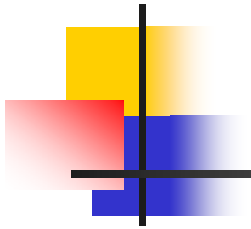
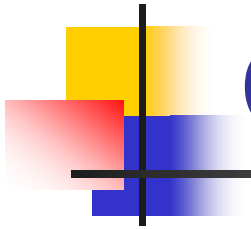


Programming with OpenGL Part 1: Background

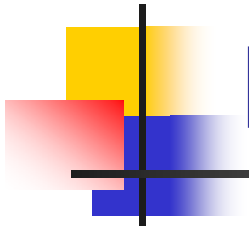
Objectives

- What is OpenGL
- Development of the OpenGL API
- OpenGL Architecture
  - OpenGL as a state machine
- Functions
  - Types
  - Formats
- Simple Programs



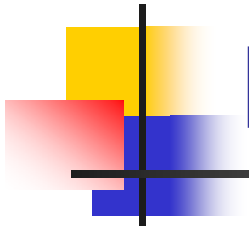
OpenGL

- Open Graphics Language
- A software interface to graphics hardware
- Consists of about 250 distinct commands
- Produce interactive 3D applications
- Streamlined and hardware-independent



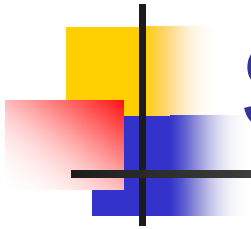
Early History of APIs

- A standard graphics API (1973)
  - IFIPS (International Federation of Information Processing Societies)
  - Graphical Kernel System (GKS)
    - 2D but contained good workstation model
  - Core
    - Both 2D and 3D
  - GKS adopted as IS0 and later ANSI standard (1980s)
- GKS not easily extended to 3D (GKS-3D)
  - Far behind hardware development



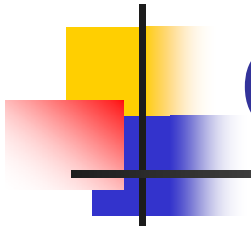
PHIGS and X

- Programmers Hierarchical Graphics System (PHIGS)
  - Arose from CAD community
  - Database model with retained graphics (structures)
- X Window System
  - DEC/MIT effort
  - Client-server architecture with graphics
- PEX combined the two
  - Not easy to use (all the defects of each)



SGI and GL

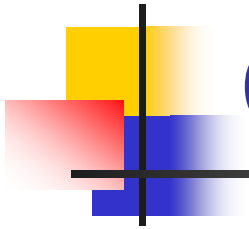
- Silicon Graphics (SGI) revolutionized the graphics workstation by implementing the pipeline in hardware (1982)
- To access the system, application programmers used a library called GL
- With GL, it was relatively simple to program three dimensional interactive applications



OpenGL

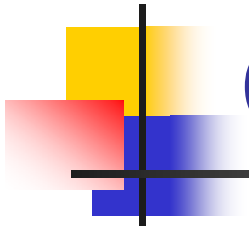
The success of GL leads to OpenGL (1992), a platform-independent API that was

- Easy to use
- Close enough to the hardware to get excellent performance
- Focus on rendering
- Omitted windowing and input to avoid window system dependencies



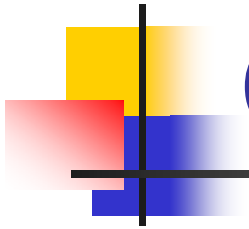
OpenGL Evolution

- Controlled by an Architectural Review Board (ARB)
  - Members include SGI, Microsoft, Nvidia, HP, 3DLabs, IBM,…….
  - Relatively stable (present version 2.1, Aug 2006)
    - Evolution reflects new hardware capabilities
      - **3D texture mapping and texture objects**
      - **Vertex and fragment programs**
  - Allows for platform specific features through extensions (ARB, NV, …)



OpenGL Libraries

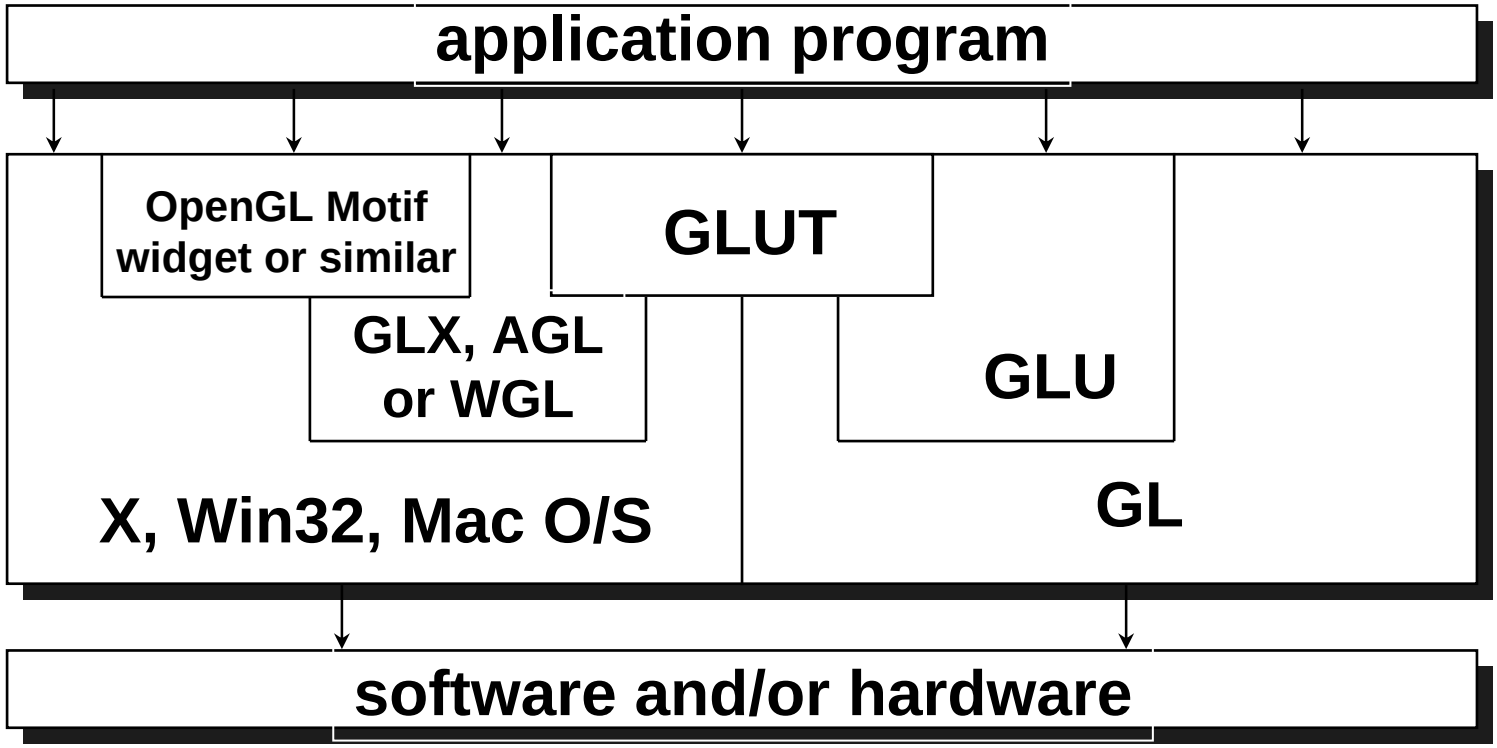
- OpenGL Core Library
  - OpenGL32 on Windows
  - GL on most unix/linux systems (libGL.a)
- OpenGL Utility Library (GLU)
  - Uses functions from OpenGL core to create more complex objects
- Links with window system
  - GLX for X window systems
  - WGL for Windows
  - AGL for Macintosh



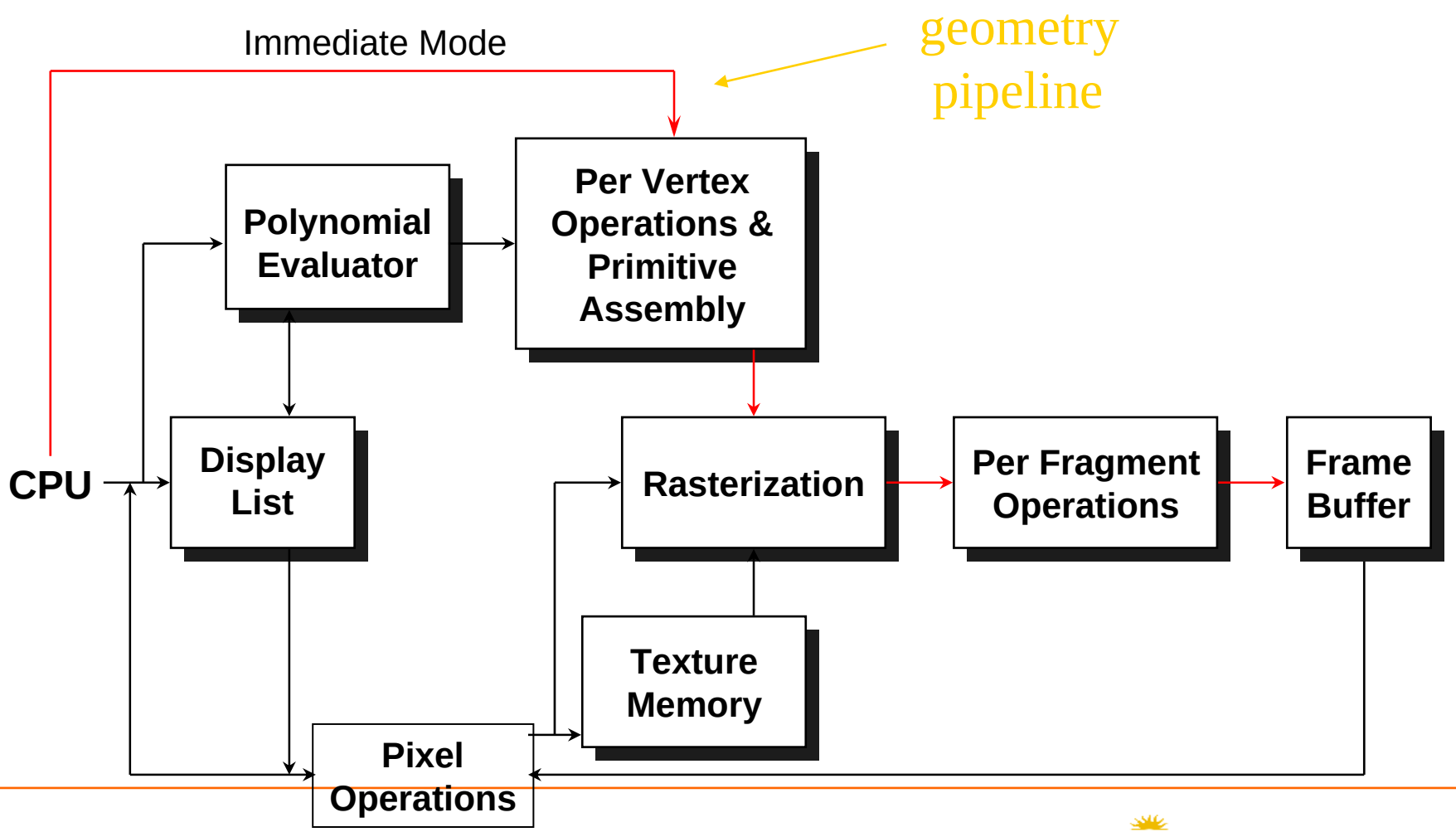
GLUT

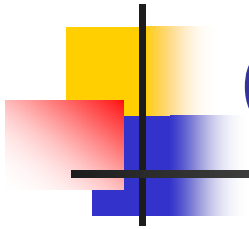
- OpenGL Utility Toolkit (GLUT)
  - Provides functionality common to all window systems
    - Open a window
    - Get input from mouse and keyboard
    - Menus
    - Event-driven
  - Code is portable but GLUT lacks the functionality of a good toolkit for a specific platform
    - No slide bars

Software Organization



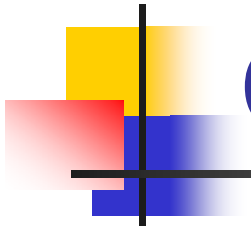
OpenGL Architecture





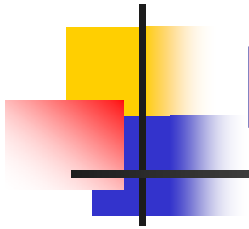
OpenGL Functions

- Primitives
  - Points
  - Line Segments
  - Polygons
- Attributes – colors, patterns, typefaces
- Transformations
  - Viewing
  - Modeling
- Control (GLUT)
- Input (GLUT)
- Query



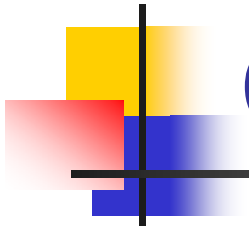
OpenGL State

- OpenGL is a state machine
- OpenGL functions are of two types
  - Primitive generating
    - Can cause output if primitive is visible
    - How vertices are processed and appearance of primitive are controlled by the state
  - State changing
    - Transformation functions
    - Attribute functions



Lack of Object Orientation

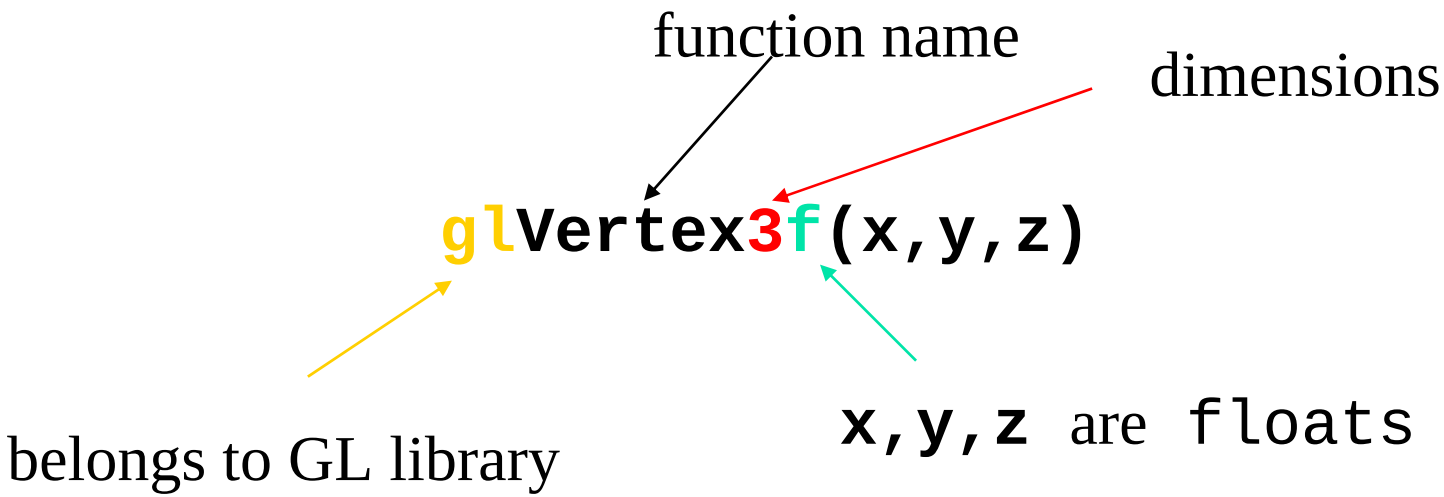
- OpenGL is not object oriented so that there are multiple functions for a given logical function
  - **`glVertex3f`**
  - **`glVertex2i`**
  - **`glVertex3dv`**
- Underlying storage mode is the same
- Easy to create overloaded functions in C++ but issue is efficiency



OpenGL #defines

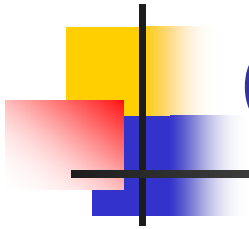
- Most constants are defined in the include files **`gl.h`**, **`glu.h`** and **`glut.h`**
  - Note **`#include <glut.h>`** should automatically include the others
  - Examples
  - **`glBegin(GL_POLYGON)`**
  - **`glClear(GL_COLOR_BUFFER_BIT)`**
- include files also define OpenGL data types: **`GLfloat`**, **`GLdouble`**,….

OpenGL function format



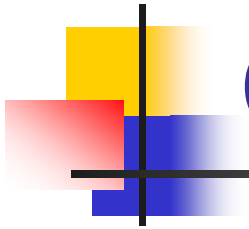

```
glVertex3fv(p)
```

p is a pointer to an array



OpenGL: Conventions

- Function names indicate argument type and number
  - Functions ending with **`f`** take floats
  - Functions ending with **`i`** take ints
  - Functions ending with **`b`** take bytes
  - Functions ending with **`ub`** take unsigned bytes
  - Functions that end with **`v`** take an array.
- Examples
  - **`glColor3f()`** takes 3 floats
  - **`glColor4fv()`** takes an array of 4 floats

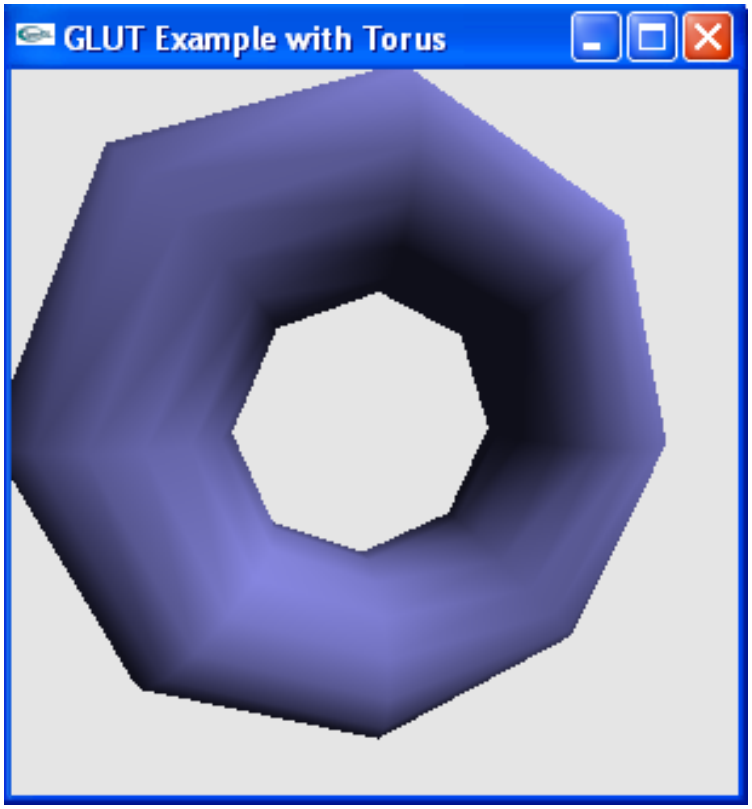


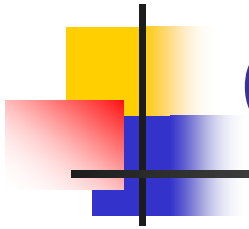
OpenGL: Conventions

- Variables written in CAPITAL letters
  - Example: GLUT_SINGLE, GLUT_RGB
  - usually constants
  - use the bitwise or command (x | y) to combine constants

A Simple Program

Generate a torus in a window





OpenGL Display

```
void myDisplay( void )
{
    static float rotationX = 0.0, rotationY = 0.0;
    glClearColor( .9f, .9f, .9f, 1.0f );
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
     /*** Rotate the object ***/
    rotationX += 3.3f;
    rotationY += 4.7f;
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    glTranslatef( 0.0, 0.0, -1.0 );
    glRotatef( rotationY, 0.0, 1.0, 0.0 );
    glRotatef( rotationX, 1.0, 0.0, 0.0 );
    glutSolidTorus( .2,.5,16,segments );
    glutSwapBuffers();
}
```