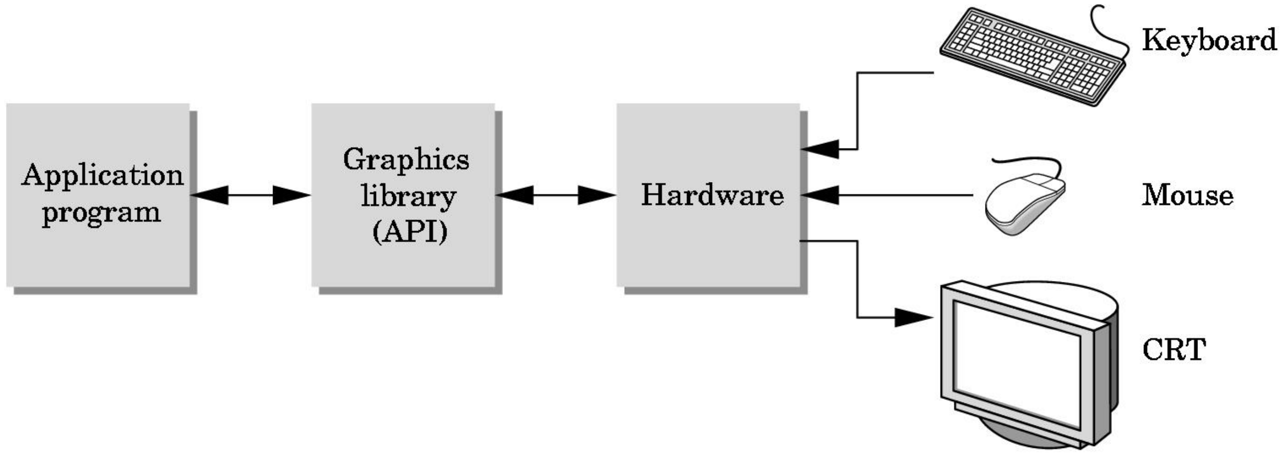
CS 432 – Interactive Computer Graphics

Lecture 1 – Part 2

Graphics APIs

The Programmer’s Interface

- Programmer sees the graphics system through a software interface
  - The Application Programmer Interface (API)



API Contents

- In order to form an image we're provide functions that allow us to specify:
  - Objects
  - Viewer (camera)
  - Light Source(s)
  - Materials
- They also handle other things:
  - Input from devices such as keyboard and mouse
  - Capabilities of system.

Object Specification

- Most APIs support a limited set of primitives including
  - Points (0D object)
  - Line segments (1D objects)
  - Polygons (2D objects)
  - Some curves and surfaces
    - Quadratics
    - Parametric polynomials
- All are defined through locations in space, or *vertices*

Example

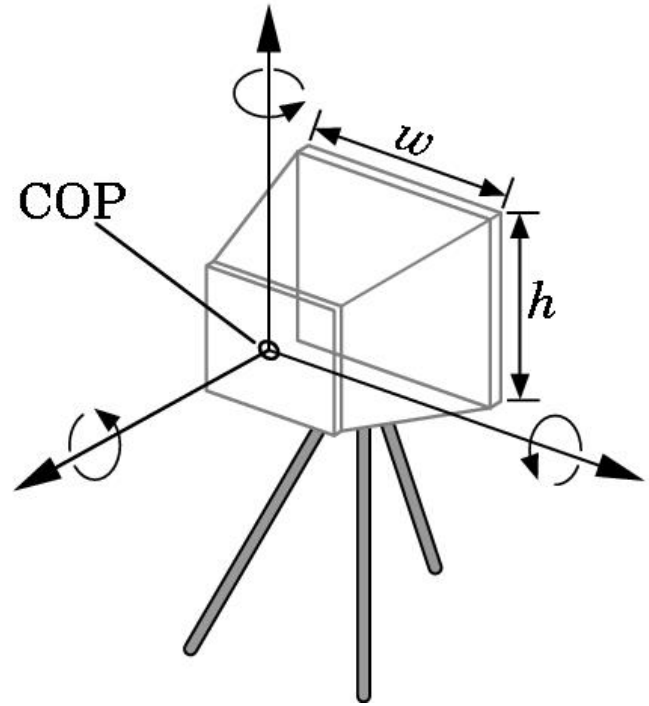
- Put geometric data in an array

```
vec3 points[3];
points[0] = vec3(0.0, 0.0, 0.0);
points[1] = vec3(0.0, 1.0, 0.0);
points[2] = vec3(1.0, 1.0, 0.0);
```

- Send array to GPU
- Tell GPU to render as triangle

Camera Specifications

- Six degrees of freedom
  - Position of center of lens
  - Orientation
- Lens
- Film size



Lights and Materials

- Types of lights
  - Point sources
  - Spot lights
  - Near and far sources
  - Color properties
- Material properties
  - Absorption: color properties
  - Scattering
    - Diffuse
    - Specular

Graphics Functions

- We could group the API functions into 7 major groups
    1. Primitive Functions
    2. Attribute Functions
    3. Viewing Functions
    4. Transformation Functions
    5. Input Functions
    6. Control Functions
    7. Query Functions

Graphics Functions

- Primitive Functions
  - Allow for specification of points, line segments, polygons, pixels, text, curves, etc..
  - OpenGL only supports points, line segments, and triangles
    - Some depreciated functions for quads and polygons
- Attribute Functions
  - Things like choosing color for line segment, pattern to fill polygon with, typefaces
  - In OpenGL we set colors by passing the info from the application to the shader or have the shader compute the color

Graphics Functions

- Viewing Functions
  - Allows us to specify views
  - OpenGL doesn’t provide any viewing functions
    - Relies on transformation in the shaders to do this
- Transformation Functions
  - Allow for operations like rotation, translation, and scaling
- Input Functions
  - For interactive programs, these let us deal with devices like mice, keyboards, tablets

Graphics Functions

- Control Functions
  - Allows us to communicate with the window system, initialize our program, and deal with errors
- Query Functions
  - Here we can ask information about the system
  - Can also ask about camera parameters, values in the frame buffer, etc..

OpenGL

- OpenGL is one API for Modern Graphics
- Why OpenGL?
  - Runs on any system
- May need GLUT
  - Provided on Macs
  - freeglut on web for Windows
- May need GLEW
  - Not needed on Macs
  - From web for Windows

GL

- Silicon Graphics (SGI) implemented 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.

OpenGL

- GL led to OpenGL (1992), a platform-independent API that was
  - Easy to use
  - Close enough to the hardware to get excellent performance
  - Focused on rendering
  - Omitted windowing and input to avoid window system dependencies

Modern OpenGL

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

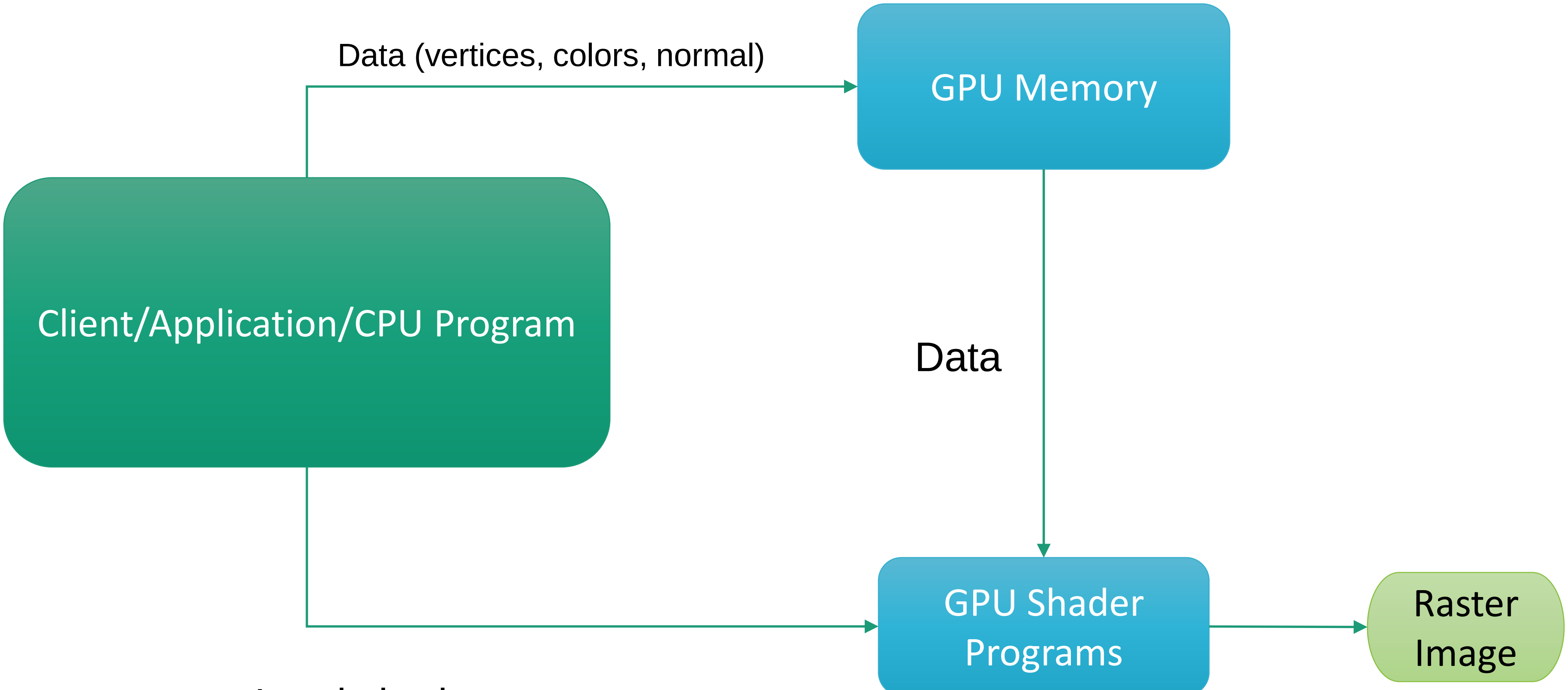
Retained vs. Immediate Mode Graphics

- This idea of how can we use the GPU brings us to two different approaches to graphics: immediate mode vs retained mode.
- Immediate mode:
  - Geometry is drawn when CPU sends it to the GPU.
  - All data must be re-sent if anything changes.
  - Once drawn, geometry on GPU is discarded.
  - Requires major bandwidth between CPU and GPU
  - Minimizes memory requirements on GPU
- Retained mode:
  - Geometry is sent to GPU and stored
  - It is displayed when directed by CPU
  - CPU may send transformations to move geometry.
  - Minimizes data transfers, but GPU now needs enough memory to store geometry.

Retained vs. Immediate Mode Graphics

- With advancements in GPUs, modern graphics usually use the retained graphics mode.
- As such we think of a graphics systems as a “client-server” system.
- Client (CPU)
  - Sends data to the server (GPU)
  - Makes requests to the server (GPU)
- Server (GPU)
  - Holds the data (send from the client, CPU)
  - Responds to requests by the client (CPU) (to draw)

Modern OpenGL



Load shader programs
Assign GPU data buffers to shader variables

OpenGL as a State Machine

- OpenGL treats the graphics system as a finite state machine (FSM)
- When we change a property through a function, that property's state is used until we change it
- In the past OpenGL relied heavily on states (even for drawing). Now most are eliminated.
  - Code is verrrrrry different!!! ☹
  - If you see anything with `glBegin` it's the old style of code that you can't use!

OpenGL Libraries

- OpenGL core library
  - OpenGL32 on Windows
  - GL on must Unix/Linux systems (libGL.a)
  - Functions start with gl.

OpenGL Windowing

- To interface with a window system and to get input, we need another library
  - This provides the “glew” between the OS windowing system and OpenGL
  - Different for each OS
    - For the X Windows System (Linux w/ window system) this is called GLX
    - For Windows, it is wgl
    - For Mac, it is agl
- Instead of using different libraries for each system, there are two readily available libraries:
  - OpenGL Extension Wrangler (GLEW)
    - Removes operating system dependencies
  - OpenGL Utility Toolkit (GLUT)
    - Provides minimum functionality for any windowing system

GLEW/GLUT

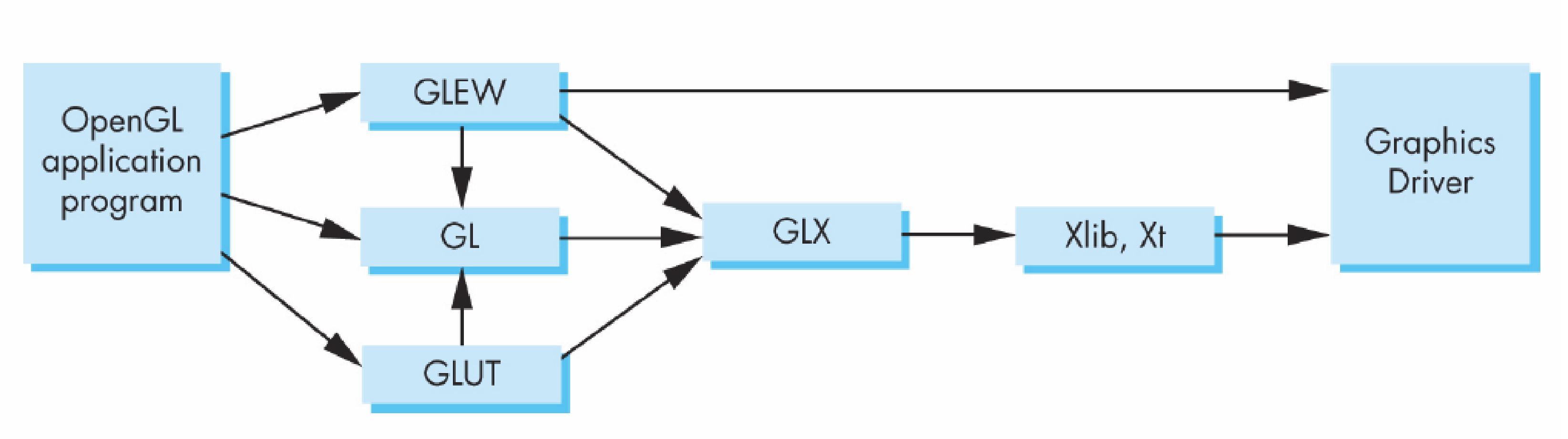
- GLEW
  - Makes it easy to access OpenGL extensions available on a particular system
  - Avoids having to have specific entry points into Windows code
  - Application needs only to include *glew.h* and run a `glewInit()`

- OpenGL Utility Tookit (GLUT)
  - Provides functionality common to all window systems
    - Open a window
    - Get input from mouse and keyboard
    - Menus
    - Even-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 remained unchanged
  - Some functionalities can't work since it requires depreciated functions
- freeglut updates GLUT
  - Added capabilities
  - Context checking

Software Organization



Setting up OpenGL Project

- See the handout for A0_OpenGLInstallation
  - This lists the necessary files for `glew`, `glut` and `freeglut`
  - Also discusses installation, linking and compiling for
    - Windows w/ Visual Studio (recommended)
    - Mac (Lion vs Leopards)
    - Linux (in particular tux with X11 forwarding)

Setting up Visual Studio

- First make sure you update your graphics driver!
  - This will give you the most recent versions of OpenGL
- Download GLEW and freeglut from the web
  - Just get the Windows binaries
  - Need *freeglut* since GLUT for Windows hasn't been updated in MANY years.

Setting up Visual Studio

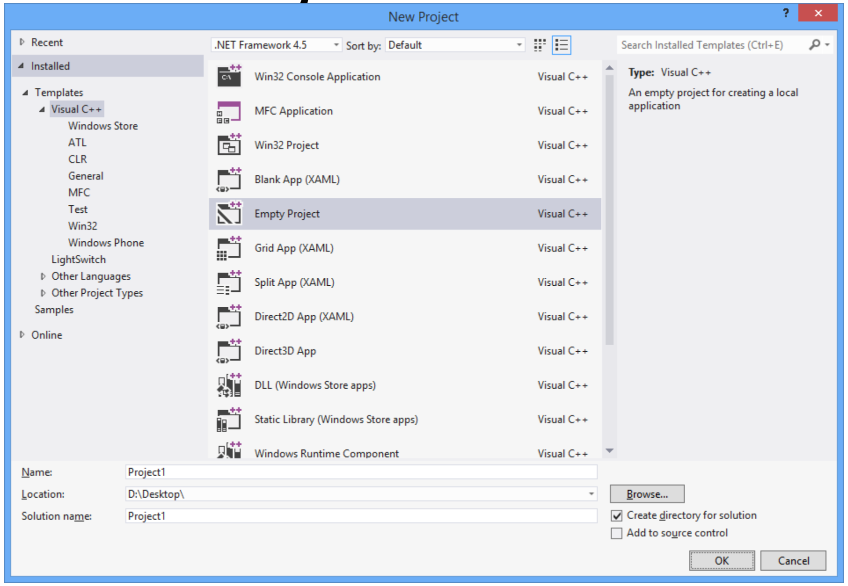
- Identify the Visual Studio folder for the version you are using (generally C:/Program Files/Microsoft Visual Studio <version>/VC where <version> is:
  - VS2005: 8.0
  - VS2008: 9.0
  - VS2010: 10.0
  - VS2012: 11.0
  - VS2013: 12.0
  - VS2015: 14.0

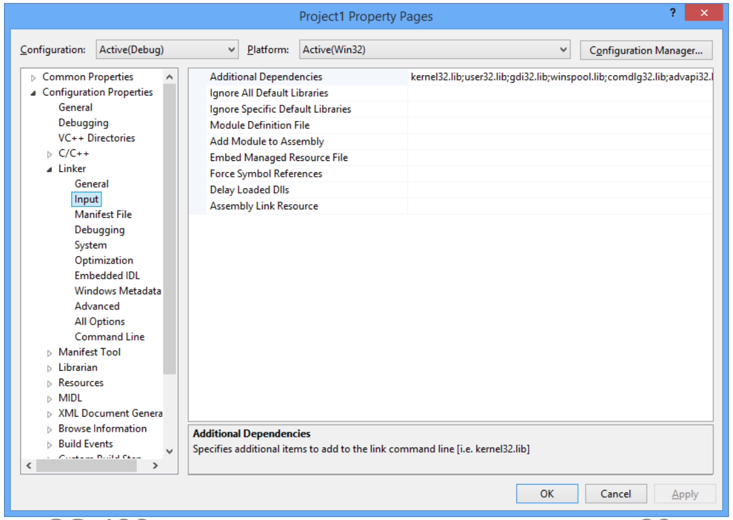
Setting up Visual Studio

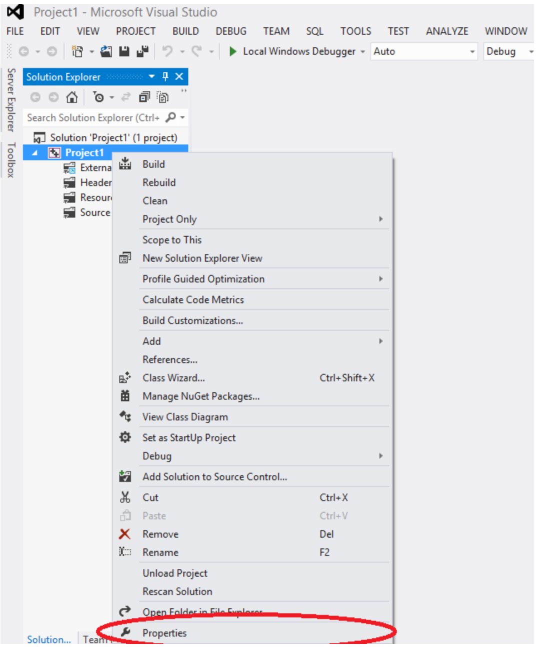
- Extract and copy files from the downloaded GLEW and freeglut directories (choose non-64-bit versions)
  - Put all .dll files in your VS folder’s /bin subfolder
  - Put all .h files in your VS folder’s /include/GL subfolder
  - Put all .lib files in your VS folder’s /lib subfolder

Visual Studio OpenGL Project

- Create an empty project
- Under project properties click on linker then on input





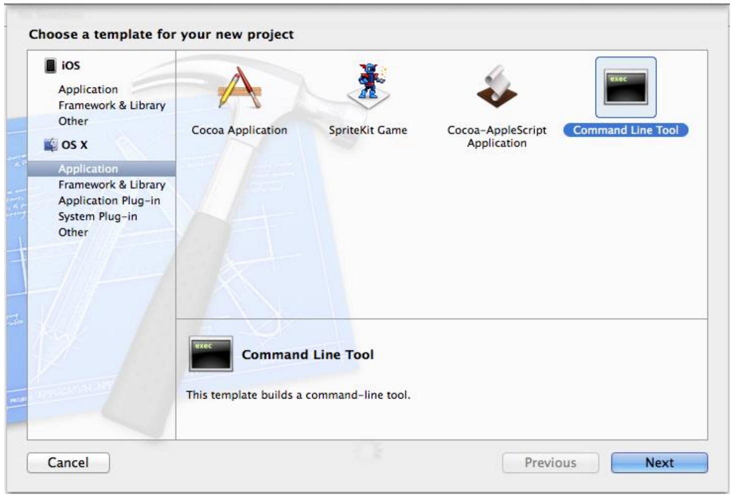


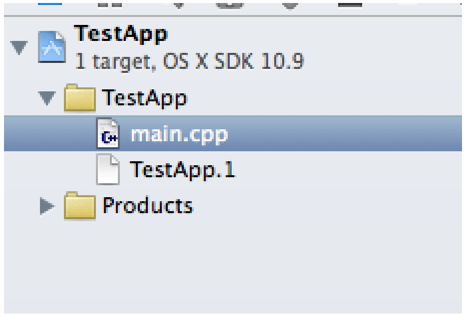
Visual Studio OpenGL Project

- Add glew32.lib and freeglut.lib to the end of the *Additional Dependencies* list
- Add the source files to your project
  - For the testing project this include:
    - Source Files → helloWorld537.cpp, InitShader.cpp
- Ensure that your project can find the *shader files* (for the test, these files are fshader00_v110.glsl and vshader00_v110.glsl)
  - If you're running from within VS copy them to where your project's .vcxproj file is.
  - If you're just clicking on the executable file, copy them to where that executable file is.

OSX Setup

- Current version of Xcode is 5.1.1
- Open Xcode, under File→New→Project select Application→Command Line Tool

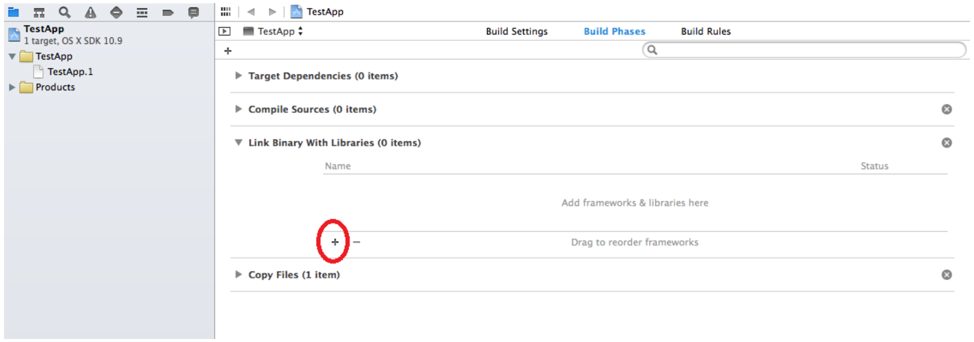


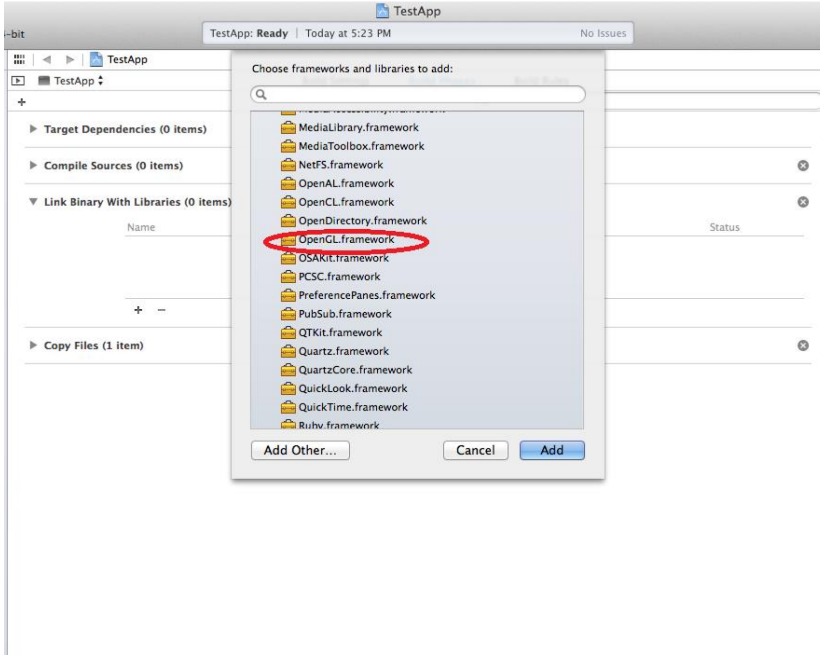


- Delete the main.cpp source file that is automatically made

OSX Setup

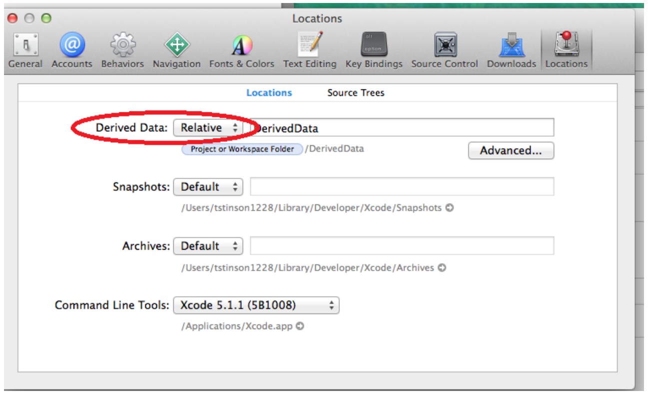
- Under *Build Phases* then *Link Binary With Libraries* click the "+" button at add the OpenGL and GLUT libraries (built into OSX)





OSX Setup

- By default Xcode builds to an odd directory.
  - To Preference then the Locations tab to see where this is
  - Consider changing the *Derived Data* directory to be *Relative*



- Like in Windows, you must copy the shader files to where the binary file is (within this Derived Data directory)

Linux/Tux Setup

- Your system needs the following libraries:
  - `LDLIBS = -lglut -lGLEW -lGL -lGLU`
  - Already installed on tux
- Other than that, you just create your makefile and run it
  - I'll provide a sample `makefile`
- Of course to run on tux you'll need to set up X11 forwarding
  - Windows: Xming + PuTTY
  - Mac: Sadly there's issues with doing x forwarding of OpenGL graphics on Macs
- **NOTE: The graphics cards on the tux cluster requires shader version 130 (as opposed to 150 provided). To get this to work all you need to do is change `#version 150` to `#version 130` in each of the *.glsl files.**