

CSCI 4239/5239

**Advanced
Computer
Graphics**

Spring 2019

Instructor

- Willem A (Vlakkies) Schreüder
- Email: willem@prinmath.com
 - Begin subject with 4239 or 5239
 - Resend email not answered promptly
- Office Hours:
 - ECST 121 Thursday 4-5pm
 - Other times by appointment
- Weekday Contact Hours: 6:30am - 9:00pm

Course Objectives

- Explore advanced topics in Computer Graphics
 - Pipeline Programming (Shaders)
 - Embedded System (OpenGL ES)
 - GPU Programming (CUDA&OpenCL)
 - Ray Tracing
 - Special topics
- Assignments: Practical OpenGL
 - Building useful applications
 - Use Qt to build professional apps



Course Organization and Grading

- Class participation (50% grade)
 - First hour: Discussion/Show and tell
 - Weekly homework assignments
 - Volunteers and/or round robin
 - Second hour: Introduction of next topic
- Semester project (50% grade)
 - Build a significant application in OpenGL
 - 10 minute presentation last class periods
- No formal tests or final
- You can skip ONE homework

Assumptions

- You need to be fluent in C/C++
 - Examples are in C++
 - You can do assignments in any language
 - I may need help getting it to work on my system
- You need to be comfortable with OpenGL
 - CSCI 4229/5229 or equivalent
 - You need a working Qt/OpenGL environment

Class Attendance

- Attendance is highly encouraged
- More of a seminar than a lecture
 - Participation is important
- I don't take attendance
- Lectures are available if you miss class
 - If you are sick stay home
- Lecture video access
 - In class students use Moodle
 - BBA students will be notified

Grading

- Satisfactory complete all assignments => A
 - The goal is to impress your friends
- Assignments **must** be submitted on time unless prior arrangements are made
 - Due by 8am Thursday morning
 - Grace period until Thursday noon
- Assignments must be completed individually
 - Stealing ideas are encouraged
 - Code reuse with attribution is permitted
- Grade <100 means not satisfactory (not A)

Code Reuse

- Code from the internet or class may be used
 - You take responsibility for any bugs in the code
 - That includes bugs in my code
 - Make the code your own
 - Understand it
 - Format it consistently
 - **Improve upon what you found**
 - **I may ask what improvements you made**
 - **Submitting code without crediting the source is violation of the CU honor code**
- The assignment is a minimum requirement

Code Expectations

- I expect professional standards in coding
 - Informative comments
 - Consistent formatting
 - Expand tabs
 - Clean code
- Good code organization
- Appropriate to the problem at hand
- See ***Homework Requirements*** on moodle

Text

- OpenGL Programming Guide (9ed)
 - Kessenich, Sellers & Schreiner
 - “OpenGL Vermillion Book”
 - Implementing Shaders using GLSL
 - Don't get an older edition
- Ray Tracing from the Ground Up
 - Kevin Suffern
 - Theory and practice of ray tracing
- Recommended by not required

Other Texts

- OpenGL SuperBible: Comprehensive Tutorial and Reference (7ed)
 - Sellers, Wright & Haemel
 - Good all-round theory and applications
- Graphics Shaders: Theory and Practice (2ed)
 - Bailey & Cunningham
 - Great shader examples

Other Texts

- OpenGL ES 3.0 Programming Guide
 - Ginsburg & Purnomo
 - “OpenGL Purple Book”
 - Has a chapter specific to the iPhone
- iPhone 3D Programming
 - Rideout
 - Great introduction to portable programs
- WebGL Programming Guide
 - Matsuda & Lea

Other Texts

- Programming Massively Parallel Processors
 - Kirk & Hwu
 - Explains GPU programming using CUDA
 - Shows how to adopt OpenCL
- CUDA by Example
 - Sanders and Kandrot
 - Great introduction using examples

Other Texts

- Advanced Graphics Programming Using OpenGL
 - Tom McReynolds and David Blythe
 - Great reference for miscellaneous advanced topics

OpenGL Resources

- www.google.com
 - Need I say more?
- www.opengl.org
 - Code and tutorials
- nehe.gamedev.net
www.lighthouse3d.com
 - Excellent tutorials
- www.mesa3d.org
 - Code of “internals”
- www.prinmath.com/csci5229
 - Example programs from CSCI 4229/5229

Assignment 0

- Due: **Friday** Jan 18 by 9pm
- Sign up with moodle.cs.colorado.edu
 - Enrollment key: 42395239
 - A picture will help me remember your names
- Submit
 - Your study area
 - Platform (Hardware, Graphics, OS, ...)
 - Any specific interests in computer graphics
 - Specific topics you want to see covered
 - Initial project idea(s)
 - BBA students let me know about schedules

My information

- Mathematical modeling and data analysis
 - PhD Computational Fluid Dynamics [1986]
 - PhD Parallel Systems (*CU Boulder*) [2005]
 - President of *Principia Mathematica*
- Use graphics for scientific visualization
- Open source bigot
- Program in C, C++, Fortran, Perl & Python
- Outside interests
 - Aviation
 - Amateur radio

Hardware Requirements

- You need hardware that will run shaders well
 - Integrated graphics may not be adequate
 - Graphics cards from the last 5 years should be OK
 - GPU computing needs high end hardware
 - A VM is probably not going to cut it
- Try on different hardware
 - AMD & nVidia sometimes behave differently
 - Try it on my machine during office hours if you are presenting

Why Qt

- Why drop GLUT?
 - It is easy to use, but limited capabilities
 - Apple is dropping GLUT
- Pros
 - It is cross platform: Linux/WinX/OSX/iOS/...
 - Provides framework for professional apps
 - Supports controls, sound, image loading, etc
- Cons
 - Hides some of the OpenGL elements
 - Steeper learning curve than GLUT

OpenGL Extension Wrangler (GLEW)

- Maps OpenGL extensions at run time
 - Provides headers for latest OpenGL
 - Finds vendor support at run time
- Check support for specific functions or OpenGL version at run time
 - Crashes if unsupported features are used
- Use only if you have to (Windows mostly)
 - Set -dUSEGLEW to selectively invoke it
 - Do NOT require GLEW (I don't need it)
 - For MinGW see moodle instructions

Assignment 1

- Due: Thursday January 24
- NDC to RGB shader
 - For every point on the objects, the color should be determined by its position in normalized device coordinates
- The goal is to make this as short and elegant as possible
 - Shader Golf
 - Figure this out for yourself
- Figure out how to do this with Qt

Nuts and Bolts

- Complete assignments on any platform
 - Assignments reviewed under Ubuntu 18.04.1 LTS
 - Ubuntu provides Qt version 5.9.5
- Submit using `moodle.cs.colorado.edu`
 - ZIP or TAR
 - Name projects hw1, hw2, ...
 - Create a .pro file named *hwX.pro*
 - Set window title to *Assignment X: Your Name*
- Include number of hours spent on assignment
- ***Check my feedback and resubmit if requested***

Project

- Should be a program with a significant graphics component
 - Something useful in your research/work
 - Graphical front end to simulation
 - Graphical portion of a game
 - Expect more from graduate students
- Deadlines
 - Proposal: Thursday March 21
 - Progress: Thursday April 4
 - Review: Thursday April 18
 - Final: **Tuesday April 30**

A few hints

- My machine runs Linux x86_64
 - gcc/g++ with nVidia & GLX
 - -Wall is a **really** good idea
 - case sensitive file names
 - int=32bit, long=64bit
 - little-endian
 - fairly good performance
- How to make my life easier
 - Try it on another machine
 - Stick to C/C++ unless you have a good reason
- **Maintain thy backups...**

Class Discussions

- If have a special interest in the topic and have something special to contribute
VOLUNTEER to lead the discussion
- If by Sunday there are no volunteers, I will appoint volunteers some on a round robin basis (in order by MD5 of names)
 - You can trade places, but **you** are responsible for arranging a substitute
- Everybody should do this at least once, but you can do more if you want
 - BBA students Zoom or Skype or screencast
- Popular topics may have more presenters

What to Present

- Should be (mostly) the assigned topic
 - Rabbit holes can be very interesting
 - Keep it within reach of the class
- Show what you did for the assignment
 - Cover principles or theory I omitted
 - Show and describe code of interest
 - Demonstrate “gotchas” you encountered
 - Impress your friends
- Keep it interesting

How to Present

- 15 minutes can be forever or over in a wink
 - Plan your time (practice out loud)
 - If you use slides figure 2 minutes per slide
- Plan your presentation
 - What are the key points you want to convey?
 - How do you illustrate the key points?
- The presentation should TEACH
 - Teaching is learning twice
 - Adapt to the questions

How to Listen

- If you don't understand, ask
 - Helps the presenter understand what's new to you
- If you disagree, say so
 - Maybe the presenter misspoke or has an different opinion worth discussing
- Be nice – you may be next!

BBA Students

- Suggest ways you can present remotely
- Provide screen cast or similar demonstration
- Zoom or Skype or other desktop sharing
 - Performance may be an issue
- Stick to the class schedule if possible

Parallel Flight Simulator Project

- Consider joining a project with many members
 - Each member has a specific subtask
 - World visualization
 - Special effects
 - Flight dynamics
 - Multi-function displays (instruments)
 - Networking
 - Flight controls
 - Sound
 - Rotating project manager
 - Responsible for managing the project for a week
 - Provide concise report of what was done the last week
 - Lay out a plan for what should be done the next week
- Somewhat like a real software project
 - I will be the client

What is a Shader?

- A shader is a computer program that runs on the GPU to calculate the properties of vertexes, pixels and other graphical processing
- Examples:
 - Vertex position or color computed by a program
 - Texture generated by a program
 - Per-pixel lighting
 - Image processing
 - Cartoon shading

How does a shader work?

- Shader Language used to specify operations
 - RenderMan, ISL, HLSL, Cg, GLSL
- Compile instructions into program
 - e.g. `glCompileShader()`
- Shader performs calculations as part of graphics pipeline
- Runs calculations on GPU instead of CPU

What is a Shader Language?

- Typically C/C++ like
 - for, while, if, ... for control flow
 - Adds special types like vec4 (4 component vector) and mat4 (4x4 matrix) and operators
 - Predefined variables used to get data (gl_Vertex) and return result (gl_Position)
- Simplifies and extends C/C++ for efficiency
 - Matrix & vector operations supported in hardware Graphics Processing Unit (GPU)
 - Built-in functions like normal, blend, etc.

GL Shader Language (GLSL)

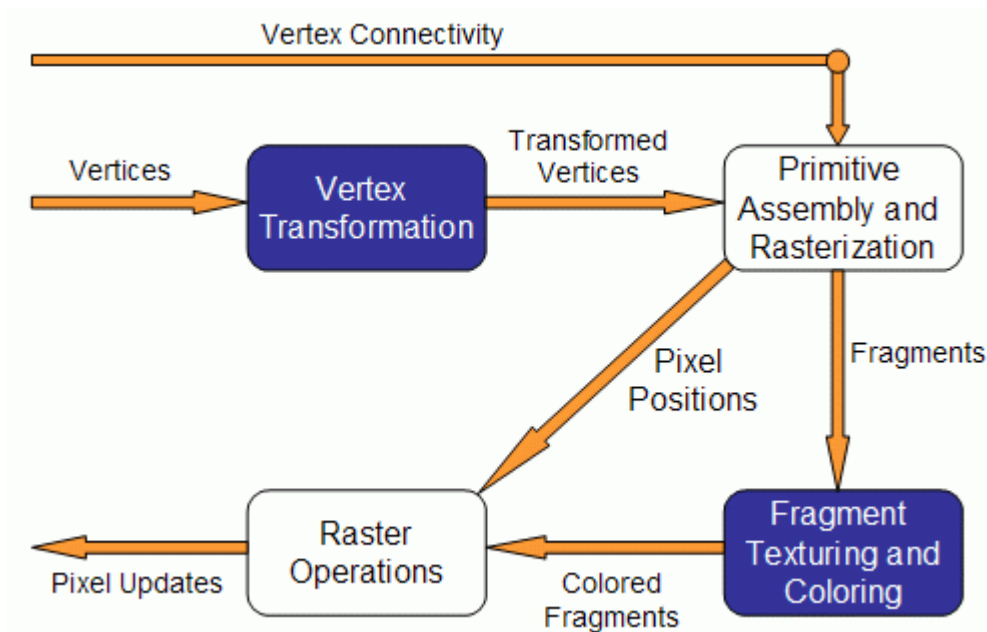
- Often call “GLSLang”
- Added to OpenGL 2.0
 - First appeared as extension in OpenGL 1.4
 - Can be accessed in older versions using extensions
 - GL Extension Wrangler (GLEW) often used
- Geared to real time graphics
 - Inserted into OpenGL pipeline
 - Vertex Shader to manipulate vertexes
 - Fragment Shader to manipulate pixels

OpenGL Deprecation

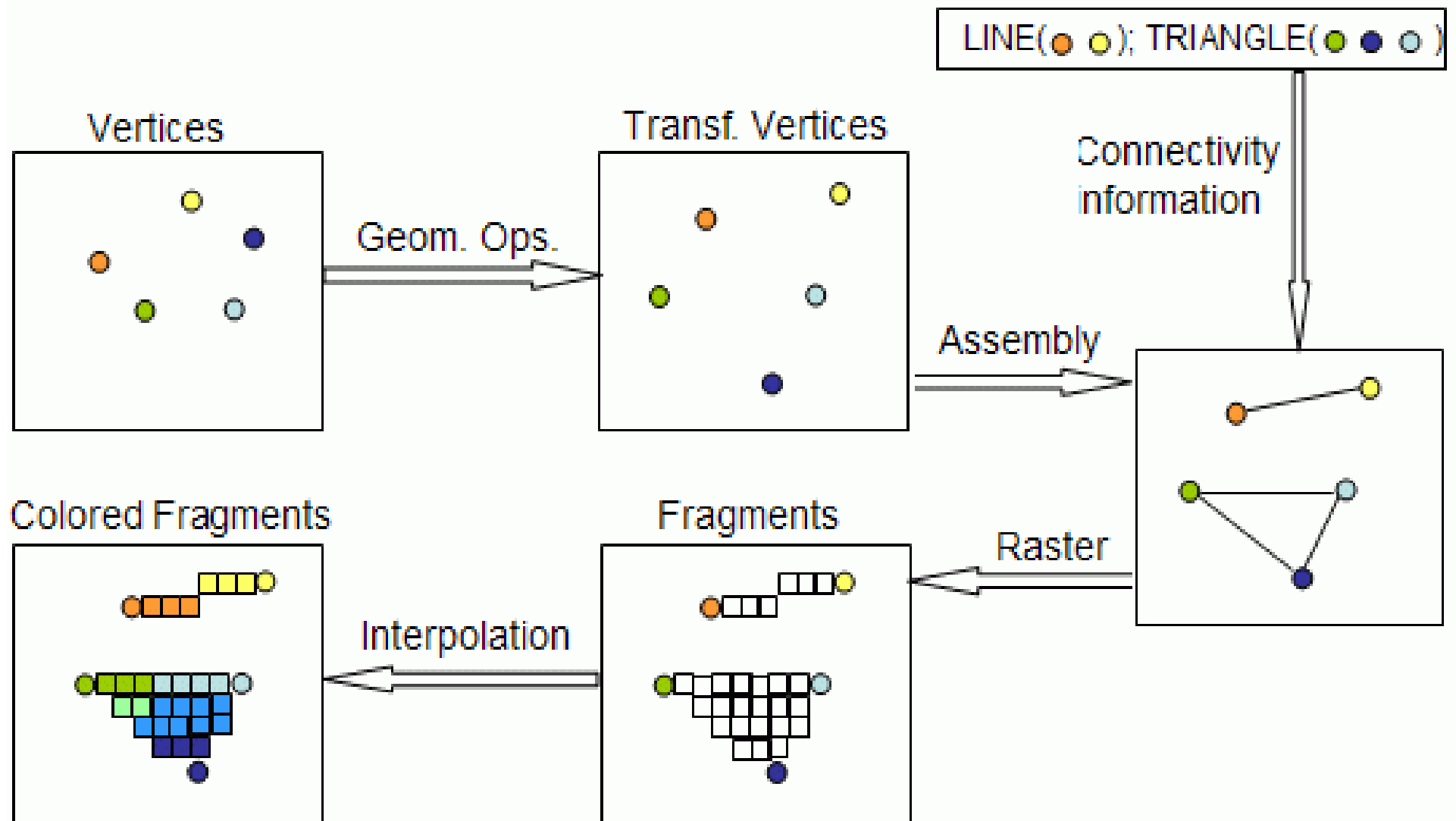
- I will mostly use OpenGL 2.x
 - Feature rich
 - Flat learning curve
 - More advanced examples will use 3.x and 4.x
- OpenGL Core Profile concentrates on rendering
 - Improved execution time performance
- User must provide deprecated functionality
 - Steepens the learning curve
 - Deprecated features in Compatibility Profile
 - Increases reliance on third party libraries

Where does GLSL fit?

- Vertex shader
 - Transformations, color, texture coordinates, ...
- Fragment shader
 - Textures, Color Interpolation, Fog, ...
- OpenGL still does Z-buffering, etc.



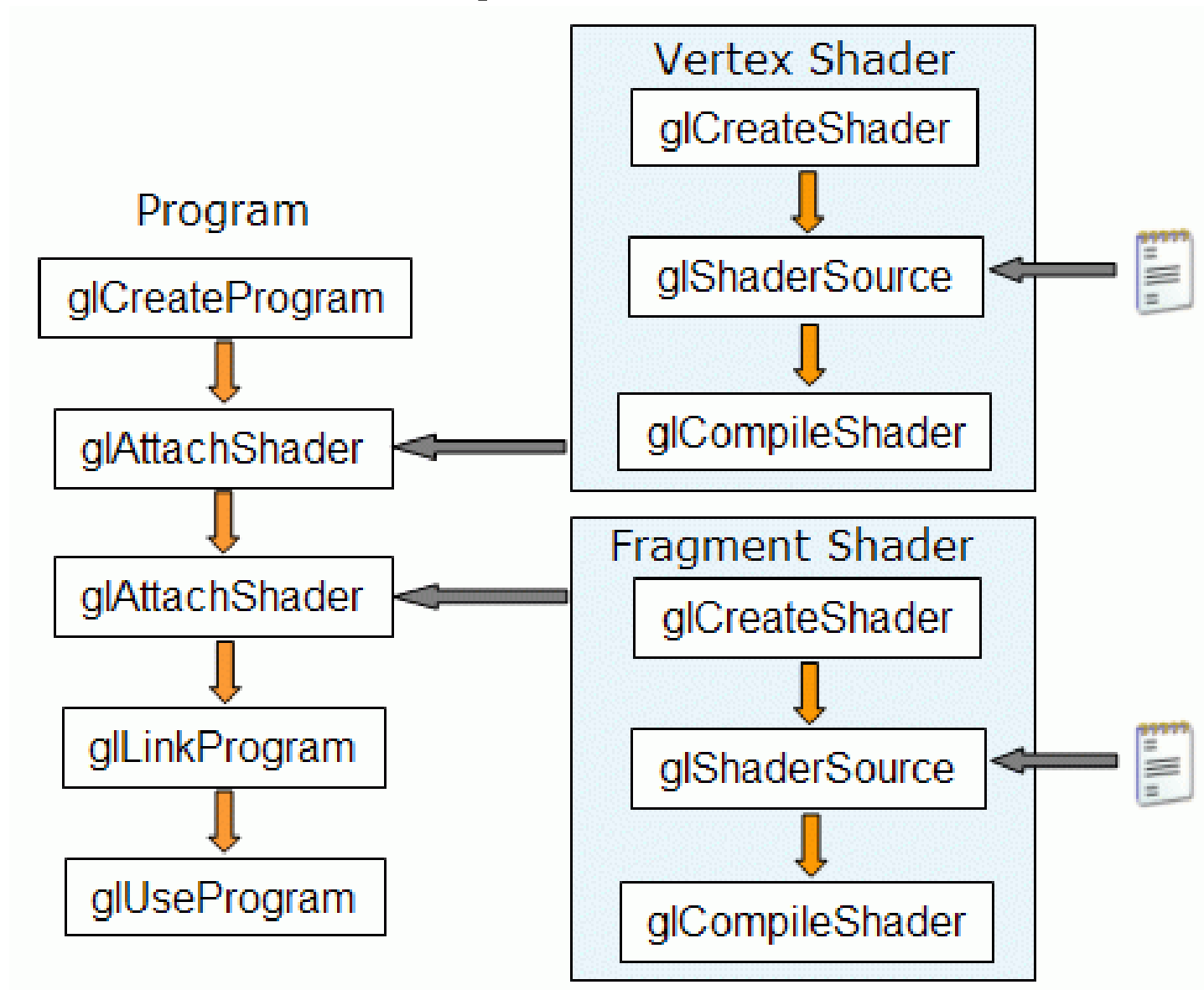
Fixed Pipeline Example



How is this different from what we have done before?

- GLSL instructions can run on GPU
 - Matrix-vector multiplications done *fast*
- Without GLSL we influence the pipeline using parameters and fixed operations
 - Lighting calculated at vertexes
 - Textures calculated at fragments
 - Vertex-fragment interpolation
 - GL_SMOOTH bilinear interpolation
 - GL_FLAT constant using last vertex
- With GLSL we can calculate values directly

How does this work with OpenGL?



Other Shader Languages

- RenderMan
 - Lucasfilm - Pixar - Disney
- OpenGL Shader (ISL)
 - SGI Interactive Shader Language
- High-Level Shader Language (HLSL)
 - Microsoft DirectX 9
- NVIDIA's Cg
 - proprietary shading language

RenderMan

- First practical shading language (1988)
- De-facto entertainment industry standard
- Remains in widespread use today
- Generally used for off-line rendering
 - Uncompromising image quality
 - Little hardware acceleration
- Credits:
 - Jurassic Park, Star Wars Prequels, Lord of the Rings
 - Toy Story, Finding Nemo, Monsters Inc, ...
- No relation to OpenGL in syntax or structure

The Rest (ISL, HLSL, Cg, ...)

- Syntax different but similar approach
- Generally similar in structure
 - Vertex Shader
 - Fragment Shader
- Geared towards real time graphics
 - Hardware support
 - Performance stressed

GLSL Versions

- GLSL 1.0 = OpenGL 1.4 (2002)
 - The first portable shader
- GLSL 1.2 = OpenGL 2.0 (2004)
 - The shader we will use
- GLSL 1.3 = OpenGL 3.0 (2008)
 - Some changes in syntax
 - Deprecates some features
- GLSL 3.3 = OpenGL 3.3
 - From here on GLSL version match OpenGL
- Set minimum version using `#version`

GLSL 1.2 Variable Qualifiers

- uniform (e.g. `gl_ModelViewMatrix`)
 - input to vertex and fragment shader from OpenGL or application [read-only]
- attribute (e.g. `gl_Vertex`)
 - input per-vertex to vertex shader from OpenGL or application [read-only]
- varying (e.g. `gl_FrontColor`)
 - output from vertex shader [read-write], interpolated, then input to fragment shader [read-only]
- const (e.g. `gl_MaxLights`)
 - compile-time constant [read-only]

The problem with shaders

- EXTREMELY hard to debug
 - No “print” statements
- You have to have to do lighting yourself
- Support is spotty
 - GLSL requires OpenGL 2.0 or extensions
 - Some features are very new
 - Generally needs decent hardware
- So why use it?
 - Ultimate flexibility
 - Unsupported features (e.g. bump maps)

Installing Qt

- Get Qt 5 from <http://www.qt.io/download/>
 - Open source is free but requires sharing
- Ubuntu:
 - apt-get install qt5-default
- OSX
 - Install Xcode with command line tools
 - Install Qt
- Windows
 - Install mingw
 - Install Qt

Using Qt

- Use QOpenGLWidget
 - Introduced in Qt 5.4
 - Replaces older QGLWidget
 - The changes are mostly internal
 - Methods are the same, so painless migration
- Expand on my examples
 - Starting with my examples as a framework flattens the learning curve
 - Make sure you understand what is going on
 - CUgl class used in later examples

Building *hw01* with Qt

- Create *hw01.pro*
- Edit source code
- Run *qmake hw01.pro* to build makefile
- Compile using *make*
- Run *hw01*
- Before ZIPing run *make distclean*