Programming with OpenGL
Part 1: Background

Ed Angel

Professor Emeritus of Computer Science

University of New Mexico

Objectives

- Development of the OpenGL API
- OpenGL Architecture
  - OpenGL as a state machine
  - OpenGL as a data flow machine
- 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 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 lead to OpenGL (1992), a platform-independent API that was

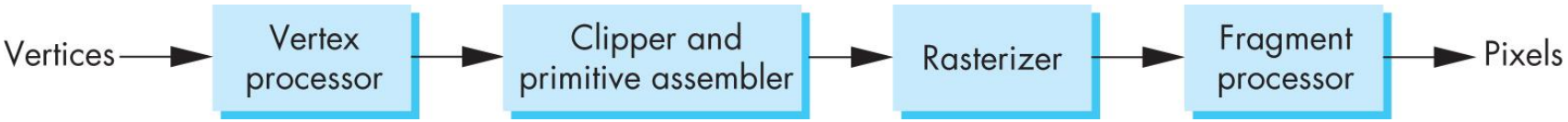
- Easy to use
- Close enough to the hardware to get excellent performance
- Focus on rendering
- Omitted windowing and input to avoid window system dependencies

OpenGL Evolution

- Originally controlled by an Architectural Review Board (ARB)
  - Members included SGI, Microsoft, Nvidia, HP, 3DLabs, IBM,…….
  - Now Kronos Group
  - Was relatively stable (through version 2.5)
    - Backward compatible
    - Evolution reflected new hardware capabilities
      - 3D texture mapping and texture objects
      - Vertex and fragment programs
  - Allows platform specific features through extensions

Modern OpenGL

- Performance is achieved by using GPU rather than CPU
- Control GPU through programs called shaders
- Application's job is to send data to GPU
- GPU does all rendering



OpenGL 3.1

- Totally shader-based
  - No default shaders
  - Each application must provide both a vertex and a fragment shader
- No immediate mode
- Few state variables
- Most 2.5 functions deprecated
- Backward compatibility not required

Other Versions

- OpenGL ES
  - Embedded systems
  - Version 1.0 simplified OpenGL 2.1
  - Version 2.0 simplified OpenGL 3.1
    - Shader based
- WebGL
  - Javascript implementation of ES 2.0
  - Supported on newer browsers
- OpenGL 4.1 and 4.2
  - Add geometry shaders and tessellator

What About Direct X?

- Windows only
- Advantages
  - Better control of resources
  - Access to high level functionality
- Disadvantages
  - New versions not backward compatible
  - Windows only
- Recent advances in shaders are leading to convergence with OpenGL

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
  - Will only work with legacy 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

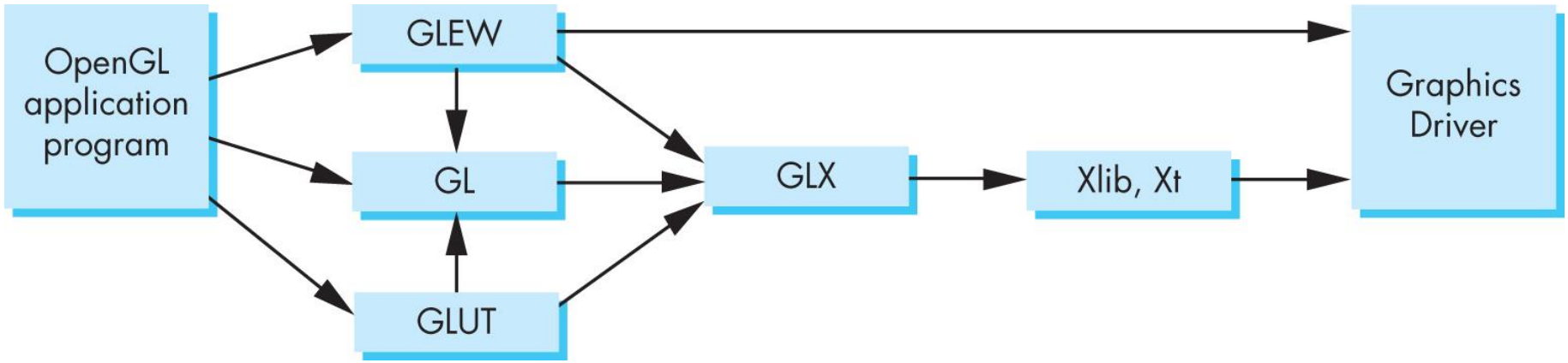
freeglut

- GLUT was created long ago and has been unchanged
  - Amazing that it works with OpenGL 3.1
  - Some functionality can't work since it requires deprecated functions
- freeglut updates GLUT
  - Added capabilities
  - Context checking

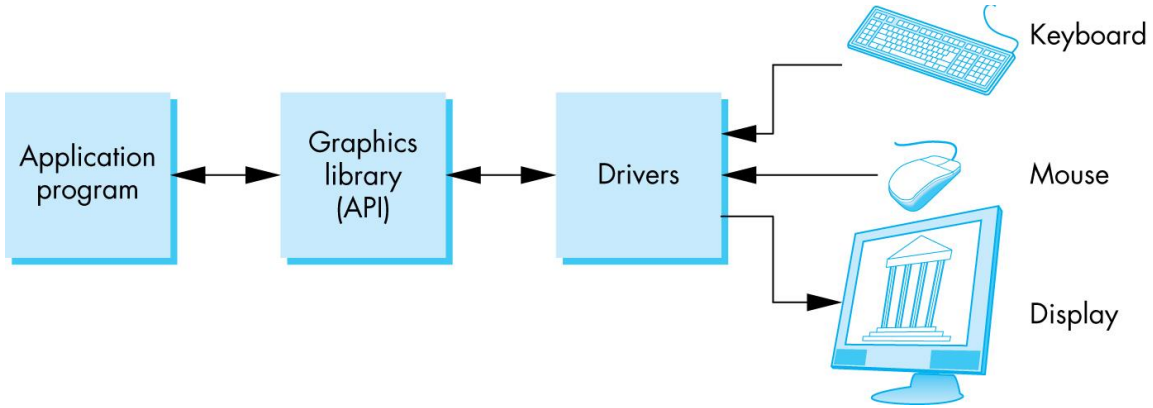
GLEW

- OpenGL Extension Wrangler Library
- Makes it easy to access OpenGL extensions available on a particular system
- Avoids having to have specific entry points in Windows code
- Application needs only to include glew.h and run a glewInit()

Software Organization



OpenGL Architecture



OpenGL Functions

- Primitives
  - Points
  - Line Segments
  - Triangles
- Attributes
- Transformations
  - Viewing
  - Modeling
- Control (GLUT)
- Input (GLUT)
- Query

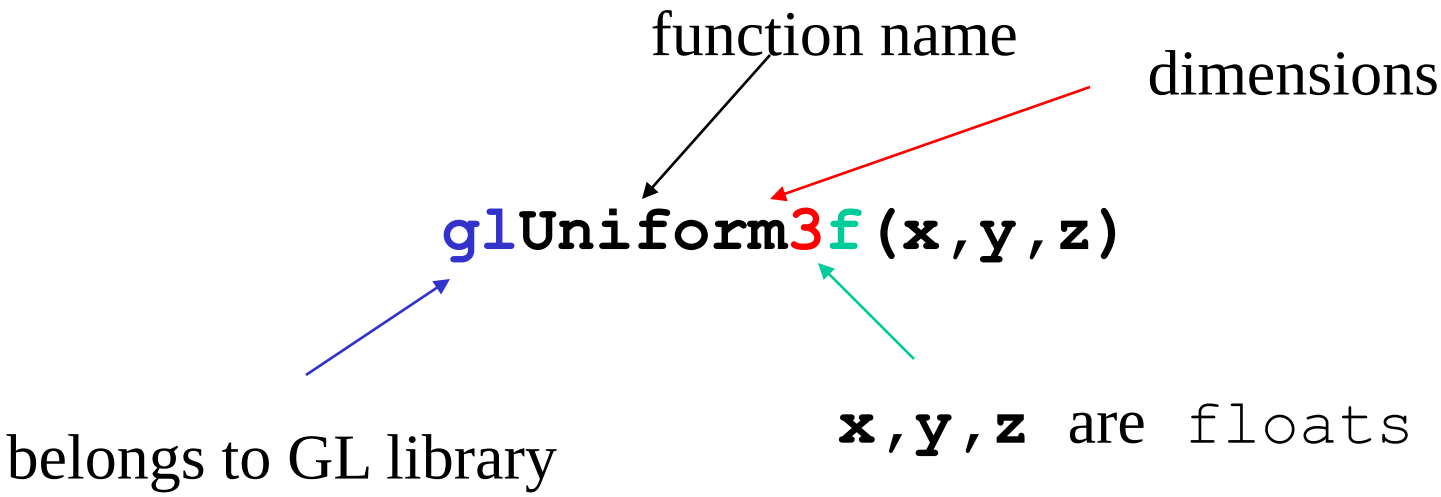
OpenGL State

- 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
    - Under 3.1 most state variables are defined by the application and sent to the shaders

Lack of Object Orientation

- OpenGL is not object oriented so that there are multiple functions for a given logical function
  - **`glUniform3f`**
  - **`glUniform2i`**
  - **`glUniform3dv`**
- Underlying storage mode is the same
- Easy to create overloaded functions in C++ but issue is efficiency

OpenGL function format




```
glUniform3fv(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
  - **`glEnable(GL_DEPTH_TEST)`**
  - **`glClear(GL_COLOR_BUFFER_BIT)`**
- include files also define OpenGL data types: **`GLfloat`**, **`GLdouble`**,….

OpenGL and GLSL

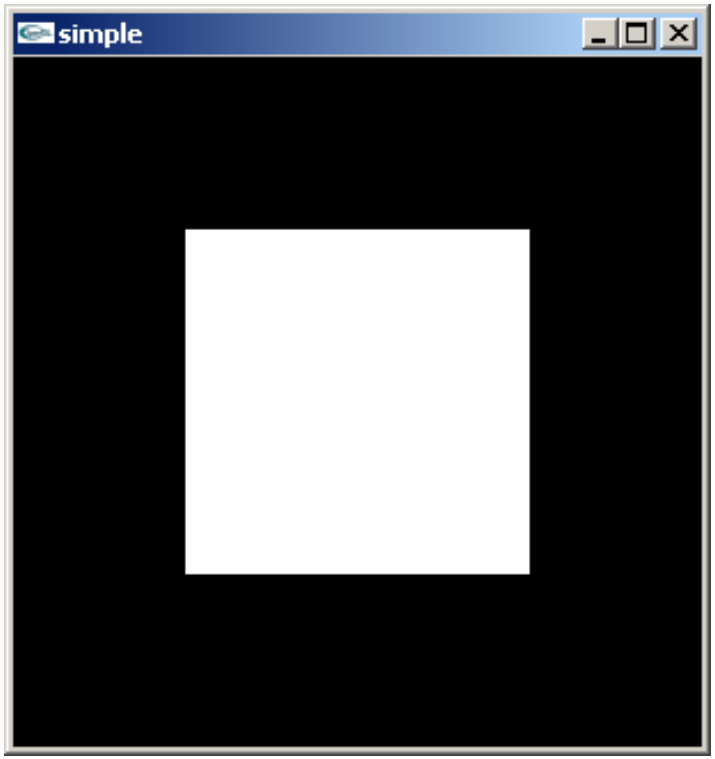
- Shader based OpenGL is based less on a state machine model than a data flow model
- Most state variables, attributes and related pre 3.1 OpenGL functions have been deprecated
- Action happens in shaders
- Job is application is to get data to GPU

GLSL

- OpenGL Shading Language
- C-like with
  - Matrix and vector types (2, 3, 4 dimensional)
  - Overloaded operators
  - C++ like constructors
- Similar to Nvidia's Cg and Microsoft HLSL
- Code sent to shaders as source code
- New OpenGL functions to compile, link and get information to shaders

A Simple Program (?)

Generate a square on a solid background



It used to be easy

```
#include <GL/glut.h>
void mydisplay(){
      glClear(GL_COLOR_BUFFER_BIT);
      glBegin(GL_QUAD;
            glVertex2f(-0.5, -0.5);
            glVertex2f(-0,5, 0,5);
            glVertex2f(0.5, 0.5);
            glVertex2f(0.5, -0.5);
      glEnd()
}
int main(int argc, char** argv){
      glutCreateWindow("simple");
      glutDisplayFunc(mydisplay);
      glutMainLoop();
}
```

What happened

- Most OpenGL functions deprecated
- Makes heavy use of state variable default values that no longer exist
  - Viewing
  - Colors
  - Window parameters
- Next version will make the defaults more explicit
- However, processing loop is the same

simple.c

```
#include <GL/glut.h>
void mydisplay(){
      glClear(GL_COLOR_BUFFER_BIT);

// need to fill in this part
// and add in shaders
}
int main(int argc, char** argv){
      glutCreateWindow("simple");
      glutDisplayFunc(mydisplay);
      glutMainLoop();
}
```

Event Loop

- Note that the program specifies 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

Notes on compilation

- See website and ftp for examples
- Unix/linux
  - Include files usually in …/include/GL
  - Compile with –lglut –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
  - Install in same places as corresponding OpenGL files
  - Create an empty application
  - Add glut32.lib to project settings (under link tab)
  - Same for freeglut and GLEW
- Cygwin (linux under Windows)
  - Can use gcc and similar makefile to linux
  - Use –lopengl32–lglut32 flags