Comp 410/510

Computer Graphics

Spring 2023

Programming with OpenGL
Part 1: OpenGL Overview

Objectives

- Development of the OpenGL API
- OpenGL Architecture
  - OpenGL as a state machine
  - Shaders
- Functions
  - Types
  - Formats
- A “simple” program

SGI and GL

- Silicon Graphics (SGI) revolutionized the graphics workstation by implementing the graphics 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

Graphics (rendering) pipeline:

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 operations to avoid window system dependencies

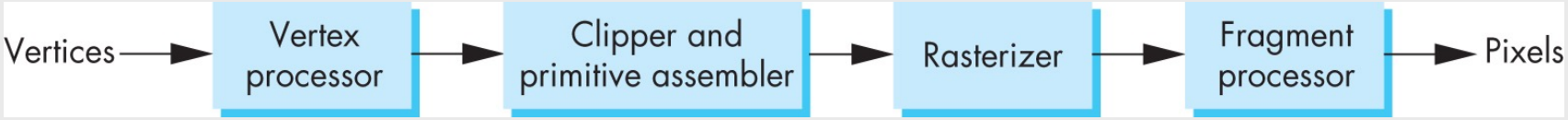
OpenGL Evolution

- Originally controlled by an Architectural Review Board (ARB)
  - Members included SGI, Microsoft, Nvidia, HP, 3DLabs, IBM,…
  - Now Khronos Group
- Was relatively stable (till version 3)
    - Backward compatible
    - Evolution reflects new hardware capabilities
      - 3D texture mapping and texture objects
      - Vertex and fragment programs (shaders)
- Allows for 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

Graphics pipeline:



Rendering

OpenGL 3.1

- Totally shader-based
  - No default shaders
  - Each application **must** provide both a vertex shader and a fragment shader
- Few built-in state variables
- Many pre OpenGL 3.0 state variables and functions deprecated or removed
- Backward compatibility not required (depends on the spesific implementation)
- Current version 4.6

Other Versions

- OpenGL ES
  - Used in embedded systems like smartphones, computer tablets, game consoles.
  - Version 1.0: simplified OpenGL 1.3
  - Version 2.0: simplified OpenGL 2.0
    - Shader based
  - Version 3.0
- WebGL
  - Javascript implementation of ES 2.0
  - Supported on most browsers

OpenGL Libraries

- OpenGL core library
- Links with spesific platforms and window systems
  - GLX for X window systems
  - WGL for Windows
  - CGL for Mac OS X

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 some functionality of a full-featured toolkit for a specific platform
    - No slide bars
- *freeglut* updates GLUT with added capabilities

GLEW

- OpenGL Extension Wrangler Library
- Provides efficient run-time mechanisms to determine and access OpenGL extensions supported on a particular platform
- OpenGL extensions are a means for OpenGL implementations to provide new or expanded functionality that the core of OpenGL does not provide
- Application needs to include `glew.h` and run `glewInit()`

GLFW

- Graphics Library Framework (GLFW)
  - Similar to GLUT but with better support
  - Lacks menus, buttons, etc.

- We will use GLFW

Software Organization

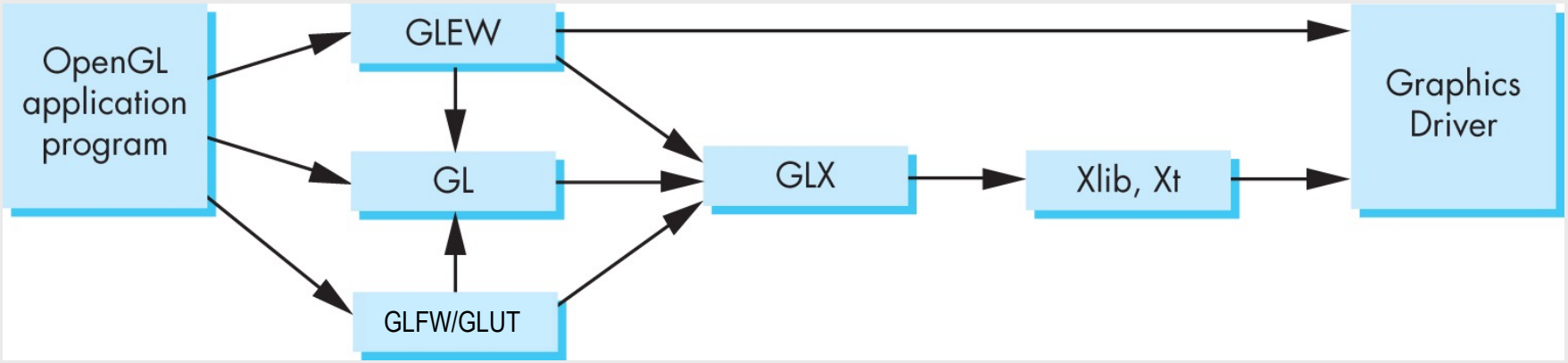


Figure assumes **X window system** employed in UNIX-like operating systems

OpenGL State

- OpenGL is a state machine
- OpenGL function types
  - Drawing
    - Causes graphics output if primitives are visible
    - How vertices are processed and appearance of primitives, are controlled by the state
    - e.g., glDrawArrays(GL_TRIANGLES, first_index, last_index)
  - State changing
    - Functions to set built-in state variables
    - e.g., glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)
    - By OpenGL 3.1, most state variables are defined by the application and sent to the shaders
  - Shader-related functions (to create, compile and link shaders)
  - Query functions (to query the state of the OpenGL context)
  - etc.

Lack of Object Orientation

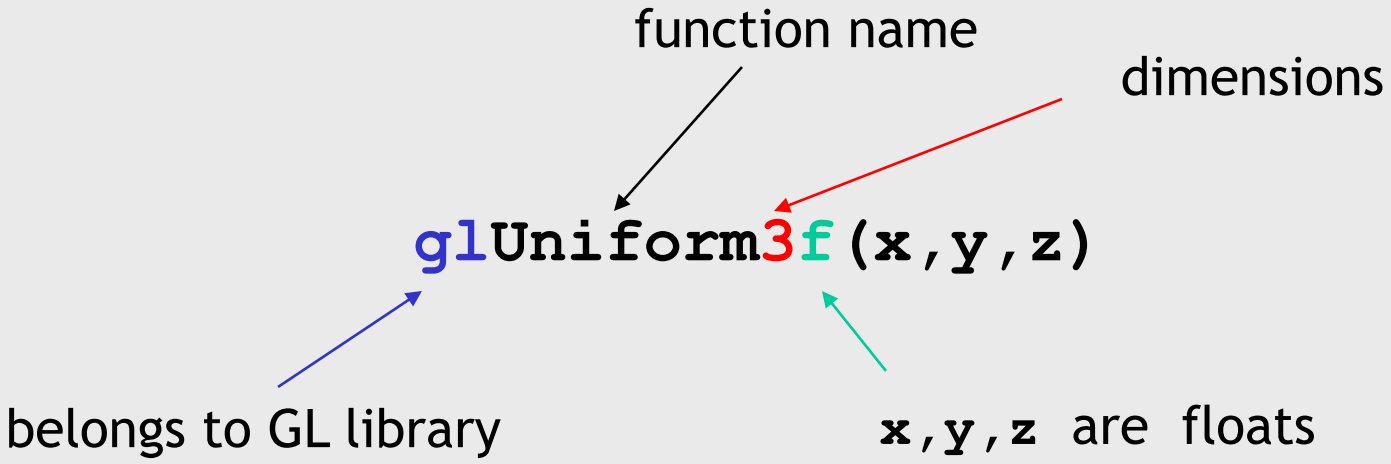
- OpenGL is **not object oriented** so that there are multiple functions for a given logical function

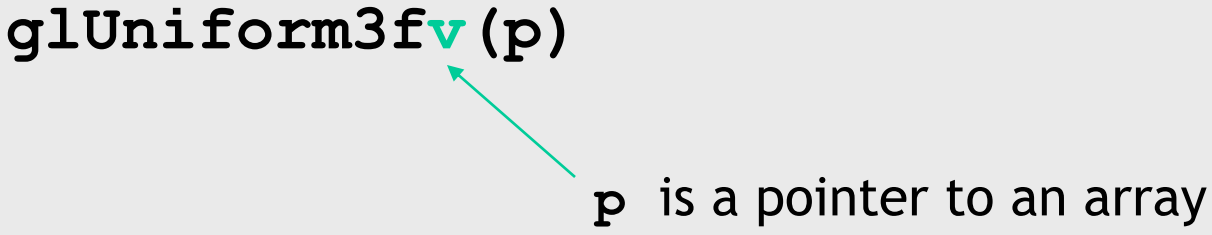
```
glUniform3f
glUniform2i
glUniform3dv
```

- Easy to create overloaded functions in C++ but issue is efficiency

`glUniform` is used to pass the value of a uniform variable into a shader.

OpenGL function format





OpenGL #defines

- Most symbolic constants (as part of OpenGL state) are defined in the include files such as **`gl3.h`** and **`glfw3.h`**
  - Examples
    - **`glEnable(GL_DEPTH_TEST)`**
    - **`glClear(GL_COLOR_BUFFER_BIT)`**

- Included headers also define OpenGL data types: **`Glfloat`**, **`Gldouble`**,…

Shader-based OpenGL

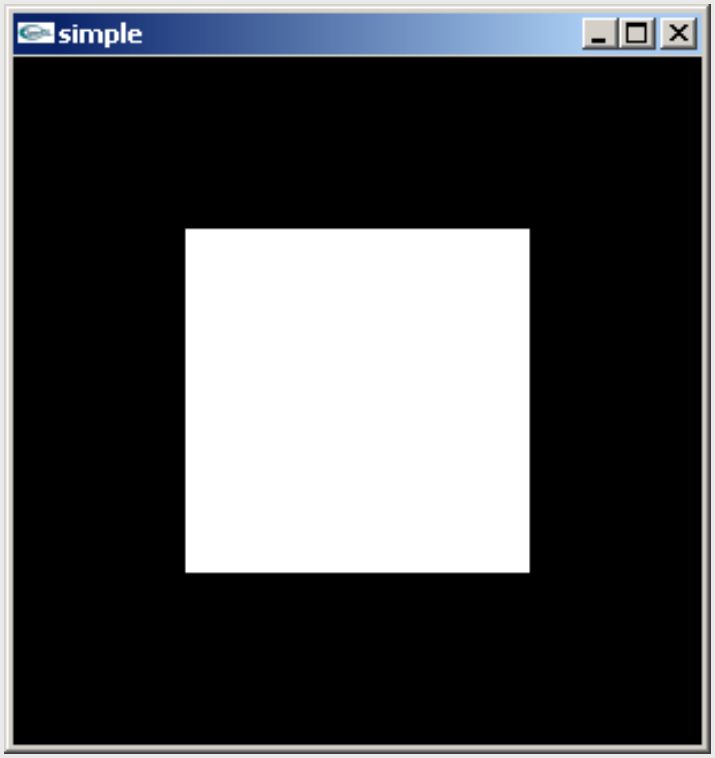
- Shader-based OpenGL is based less on a **state machine** model than a **data flow** model
- Most built-in state variables, attributes and related pre 3.1 OpenGL functions have been deprecated or removed
- State variables are mostly defined by the application and sent to the shaders
- Action happens in shaders
- Application’s job is to get data to GPU

OpenGL and GLSL

- GLSL: OpenGL Shading Language
- Shaders are coded in GLSL
- 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 is sent to GPU as source code
- OpenGL functions to compile, link and get information to shaders

A Simple Program

Generate a square on a solid background



It used to be simple

```
#include <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();
}
```

All functionalities in red have been removed by OpenGL 3.1.

What happened

- Most OpenGL functions deprecated
- The previous code example makes heavy use of state variable default values most of which no longer exist
  - Viewing
  - Colors
  - Window parameters
- Next version (next lecture) will make the defaults more explicit
- However, the processing loop is the same

simple.c

```
#include <gl3.h>
#include <glfw3.h>

void init(){
 // need to add shaders
 // need to define state variables
 // etc
}

void display(){
      glClear(GL_COLOR_BUFFER_BIT);
      // need to fill in this part
}

int main(int argc, char** argv){
   //need to add here some GLFW initialization
   GLFWwindow* window = glfwCreateWindow(500, 500, "Simple", NULL, NULL);
   glfwMakeContextCurrent(window);
   init();
   while (!glfwWindowShouldClose(window)){
        display();
        glfwSwapBuffers(window);
        glfwPollEvents();
   }
}
```

Event loop

simple.c

```
#include <gl3.h>
#include <glfw3.h>

void init(){
 // need to add shaders
 // need to define state variables
 // etc
}

void display(){
      glClear(GL_COLOR_BUFFER_BIT);
      // need to fill in this part
}

int main(int argc, char** argv){
   //need to add here some GLFW initialization
   GLFWwindow* window = glfwCreateWindow(500, 500, "Simple", NULL, NULL);
   glfwMakeContextCurrent(window);
   init();
   while (!glfwWindowShouldClose(window)){
        display();
        glfwSwapBuffers(window);
        glfwPollEvents();
   }
}
```

- The **display** function is executed repeatedly in the main loop, during which an event may happen:
- The state variables may be updated, or some user input may be provided or a system related event may happen
- Hence the need for refreshing the frame buffer

Event loop