



The University of New Mexico

---

# Programming with OpenGL

## Part 1: Background

Ed Angel

Professor of Computer Science,  
Electrical and Computer  
Engineering, and Media Arts  
University of New Mexico



# Objectives

---

- Development of the OpenGL API
- OpenGL Architecture
  - OpenGL as a state machine
  - You might need to change the state
- Functions
  - Types
  - Formats
- Simple program



# Early History of APIs

---

- IFIPS (1973) formed two committees to come up with a standard graphics API
  - Graphical Kernel System (GKS)
    - 2D but contained good workstation model
  - Core
    - Both 2D and 3D
  - GKS adopted as ISO 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
- First Animation Movie Oscar



# 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
- C/C++ version it is not Object Oriented
- Focus on rendering
- Omitted windowing and input to avoid window system dependencies



# OpenGL Evolution

---

- Originally controlled by an Architectural Review Board (ARB)
  - Members included SGI, Microsoft, Nvidia, HP, 3DLabs, IBM,.....
  - Relatively stable (present version 2.1; 4.x)
    - Evolution reflects new hardware capabilities
      - 3D texture mapping and texture objects
      - Vertex programs
  - Allows for platform specific features through extensions
  - ARB replaced by Kronos



# OpenGL Libraries

---

- OpenGL core library
  - OpenGL32 on Windows
  - GL on most unix/linux systems (libGL.a)
- OpenGL Utility Library (GLU)
  - Provides functionality in OpenGL core but avoids having to rewrite code
- Links with window system
  - GLX for X window systems
  - WGL for Windows
  - AGL for Macintosh



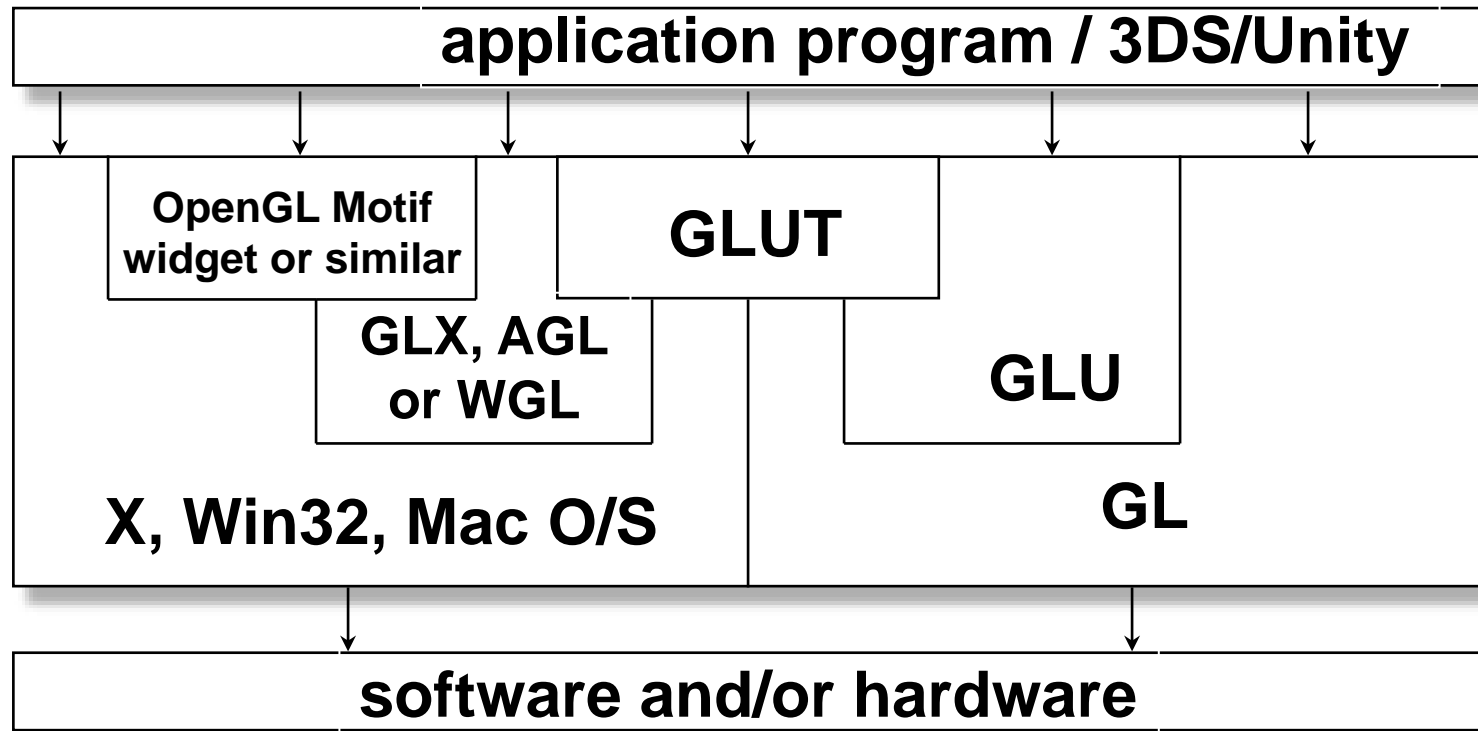


# GLUT

- 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

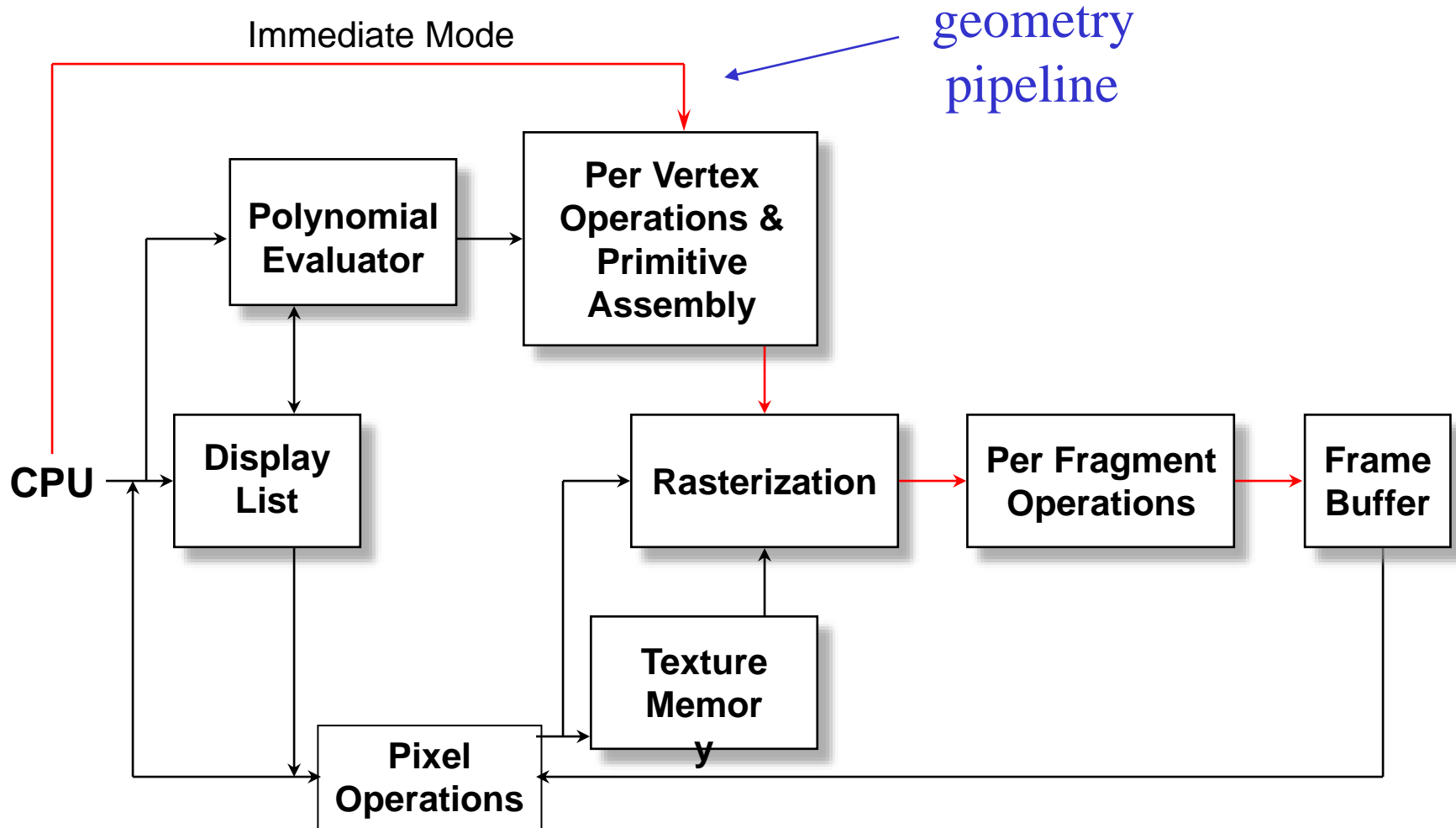


# Software Organization





# OpenGL Architecture





# OpenGL Functions

---

- Primitives
  - Points / Vertices
  - Line Segments
  - Polygons
- Attributes e.g., `glPointSize(4.0)` every rendered point is 4x4 pixels (state variable)
- Transformations
  - Viewing – Camera
  - Modeling - For the model
- 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 `glScalef();`
    - Attribute functions `glLinedepth(2.0)`



# 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



The University of New Mexico

# OpenGL function format

---

function name

dimensions

`glVertex3f(x, y, z)`

belongs to GL library

`x, y, z` are floats

`glVertex3fv(p)`

`p` is a pointer to an array



# OpenGL #defines

---

- Most constants are defined in the include files `gl.h`, `glu.h` and `glut.h`
  - Note `#include <GL/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`, ....
- IBM-Float format? Before IEEE754
- EBCDIC? Before ASCII



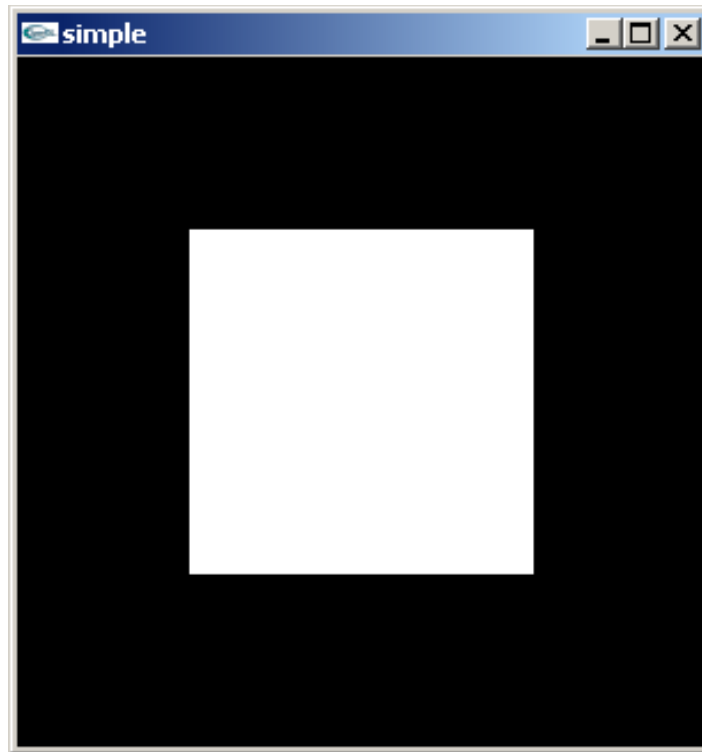


The University of New Mexico

# A Simple Program

---

Generate a square on a solid background





# simple.c

```
#include <GL/glut.h>
void mydisplay() {
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
        glVertex2f(-0.5, -0.5);
        glVertex2f(-0.5, 0.5);
        glVertex2f(0.5, 0.5);
        glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}
int main(int argc, char** argv) {
    glutCreateWindow("simple");
    glutDisplayFunc(mydisplay);
    glutMainLoop();
}
```



# Event Loop

- Note that the program defines a *display callback* function named **mydisplay**
  - Every glut program must have a display callback
  - The display callback is executed whenever OpenGL decides the display must be refreshed, for example when the window is opened
  - The **main** function ends with the program entering an event loop



# Defaults

---

- **simple.c** is too simple
- Makes heavy use of state variable default values for
  - Viewing
  - Colors
  - Window parameters
- Next version will make the defaults more explicit



# Notes on compilation

---

- See website and ftp for examples
- Unix/linux
  - Include files usually in `.../include/GL`
  - Compile with `-lglut -lglu -lgl loader` flags
  - May have to add `-L` flag for X libraries
  - Mesa implementation included with most linux distributions
  - Check web for latest versions of Mesa and glut



# Compilation on Windows

---

- Visual C++
  - Get glut.h, glut32.lib and glut32.dll from web
  - Create a console application
  - Add opengl32.lib, glut32.lib, glut32.lib to project settings (under link tab)
- Borland C similar
- Cygwin (linux under Windows)
  - Can use gcc and similar makefile to linux
  - Use `-lopengl32 -lglu32 -lglut32` flags