COS426 Computer Graphics

Precept 7
Aleksey Boyko
April 8, 2011

Topic

Topics

- Getting started
- Initialization
- Drawing
- Transformations
  - Cameras
  - Animation
- Input
  - Keyboard
  - Mouse
- Textures
- Lights
- Programmable pipeline elements (shaders)

Open(GL) (S)hading (L)anguage

- Integration with OpenGL
  - Easy to adapt existing code to use custom shaders
- Run-time compilation
  - No platform-specific binaries
- High level C-like language
  - Hardware vendors are free to optimize
- Cross-platform open standard
- Support for modular programming

Problem

- Microsoft likes HLSL and Cg
  - Install latest vendor's drivers
    - http://www.intel.com/support/graphics/sb/cs-010479.htm
    - http://www.nvidia.com/Download/index.aspx?lang=en-us
    - http://support.amd.com/us/kbarticles/Pages/Catalyst-OpenGL-preview-driver.aspx
- It is a driver so we need a way to bind driver API to OpenGL API
  - We will use GLEW

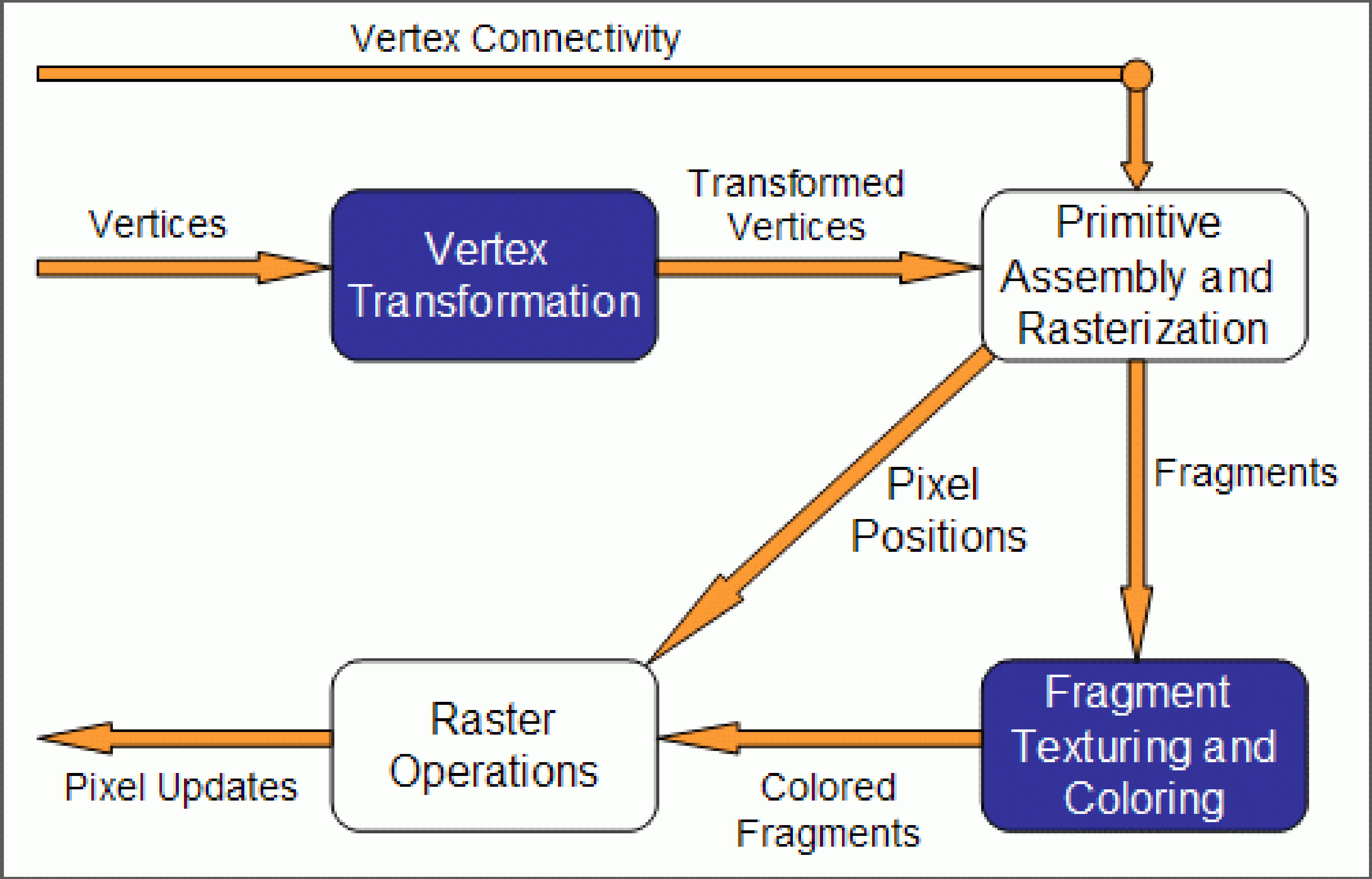
Open(**GL**) (**E**)xtension (**W**)rangler GLEW library

- Cross-platform open-source C/C++ extension loading library
- Efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform
- OpenGL functionality is exposed in a single header file
- Tested on Windows, Linux, Mac OS X, etc.

Installing GLEW

- Download from http://glew.sourceforge.net/
- Rebuild (optional but suggested)
  - *nix: make install
  - Windows: (glew)/build/
- Put in “standard” locations (make install does it)
  - Shared libraries (.dll)
  - Static libraries (.lib)
  - Headers (.h)

Now to shaders



From http://www.lighthouse3d.com/tutorials/glsl-tutorial/pipeline-overview/

Shader

- Replaces the stage implementation completely
  - Don’t have to do everything
  - Can’t rely on fixed functionality to do some part
- Vertex shader performs per-vertex operations
  - No access to other vertices
- Fragment shader performs per-pixel operations
  - No access to other pixels

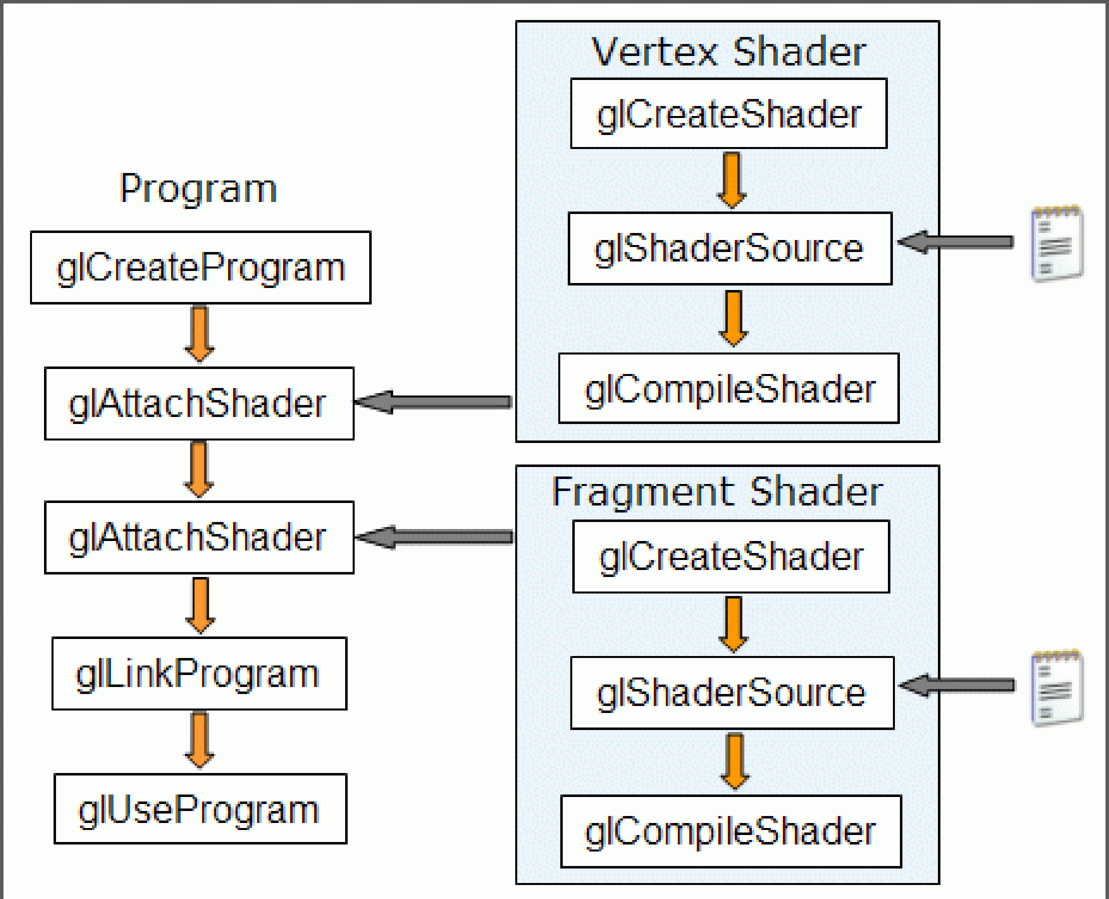
Vertex shader

- Does (per-vertex)
  - MODEL_VIEW and PROJECTION transformations
  - Normal transformation and normalization
  - Texture coordinate generation and transformation
  - Lighting
  - Color computation
- At least
  - Set gl_Position

Fragment shader

- Does (per-pixel)
    - Compute color and texture coordinates
    - Texture application
    - Fog computation
    - Computing normals
- At least
    - Set gl_FragColor
    - Or discard the fragment
- Cannot
    - Change pixel's coordinate

Creating a shader



Creating a new shader

- GLuint glCreateShader(
  - GLenum *shaderType*)
    - GL_VERTEX_SHADER
    - GL_FRAGMENT_SHADER
  - Returns shader’s handle
    - Just like in case of texture

Providing the source code

- void **glShaderSource**(
  - GLuint *shader*,
    - Shader handle
  - GLsizei *count*,
    - Number of elements in the string
  - const GLchar ***string*,
    - Array of pointers to the source code strings
  - const GLint **length*)
    - Length of strings
    - If NULL, strings are null-terminated

Compiling the shader

- void **glCompileShader**(
  - GLuint *shader*)
    - Shader handle

Shader program

- GLuint **glCreateProgram**(*void*)
  - Provides a handle for the program
- void **glAttachShader**(
  - GLuint *program*,
  - GLuint *shader*)
- void **glLinkProgram**(
  - GLuint *program*)
- void **glUseProgram**(
  - GLuint *program*)
    - If 0 – use fixed pipeline implementation

Cleaning-up

- Detach shader
  - *void **glDetachShader**(*
    - *GLuint program,*
    - *GLuint shader)*
- Delete shader
  - *void **glDeleteShader**(*
    - *GLuint shader)*
- Stop using program
  - ***glUseProgram**(0)*
- Delete program
  - *void **glDeleteProgram**(GLuint program)*

Communicate data to shader

- Uniform variables
- Attribute variables
- Varying variables
- Special output variables

Uniform variables

- Change infrequently
  - Cannot be changed between glBegin() and glEnd()
- Accessible by both shaders
- Read only in both shaders

Uniform variables

- Built-in
  - gl_ModelViewMatrix
  - gl_Fog
  - Etc.
  - Set with respective OpenGL calls

Uniform variables

- User-defined
  - Establish location with
    - GLint **glGetUniformLocation**(
      - GLuint *program*,
      - const GLchar **name*)
  - Set value with
    - GLint glUniform{1,2,3,4}f[v](
      - GLint location,
      - GLfloat v0, GLfloat v1, …
      - [GLsizei count, GLfloat *v]);
      - {>1}:[GLsizei count, GLboolean transpose, GLfloat *v)]

Uniform variables

- In shader code
  - uniform float specIntensity;
  - uniform vec4 specColor;
  - uniform float t[2];
  - uniform vec4 colors[3];

Attribute variables

- Can be different for each vertex
- Accessible by vertex shader only
- Read only by vertex shader

Attribute variables

- Built-in
  - gl_Color
  - gl_Normal
  - gl_Vertex
  - Etc..
  - Set with respective OpenGL calls

Attribute variables

- User-defined
  - Establish location with
    - GLint glGetAttribLocation(
      - GLuint *program*,
      - const GLchar **name*)
  - Set value with
    - GLint glVertexAttrib{1,2,3,4}f[v](
      - GLint location,
      - GLfloat v0, GLfloat v1, …
      - [GLfloat *v]);

Attribute variables

- In shader code
  - attribute float specIntensity;
  - attribute vec4 specColor;
  - attribute float t[2];
  - attribute vec4 colors[3];

Varying variables

- Written by vertex shader
- Interpolated for primitive
- Read only by fragment shader
- Built-in/user-defined/special

Varying variables

- Built-in
  - gl_FrontColor
  - gl_BackColor
  - gl_FogFragCoord
  - Etc..

Varying variables

- User-defined code
    - varying float specIntensity;
    - varying vec4 specColor;
    - varying float t[2];
    - varying vec4 colors[3];

Varying variables

- Special
    - gl_Position
    - gl_PointSize
    - gl_ClipVertex

Special output variables

- Passed to final stages
  - gl_FragColor
  - gl_FragDepth
  - gl_FragData[n]

GLSL

- A lot like C
  - Scope
  - Conditionals
  - Loops
  - Functions
    - Each type of shader has to have *void main()*
  - Structs
  - Comments
- Different
  - Access using .xyzw, .rgba, .stpq
  - Parameters in/out/inout
  - No type casting or promotion:
    - use explicit constructors!

Data types

- Simple
  - float
  - bool
  - int
- Vectors
  - vec{2,3,4}
  - bvec{2,3,4}
  - ivec{2,3,4}
- Matrices
  - mat{2,3,4}

Sample implementation

Code example

References

Code from this precept:

http://www.cs.princeton.edu/courses/archive/spr11/cos426/precepts/GLSLTutorial.zip

More tutorials (partly used in the presentation):

http://www.lighthouse3d.com/tutorials/glsl-tutorial/

http://nehe.gamedev.net/

OpenGL quick reference:

http://www.khronos.org/files/opengl4-quick-reference-card.pdf