CSCI 4239/5239

Advanced Computer Graphics

Spring 2024

Instructor

- Willem A (Vlakkies) Schreüder
- Email: vlakkies@colorado.edu
  - Begin subject with 4239 or 5239
  - Resend email not answered promptly
- Office Hours:
  - Monday 3-4pm by Zoom
  - Thursday 11am-noon by Zoom or in ECOT 732
  - 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 GLFW to build programs

Rasterized
Ray traced

Course Organization

- Tuesday:
  - Discussion of previous homework
  - Presentations
  - Volunteers and/or round robin
- Thursday: Introduction of next topic
  - Lecture
  - Example programs

Ungrading

- You self-assign the final grade
  - I may adjust it if I disagree
- After each Tuesday class period
  - Set SMART goals for the next week
  - Evaluate your goals from the pervious week
  - Keep a record of these weekly evaluations that I will review with you as part of the final project during the last week of class
  - This should be a 10 minute activity

S.M.A.R.T. Goals

- Summarize your goals each week
  - **S**pecific - what you want to achieve
  - **M**easureable - evaluating success
  - **A**chievable - doable in a week/semester
  - **R**elevant - somewhat on topic
  - **T**imely - plan your week
- You weekly journal should follow this pattern

Assumptions

- You need to be fluent in C/C++
  - Examples are in C or 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 OpenGL environment

Class Attendance

- **Attendance is expected**
  - I don’t typically take attendance
- More of a seminar than a lecture
  - Participation is important
  - Mute your microphone when on Zoom
  - Have your video on, expecially when talking
- If you are sick stay home
  - Zoom lectures are recorded
  - Catch up if you miss class

Grading Expectations

- The goal is to impress your friends
- Assignments **must** be submitted on time unless prior arrangements are made
  - Due by 23:59 Monday
  - Grace period until Tuesday at 8:00am
- Assignments must be completed individually
  - Stealing ideas are encouraged
  - Code reuse with attribution is permitted
- I will review your submissions before class

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
    - Clean out unused code
- Good code organization
- Appropriate to the problem at hand
- See ***Expectations*** on Canvas
- You need to understand ***every line***

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
- 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
- Physically Based Rendering
  - Pharr, Jakob and Humpfreys
  - Only for PBRT homework
  - 3rd edition for PBRTv3

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: **Wednesday Jan 18** by 23:59
- Check your Canvas notification settings
  - Set notifications to immediate
- 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)
  - Does office hours work for you?
  - Distance students let me know about attendance

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 be marginal
  - 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/Intel sometimes behave differently
  - I have nVidia hardware

Examples use glfw

- Why drop GLUT?
  - Apple support for GLUT is waning
  - It is easy to use, but limited capabilities
- Why glfw
  - It is cross platform: Linux/WinX/OSX/iOS/...
  - Very light weight wrapper to OpenGL
  - Does not do sound, load images, etc
  - Actively being developed (Vulcan is coming...)
- Can I use SDL or another wrapper?
  - As long as it is cross platform

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)
  - See Canvas for installation instructions

Installing glfw

- *http://www.glfw.org/*
- Ubuntu:
  - apt-get install glfw3-dev
- OSX
  - Install Xcode with command line tools
  - Install homebrew
  - Install toolchain, glfw and glew
- Windows
  - Install MSYS2/MinGW
  - Install toolchain, glfw and glew with pacman

CSCIx239 Library

- Includes GLFW and GLEW headers
- Many convenience functions
  - InitWindow starts GLFW and GLEW
  - Projection, Print, Fatal, ErrCheck, ...
  - Load textures and OBJs
  - Simple objects (Cube, Sphere, ...)
  - Compile Shaders
  - Matrix operations
  - Performance (FPS, elapsed)
- **Make sure you know what it does**

Assignment 1

- Due: Monday January 22
- 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
  - Make every operation count
- Test your toolchain

Nuts and Bolts

- Complete assignments on any platform
  - Assignments reviewed under Ubuntu 22.04.3 LTS
  - Ubuntu provides glfw 3.3
- Submit using Canvas
  - ZIP without creating an extra folder
  - Name projects hw1, hw2, ... (lower case)
  - Include all source code, makefile and data files
  - Set window title to *Homework 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: Wednesday March 20
  - Progress: Wednesday April 10
  - Review: Monday April 22
  - Final: Wednesday May 1

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 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
- You must present homework at least twice, but you can do more if you want
- 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

- 20 minutes can be forever or over in a wink
  - Plan your time (practice out load)
  - 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 is 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!

Zoom Etiquette

- Try to arrange a quiet background
- **Mute your audio when joining Zoom**
- Turn on video when talking
  - Leave your video on at other times if you can
- Interrupt me if I miss your chat question

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)

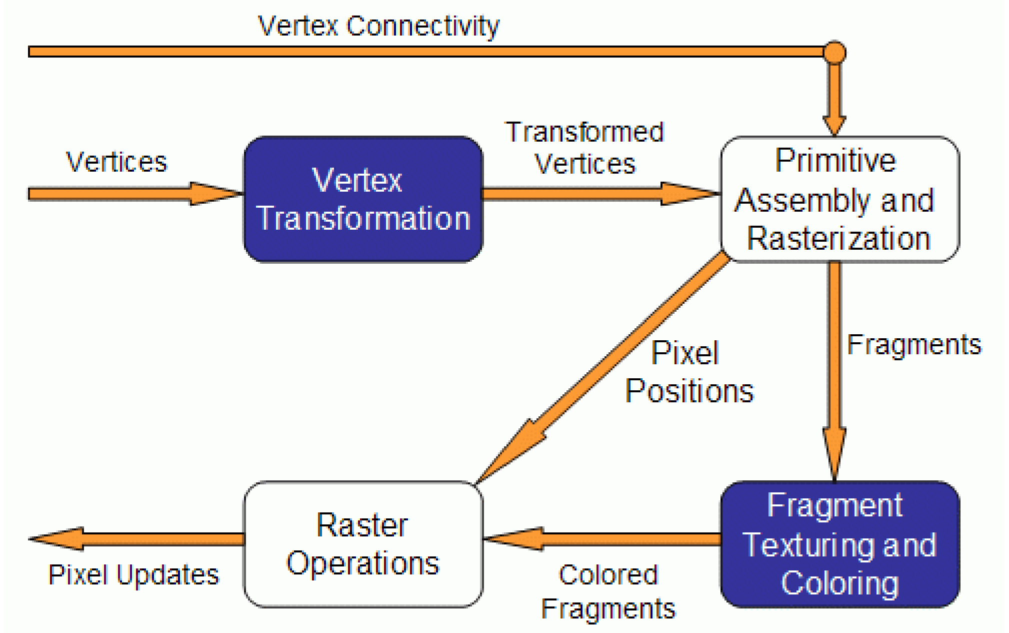
- Often call “GLSLang”
- Added to OpenGL 2.0
  - First appeared as extension in OpenGL 1.4
  - Can be accessed in older versions using extentions
  - GL Extension Wrangler (GLEW) often used
- Geared to real time graphics
  - Inserted into OpenGL pipeline
  - Vertex Shader to manipulate vertexes
  - Fragment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

OpenGL Verions

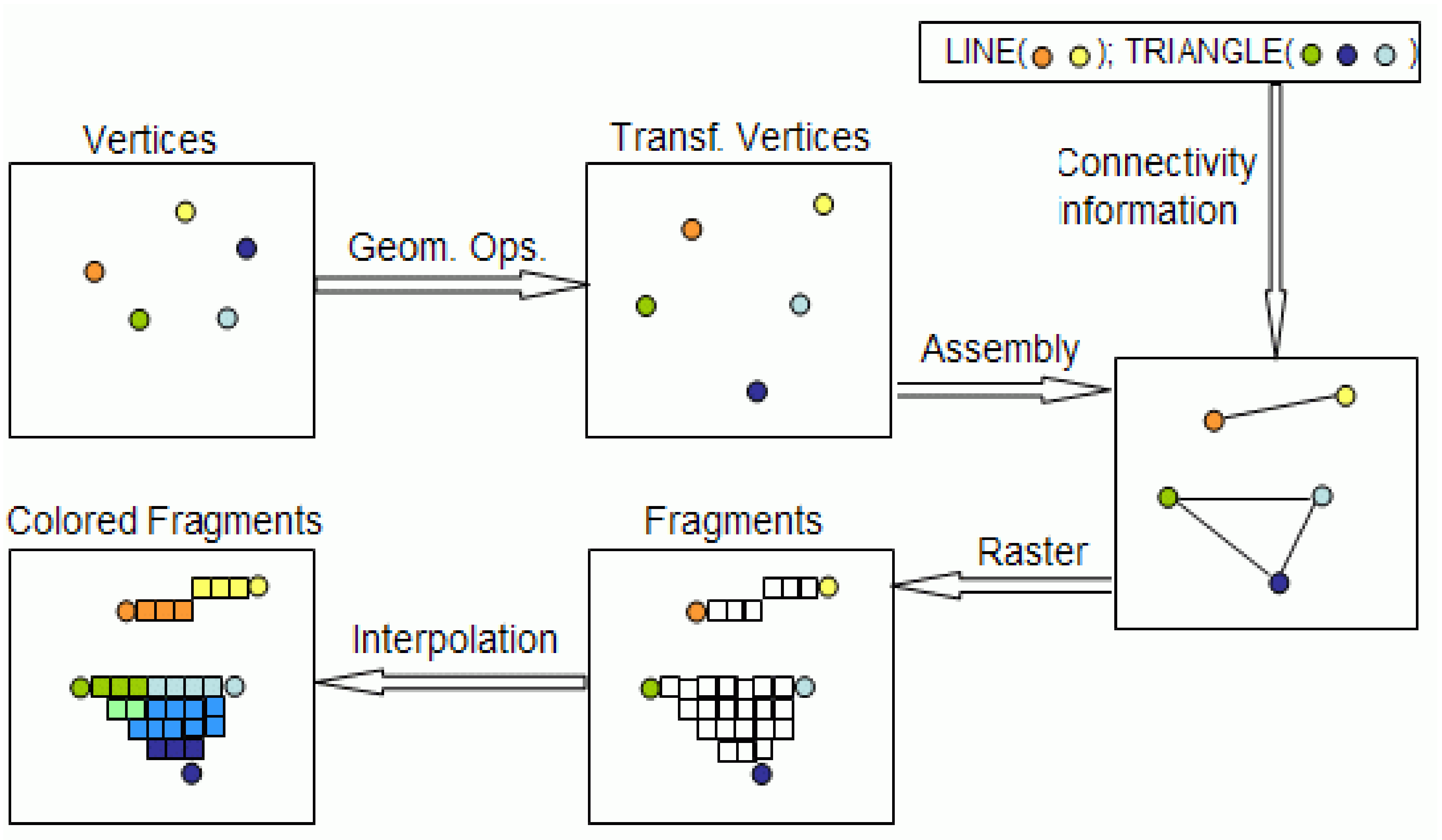
- I will use different OpenGL versions depending on what is convenient for the problem at hand
  - OpenGL 2.x
    - Feature rich
    - Flat learning curve
    - Convenient in many applications
  - OpenGL 4.x
    - Somewhat different syntax
    - Needed for advanced shaders
- OpenGL Core & Compatibility Profiles
- **You can use whatever version you want**

Where does GLSL fit?

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



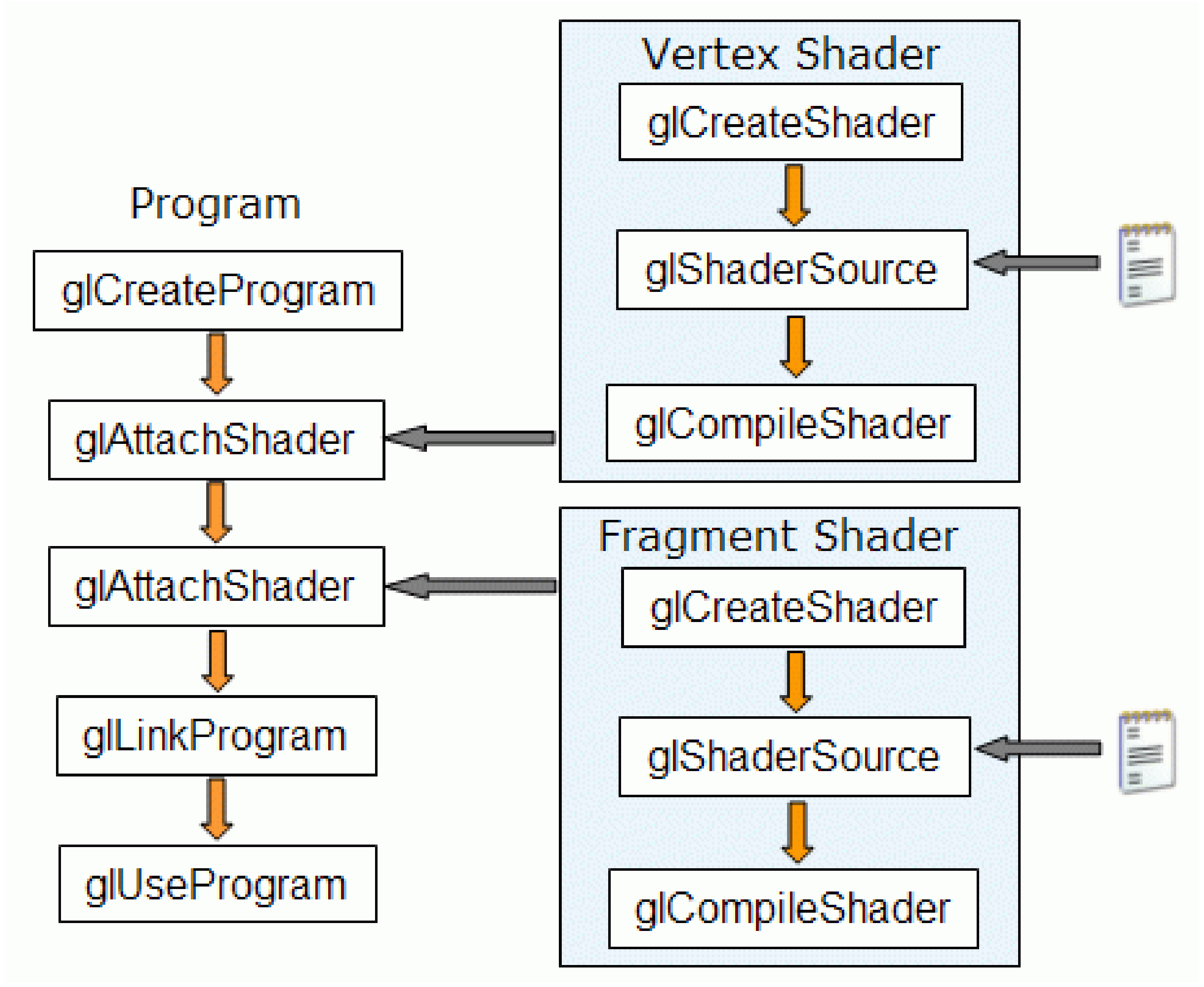
Fixed Pipeline Example



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-frament 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

- const (e.g. gl_MaxLights)
  - compile-time constant [read-only]
- 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]

GLSL 4 Variable Qualifiers

- const
  - compile-time constant
- uniform
  - data from CPU to shader
- in
  - per-vertex input to vertex shader
  - input from previous shader for others
- out
  - resulting vertex and fragment properties
  - output to next shader

The problem with shaders

- EXTREMELY hard to debug
  - No “print” statements
- You have to have to most things yourself
- Support for latest features are spotty
  - Needs GLEW on Windows
  - Generally needs decent hardware
- So why use it?
  - Ultimate flexibility
  - Unsupported features (e.g. bump maps)