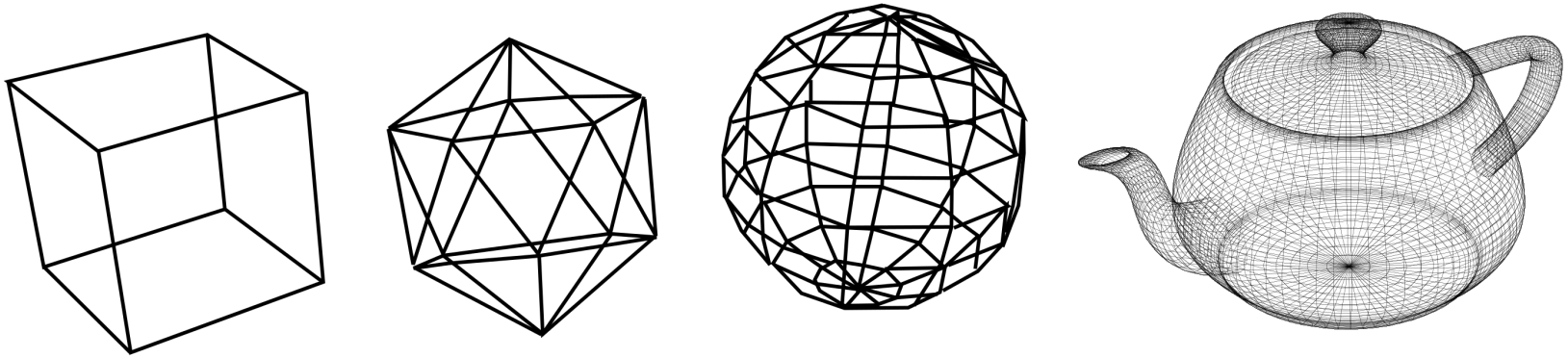
VIETNAM ACADEMY OF SCIENCE AND TECHNOLOGY
UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI

COMPUTER GRAPHICS

Lecture 7: OpenGL

Lecturer: Dr. NGUYEN Hoang Ha



Reference: Chapter 8-9 Principles and Practice (3rd Edition), Addison-Wesley, 2014

OpenGL Overview

What is OpenGL?

- A software interface to graphics hardware
- Low level graphics rendering API for High-quality color images
- Philosophy:
  - Operating system independent
  - Platform independent
  - Window system independent
  - Rendering only
  - Aims to be realtime
  - Takes advantage of graphics hardware where it exists
  - Client-server system
  - Standard supported by major companies

E[illegible] H[illegible] A[illegible]

- IFIPS (1973) formed two committees to come up with a standard graphics API
  - G[illegible] (G[illegible])
    - 2D [illegible]
  - G[illegible] A[illegible] (1980s)
- GKS not easily extended to 3D (GKS-3D)
- Far behind hardware development

PHIGS

- Programmers Hierarchical Graphics System (PHIGS)
  - A
  - CAD
  - D
- X Window System
  - DEC/
  - C
- PEX combined the two
  -

SGI and GL

- Silicon Graphics (SGI) revolutionized the graphics workstation by implementing the pipeline in hardware (1982)
- To use the system, application programmers used a library called GL
- With GL, it was relatively simple to program three dimensional interactive applications

OpenGL

- The success of GL lead to OpenGL (1992), a platform-independent API that was
  - Easy to use
  - Close enough to the hardware to get excellent performance
  - Focus on rendering
  - Omitted windowing and input to avoid window system dependencies
- OpenGL is controlled by OpenGL Architectural Review Board (ARB), members include SGI, IBM, Nvidia, HP, 3D labs, Microsoft.
- Since 2003 OpenGL supports vertex/pixel shaders

□□□G □F □□□□□□□□

- Texture mapping
- z-buffering
- Double buffering
- Lighting effects
- Smooth shading
- Material properties
- Alpha blending
- Transformation matrices

- C#: The framework Tao for Microsoft .NET includes OpenGL between other multimedia libraries
- Delphi: Dot[4]
- Fortran: f90gl supports OpenGL 1.2, GLU 1.2 and GLUT 3.7 [5]
- Java:
- Java Bindings for OpenGL (JSR 231) and Java OpenGL (JOGL)
- Lightweight Java Game Library (LWJGL)
- Lisp: See the cl-opengl project at common-lisp.net
- Perl:
- Perl OpenGL (POGL) module - shared libs written in C
- C vs Perl and Perl vs Python benchmarks
- PHP: See http://phpopengl.sourceforge.net/
- Python: PyOpenGL supports GL, GLU and GLUT [8]
- Ruby: See [9] - supports GL, GLU and GLUT
- Smalltalk as seen in Croquet Project running on Squeak Smalltalk
- Visual Basic: ActiveX Control

Programming with OpenGL

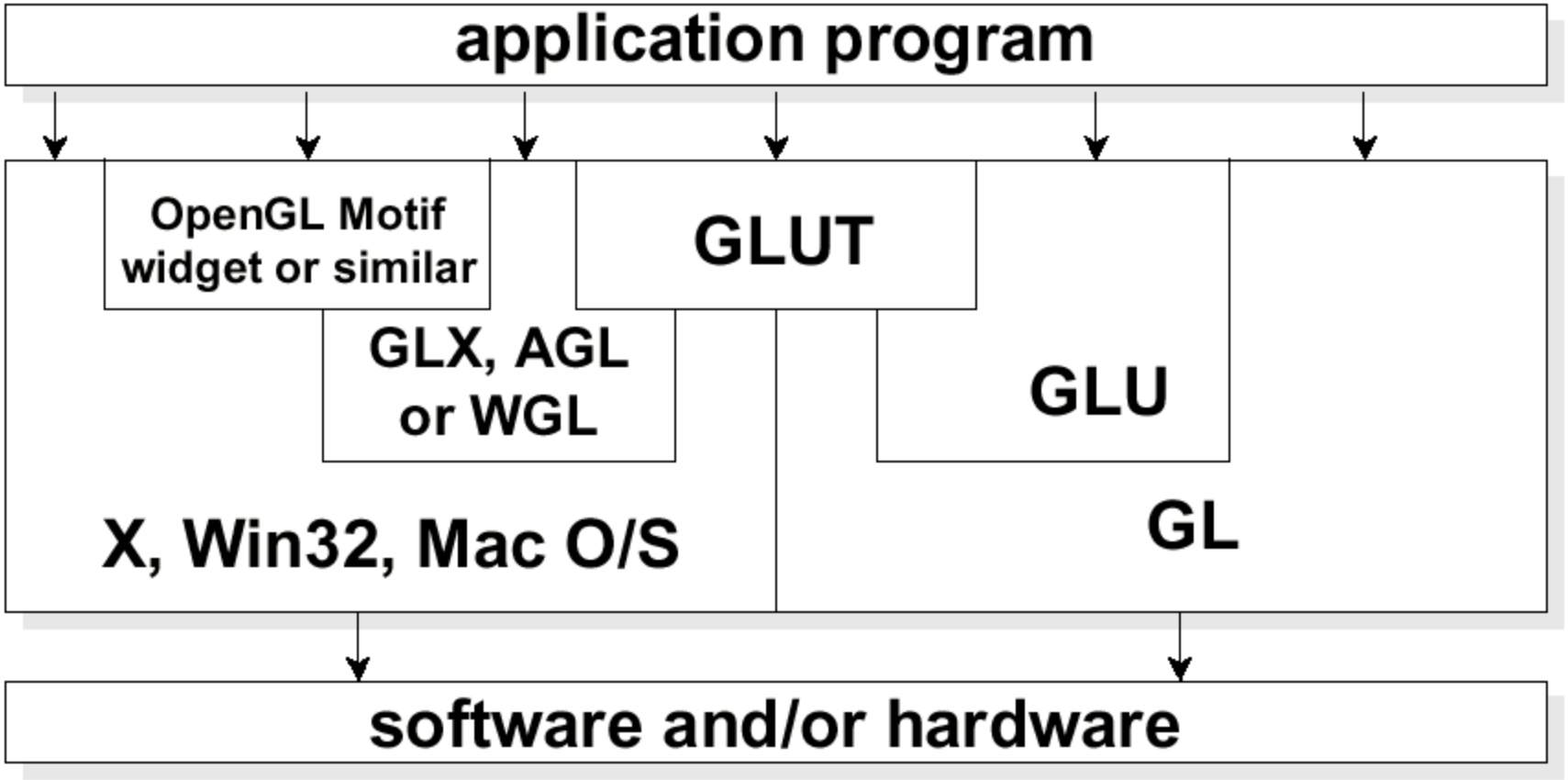
Related A□□□

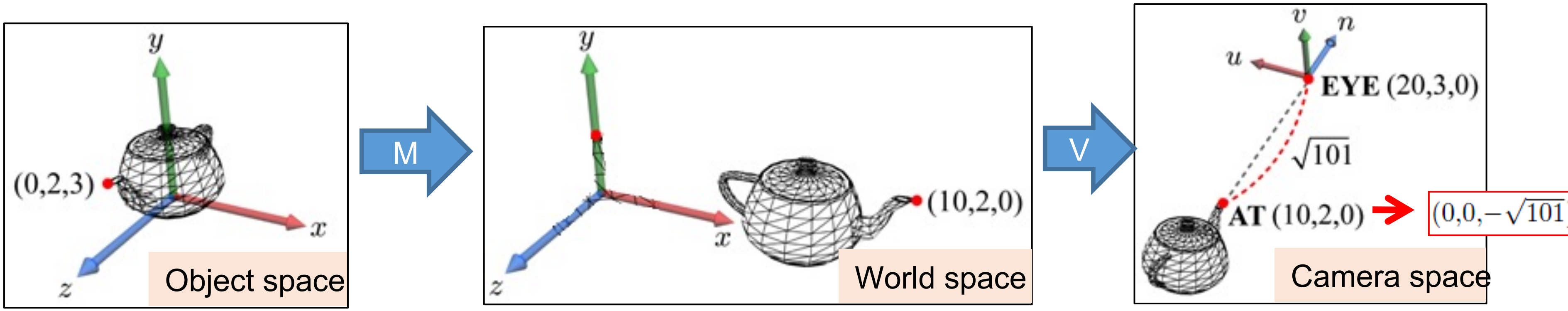
- OpenGL Utility Library (GLU)
  - Utilities e g setting camera view and projection
- GLUT (OpenGL Utility Toolkit)
  - A□auxiliary library
    - A portable windowingA□□
    - Easier to show the output of your OpeⱯGL application
    - Not officially part of OpeⱯGL
  - Handles
    - Window creation
    - OS system calls mouse buttons, movement, keyboard, etc...

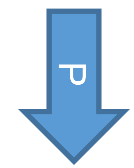
[illegible] A [illegible] ([illegible])

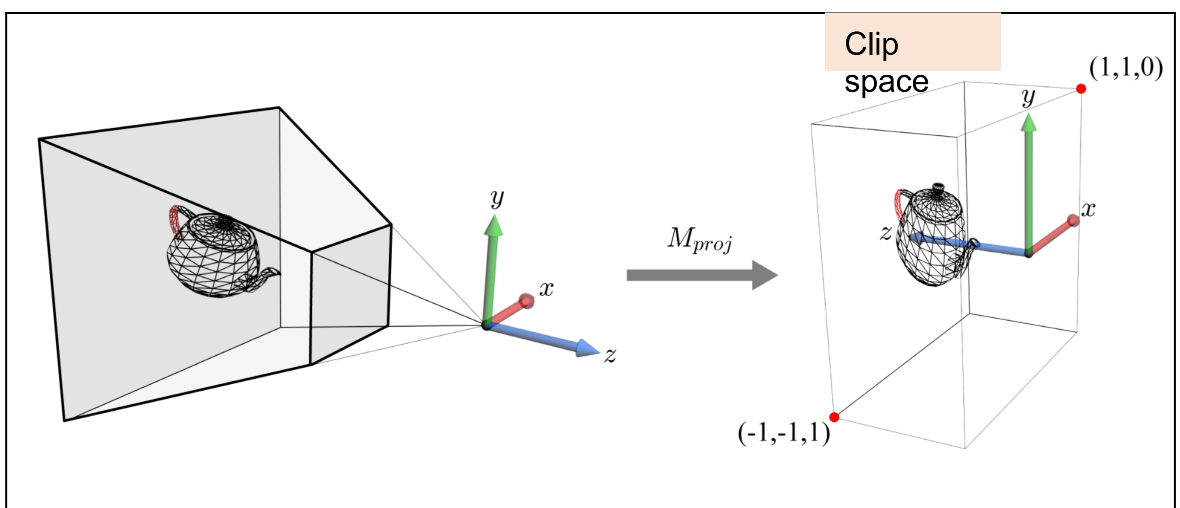
- OpenGL Extension Wrangler Library (GLEW)
  - D[illegible] G[illegible]
  - C[illegible]-[illegible] [illegible]-[illegible] C/C++ [illegible]
- Graphics Library Framework (GLFW)
  - [illegible], [illegible] G[illegible], [illegible], [illegible].

OpenGL and Related APIs



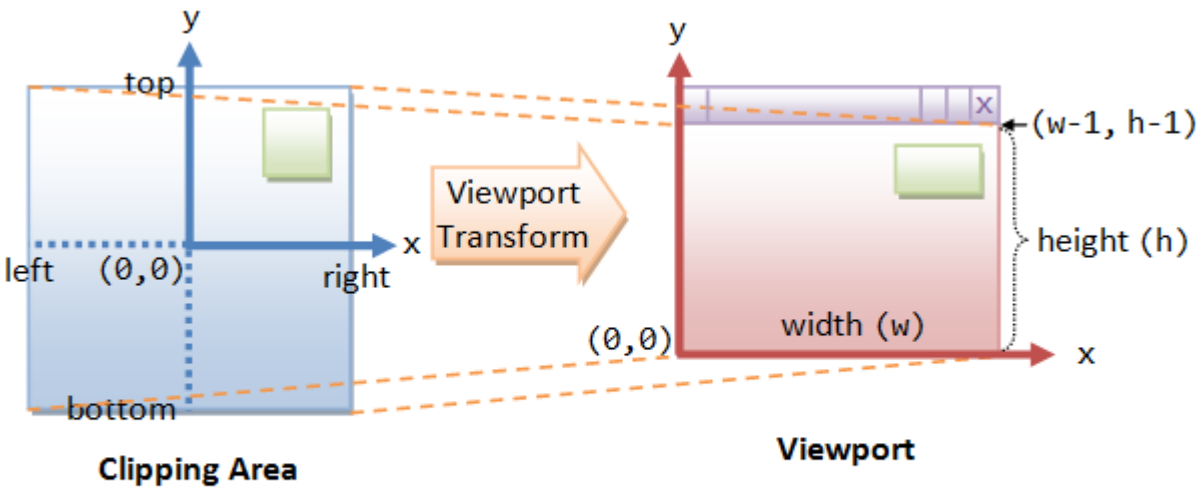






Model, View and Projection (MVP)

```
glMatrixMode(GL_MODELVIEW);
…
glMatrixMode(GL_PROJECTION);
```



Clipping Area and Viewport: Objects will be distorted if the aspect ratios of the clipping area and viewport are different.

[illegible]

- Download GLUT binaries for windows from “Nate Robins” ‘s website http://www.xmission.com/~nate/glut.html
- Put the files at appropriate folders

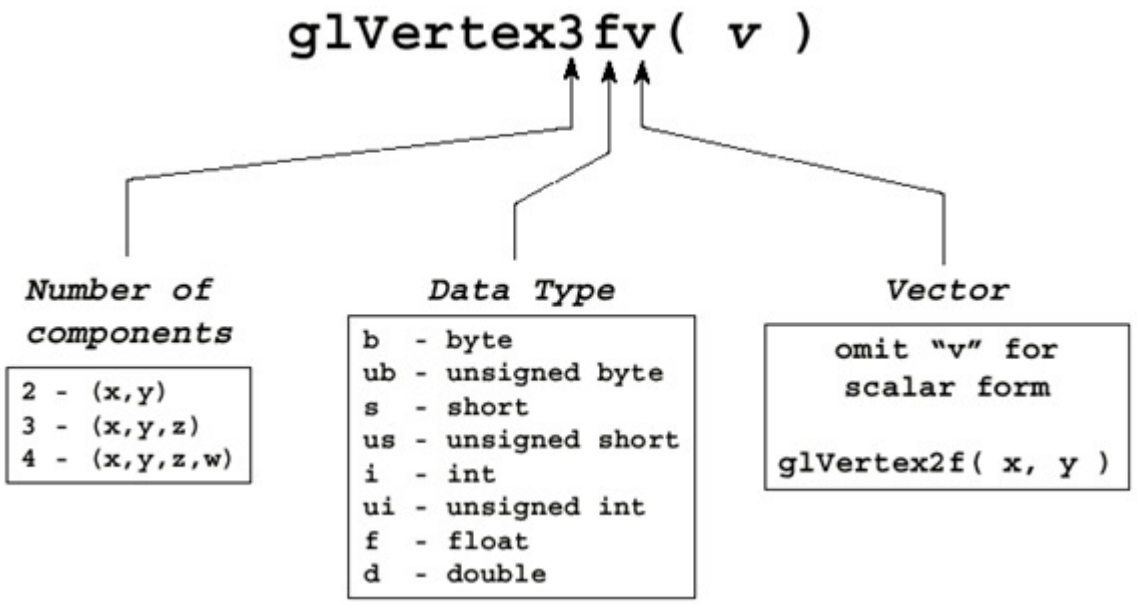
File	Location
glut32.dll	C:\WINDOWS\system\ (or system32)
glut32.lib	C:\Program Files\Microsoft Visual Studio 2005\VC\PlatformSDK\Lib
glut.h	C:\Program Files\Microsoft Visual Studio 2005\VC\PlatformSDK\Include\gl

- Make sure Visual Studio c++ projects links in the GLUT/gl/glu libraries. Go to Menu: “Project -> (your-project-name) Properties”

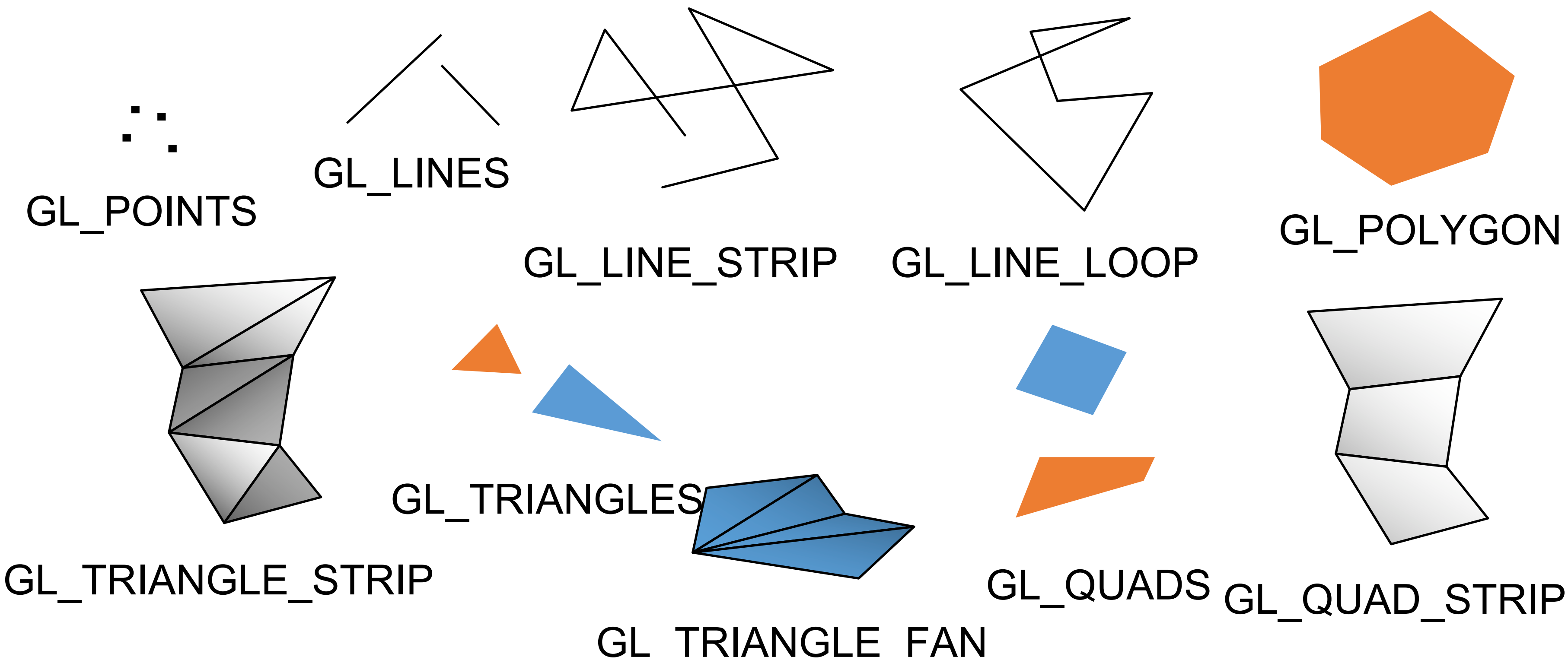
  Tab: “Configuration Properties -> Linker -> Input”
  Under “Additional Dependencies”, add “glut32.lib opengl32.lib glu32.lib”

- Under Configuration Properties->C++->General->Additional Include Directories : add "C:\Program Files\Microsoft Visual Studio 8\VC\PlatformSDK\Include"

- Begins with lower case: gl, glu, glut, glew, glfw
- Fllowed by the purpose of the function, in camel case (initial-capitalized) e.g., glColor
- Followed by specifications for the parameters, e.g., glColor3f
- OpenGL Command Formats



[illegible]



- All geometric primitives are specified by vertices
- Per-vertex data:
  - [illegible]

B[illegible]

```
#include <windows.h>  // for MS Windows
#include <GL/glut.h>  // GLUT, include glu.h and gl.h
void display() {/* Handler for window-repaint event. Called back when the window first appears and whenever the window needs to be re-painted. */
              glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set background color to black and opaque
              glClear(GL_COLOR_BUFFER_BIT);         // Clear the color buffer (background)
              glBegin(GL_TRIANGLES);
              glVertex3f(-0.5f, -0.4f, -1.0f);
              glVertex3f(0.5f, -0.4f, -1.0f);
              glVertex3f(0.0f, 1.0f, -1.0f);
              glEnd();
              glFlush();  // Render now
}
int main(int argc, char** argv) {
              glutInit(&argc, argv);          // Initialize GLUT
              glutInitWindowSize(640, 480);   // Set the window's initial width & height - non-square
              glutInitWindowPosition(50, 50); // Position the window's initial top-left corner
              glutCreateWindow("Model Transform");  // Create window with the given title
              glutDisplayFunc(display);       // Register callback handler for window re-paint event
              glutMainLoop();                 // Enter the infinite event-processing loop
              return 0;
```

Matrix Operations

- Specify Current Matrix Stack
  - **glMatrixMode(GL_MODELVIEW or GL_PROJECTION)**
- Other Matrix or Stack Operation
  - **glLoadIdentity() glPushMatrix() glPopMatrix()**
- Viewport
  - usually same as window size
  - viewport aspect ratio should be same as projection transformation or resulting image may be distorted
  - **glViewport(x, y, width, height)**

Applying Projection

- Transformations
- Typical use (orthographic projection)

```
glMatrixMode( GL_PROJECTION );
glLoadIdentity();
glOrtho( left, right, bottom, top, zNear, zFar );
```

- 2 projection methods:

```
void gluOrtho (
    GLdouble left,
    GLdouble right,
    GLdouble bottom,
    GLdouble top,
     GLdouble zNear,
     GLdouble zFar,
);
```

```
void gluPerspective (
    GLdouble fovy,
    GLdouble aspect,
    GLdouble zNear,
    GLdouble zFar
);
```

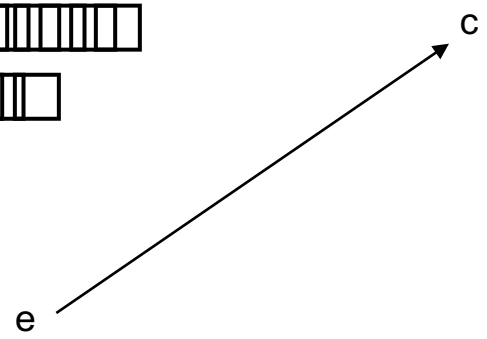
Viewing Transformations

- Position the camea/eye in the scene
- To “fly through” a scene
- change viewing transformation and redraw scene

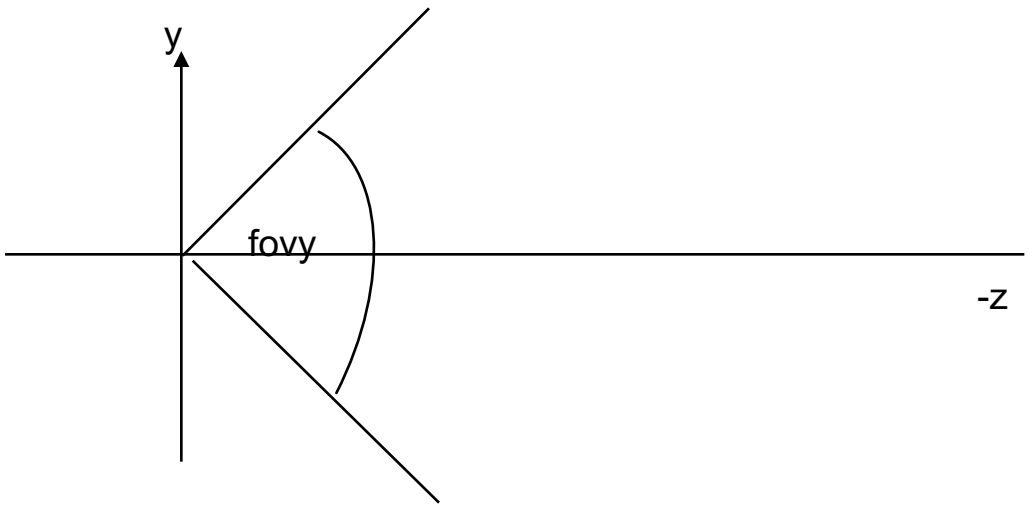
```
gluLookAt(eye x ,eye y ,eye z ,
          aim x ,aim y ,aim z ,
          up x ,up y ,up z)
```

- up vector determines unique orientation
- careful of degenerate positions

- Constructing an ‘M’ matrix
  - A(
    ,
    ,
    ,
    ,
    ,
    ,
    ,
    ,
)
- Matrix that maps
  - (
    ,
    ,
)
  - (
    ,
    ,
)
  - (
    ,
    ,
)
- Premultiplies current matrix



- To specify projection matrix:
  - [illegible]



- OpenGL uses a RH coordinate system throughout (hence the default VPN is the negative z-axis).
- It adopts the convention of points as column vectors and post-multiplication:
- The transpose of all our matrices should be used!

$$\begin{pmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 0 & c \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Modeling Transformations

- Move object
  - **glTranslate{fd}(x, y, z)**
- Rotate object around arbitrary axis
  - **glRotate{fd}(angle, x, y, z)**
  - angle is in degrees
- Dilate (stretch or shrink) object
  - **glScale{fd}(x, y, z)**

Common Transformation Usage

- Example of **reshape()** routine
  - restate projection & viewing transformations
- Usually called when window resized
- Registered a callback for **glutReshapeFunc()**

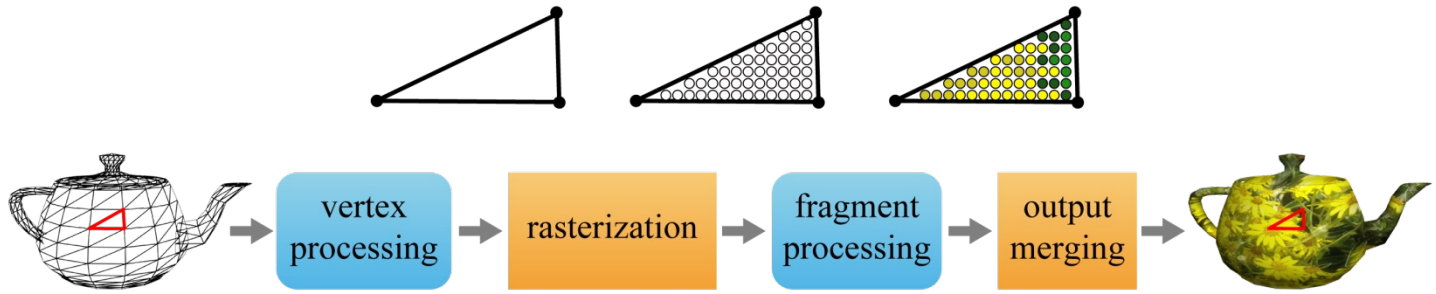
Reshape(): □□□□□□□□□□□& □□□□A□

```
void changeSize(int w, int h) {
   if(h == 0)h = 1; // Prevent a divide by zero, when window is
   too short
   float ratio = 1.0* w / h;

   // Reset the coordinate system before modifying
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   // Set the viewport to be the entire window
       glViewport(0, 0, w, h);
   // Set the correct perspective.
   gluPerspective(60, ratio,0.1, 100);//fovy, ratio, near, far

   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   gluLookAt (Eye.x, Eye.y, Eye.z, 0.0, 0.0, 0.0, 0.0, 1.0,
   0.0);//eye, at, up
}
```

Shaders



- OpenGL supports 02 shaders
  - [illegible]
  - F[illegible]
- Language: GL Shader Language (GLSL)
  - C[illegible]
  - B[illegible]:
    - G[illegible]

```
GLuint programID = LoadShaders(
            "SimpleVertexShader.vertexshader",
            "SimpleFragmentShader.fragmentshader" );

glUseProgram(programID);

glVertexAttribPointer(
0,                  // attribute 0. No particular reason for
0, but must match the layout in the shader.
3,                  // size
GL_FLOAT,           // type
GL_FALSE,           // normalized?
0,                  // stride
(void*)0            // array buffer offset
);
```

C++

```
// Input vertex data, different for all executions of this shader.

layout(location = 0) in vec3 vertexPosition_modelspace;

void main(){

   gl_Position.xyz = vertexPosition_modelspace;

   gl_Position.w = 1.0;

}
```

GLSL

- layout(location = 0)" refers to the buffer we use to feed the *vertexPosition_modelspace* attribute

F□□□□□□□□□□□□□□□□E□□□□□

```
#version 330 core
// Ouput data
out vec3 color;
void main()
{
        // Output color = red
        color = vec3(1,0,0);
}
```