OpenGL I: Intro
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 the 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
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()
OpenGL Architecture
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

\textbf{function name} \hspace{1cm} \textbf{dimensions}

\texttt{glUniform3f}(x, y, z)

belongs to GL library

\(x, y, z\) are \texttt{floats}

\texttt{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 of 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