Shaders

CSCI 4229/5229
Computer Graphics
Fall 2012

What is a Shader?

- Wikipedia:
  - A shader is a computer program used in 3D computer graphics to determine the final surface properties of an object or image. This often includes arbitrarily complex descriptions of texture mapping, light absorption, diffusion, reflection, refraction, shadowing, surface displacement and post-processing effects.
- Examples:
  - Vertex color computed by a program
  - Texture generated by a program instead of image

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)

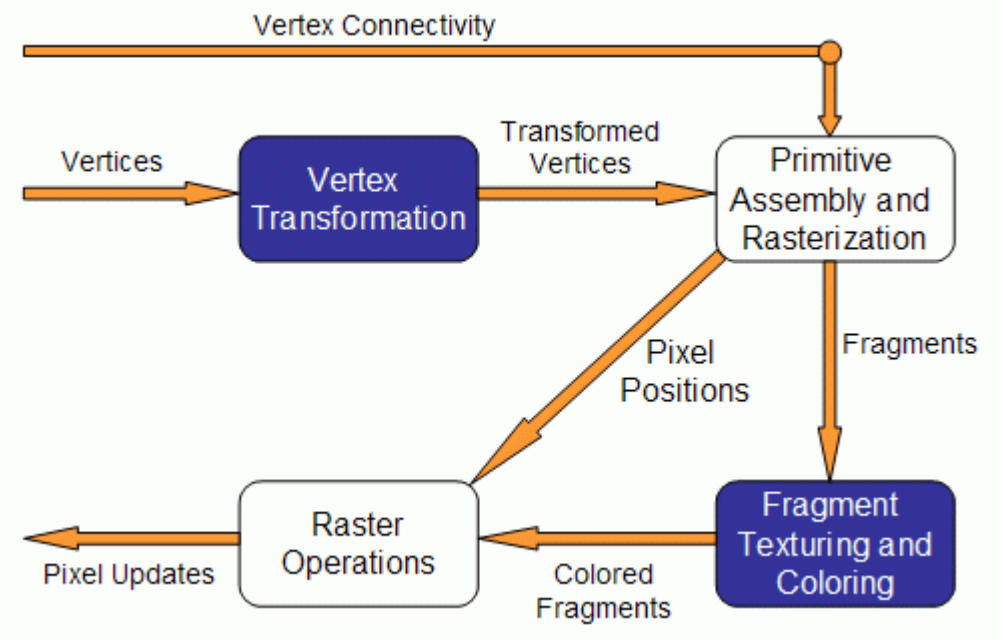
- Often call “GLSLang”
- Added to OpenGL 2.0
  - First appeared as extension in OpenGL 1.5
  - Can be accessed in older versions using extentions
  - GL Extension Wrangler (GLEW) often used
- Geared to real time graphics
  - Inserted into OpenGL pipeline
  - Vertex Shader to manipulate vertexes
  - Fragment Shader to manipulate pixels

GLSL Resources

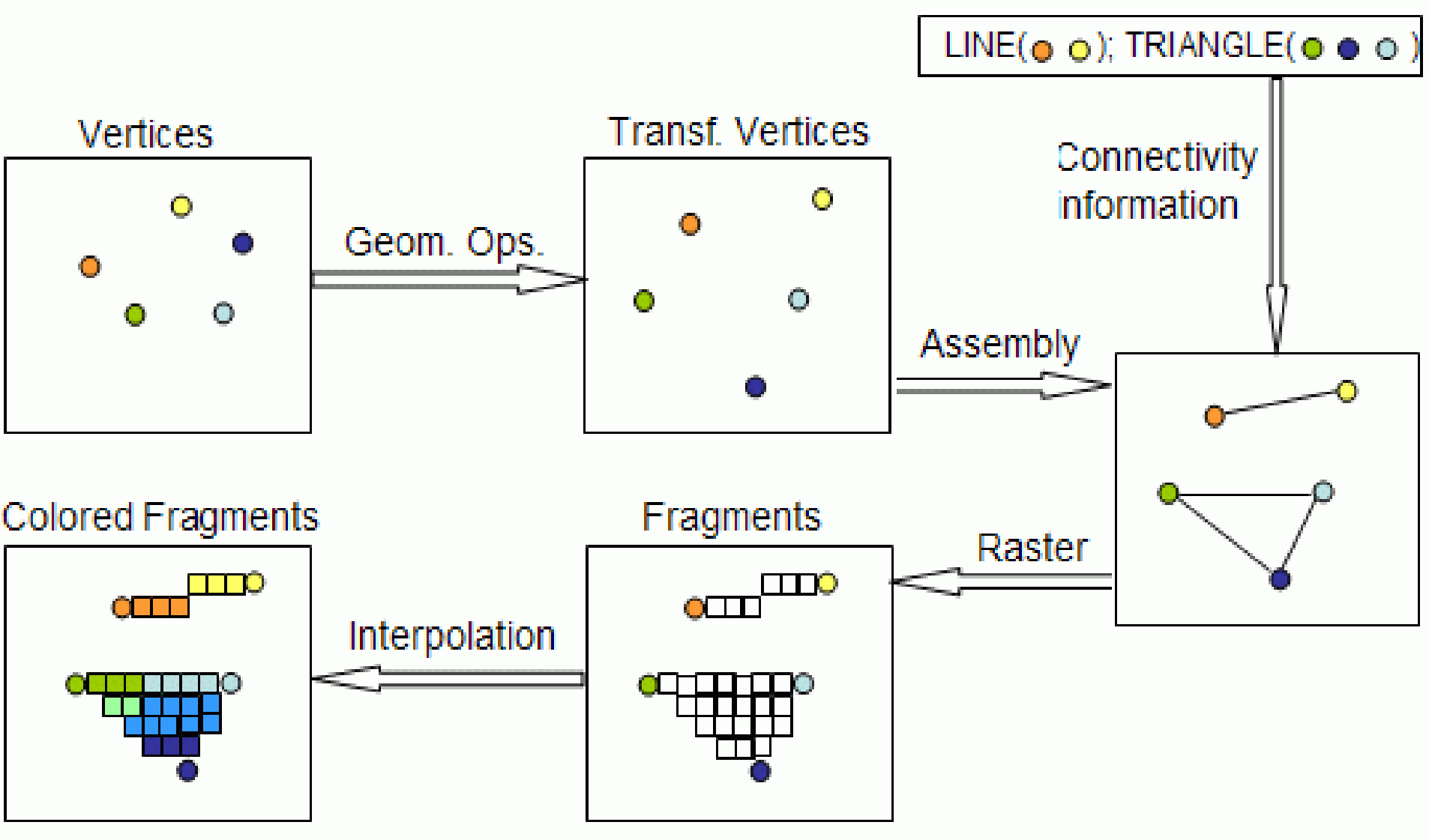
- Red Book
  - Chapter 15 is a great introduction to GLSL
  - Appendix I is a concise language reference
- Orange Book (3ed)
  - Very detailed
  - Not for beginners
- GLSL Quick Reference
  - “Cheat sheet”
- Many online references (some < OpenGL 2.0)
  - http://www.lighthouse3d.com/opengl/glsl/

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



GLSL Vertex Shader Example

```
uniform float t;        //  Time (passed in from application)
attribute vec4 vel;  // Velocity
const vec4   g = vec4(0.0,-9.8,0.0);   //  Gravity
void main()
{
    vec4 position = gl_Vertex;
    position += t*vel + t*t*g;
    gl_Position = gl_ModelViewProjectionMatrix * position;
}
```

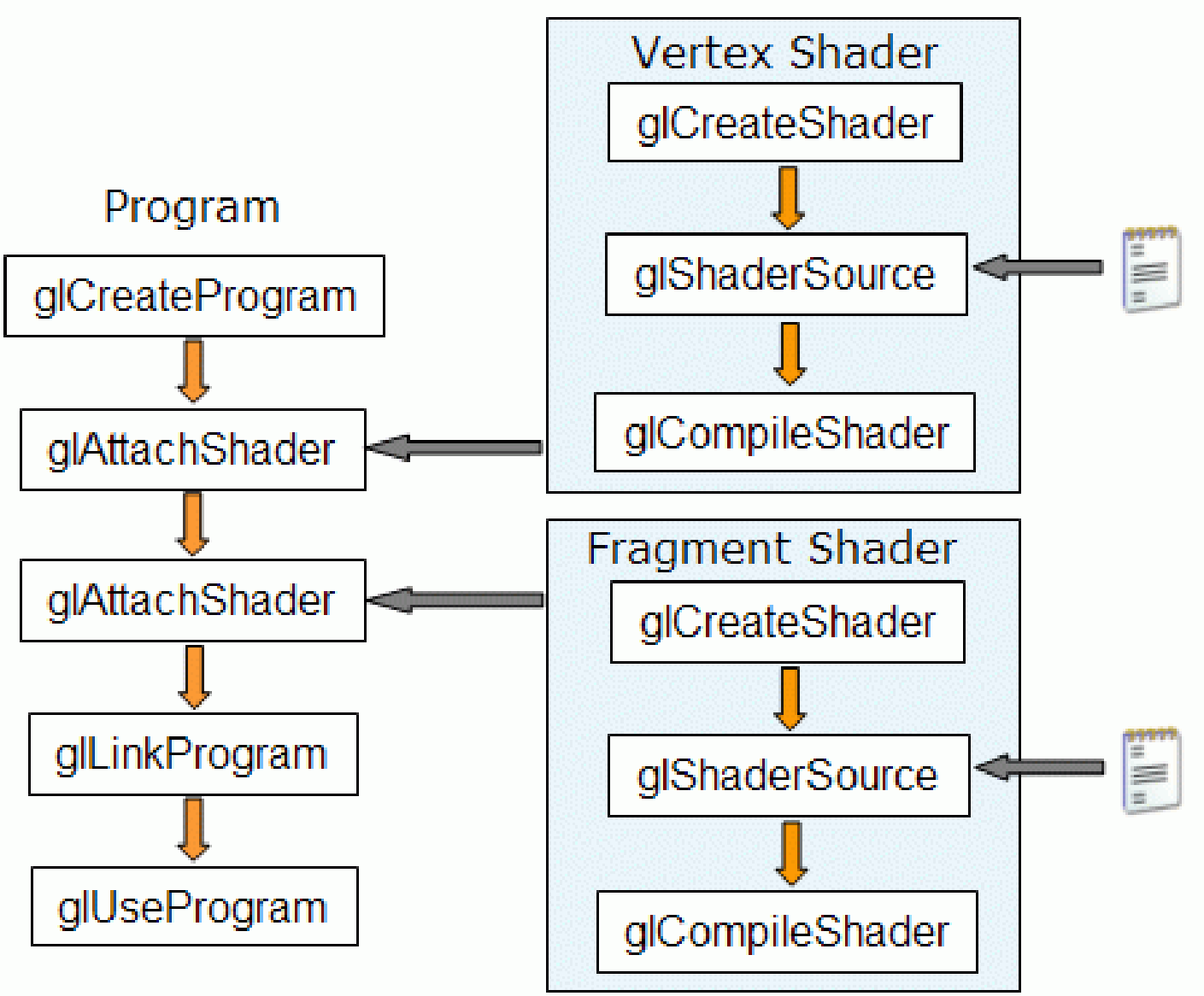
GLSL Variable Qualifiers

- 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]

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

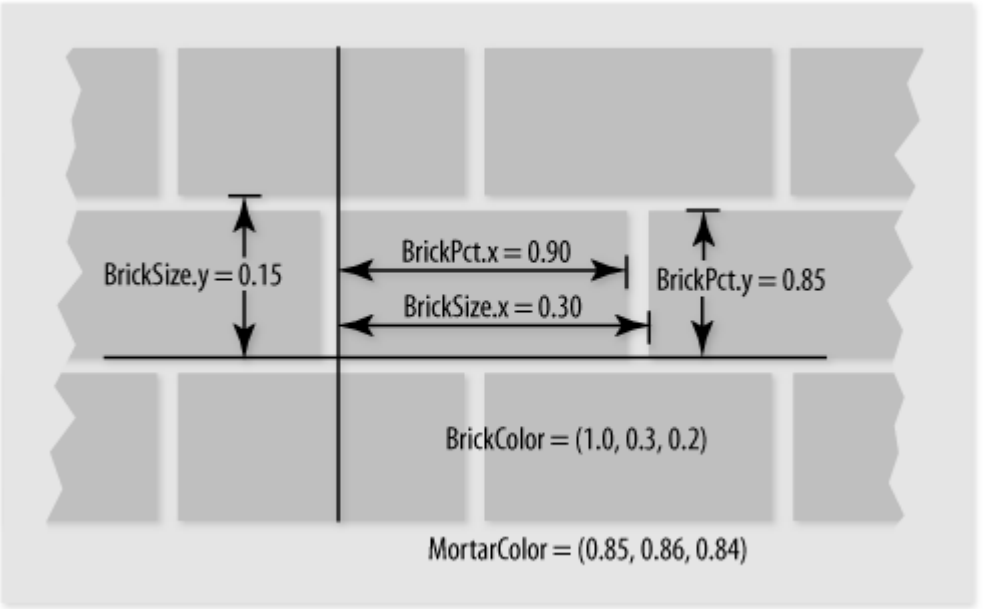
- First practical shading language (1988)
- De-facto entertainment industry standard
- Remains in widespread use today
- Generally used for off-line rendering
  - Uncompromising image quality
  - Little hardware acceleration
- Credits:
  - Jurassic Park, Star Wars Prequels, Lord of the Rings
  - Toy 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

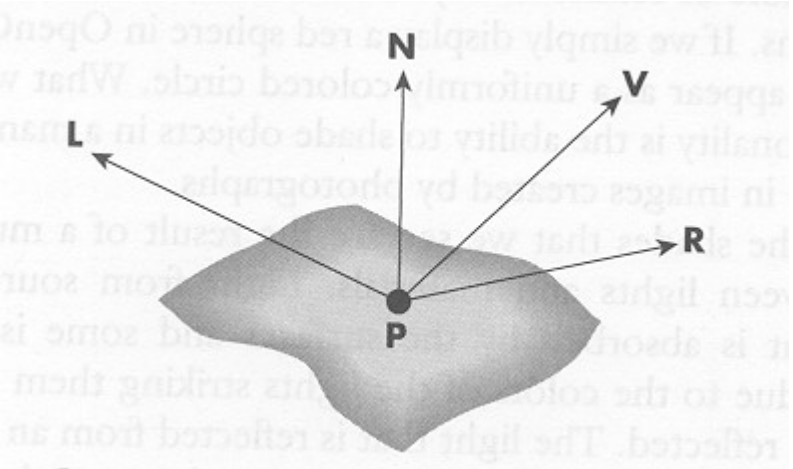
Brick Shader (Orange Book Ch 6)

- Uses scalar Phong shading for lighting
- Calculates brick/mortar based on model coordinates



Phong Shading

- L light source
- N normal vector for surface
- R reflected light
  - $R = 2(L \bullet N)N - L$
- V viewer (eye)
- Intensity $(V \bullet R)^S MC$
  - S shininess
  - M material reflection coefficient
  - C color if light source
- Calculated independently for R,G,B



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
  - Still somewhat a work in progress
  - Generally needs decent hardware
- So why use it?
  - Ultimate flexibility
  - Unsupported features (e.g. bump maps)

The Future of Shaders

- WebGL and OpenGL ES 2.0 **requires** shaders (no fixed pipeline support)
  - Desktop OpenGL maintains compatibility
- Significantly steeper learning curve
  - User must supply transformation matrices for modelview and projection matrix
  - User must explicitly perform lighting calculations in the shader
  - Third party libraries provide functionality previously part of the standard