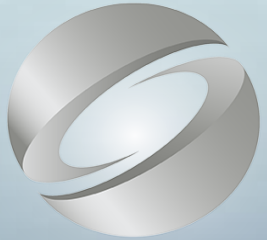
An Introduction to OpenGL Programming

Modified to be applicable to PyOpenGL

Ed Angel University of New Mexico

Dave Shreiner ARM, Inc.



SIGGRAPH 2013

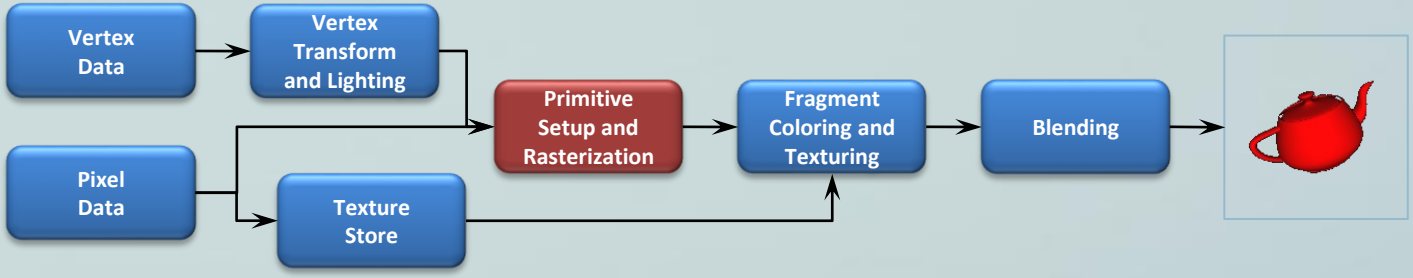
The 40th International Conference and Exhibition on Computer Graphics and Interactive Techniques

Evolution of the OpenGL Pipeline

SIGGRAPH 2013
The 40th International Conference and Exhibition
on Computer Graphics and Interactive Techniques

In the Beginning …

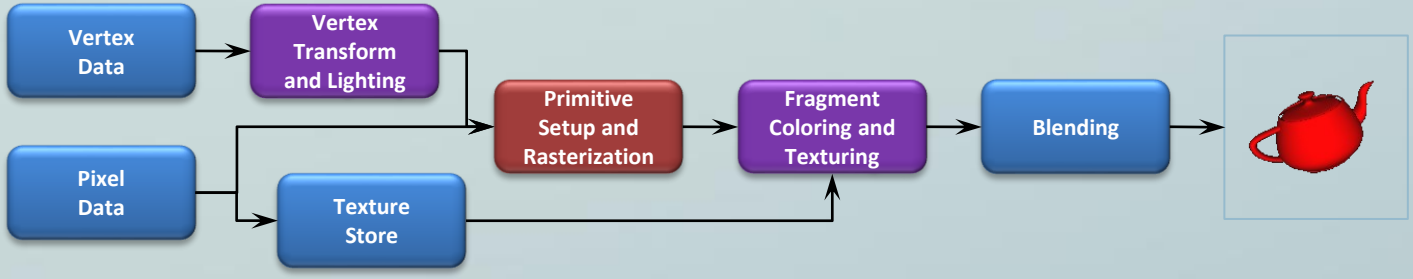
- OpenGL 1.0 was released on July 1st, 1994
- Its pipeline was entirely *fixed-function*
  - the only operations available were fixed by the implementation



- The pipeline evolved
  - but remained based on fixed-function operation through OpenGL versions 1.1 through 2.0 (Sept. 2004)

Beginnings of The Programmable Pipeline

- OpenGL 2.0 (officially) added programmable shaders
  - *vertex shading* augmented the fixed-function transform and lighting stage
  - *fragment shading* augmented the fragment coloring stage
- However, the fixed-function pipeline was still available



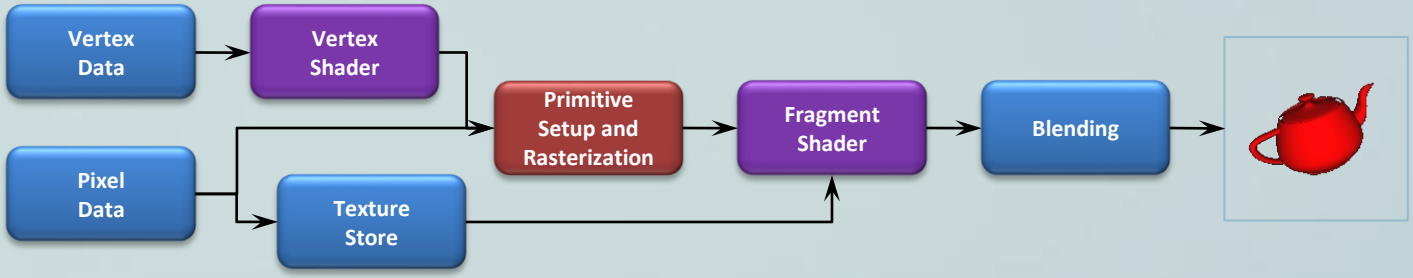
An Evolutionary Change

- OpenGL 3.0 introduced the *deprecation model*
  - the method used to remove features from OpenGL
  - e.g. glBegin(), glEnd()
- The pipeline remained the same until OpenGL 3.1 (released March 24th, 2009)
- Introduced a change in how OpenGL contexts are used

Context Type	Description
Full	Includes all features (including those marked deprecated) available in the current version of OpenGL
Forward Compatible	Includes all non-deprecated features (i.e., creates a context that would be similar to the next version of OpenGL)

The Exclusively Programmable Pipeline

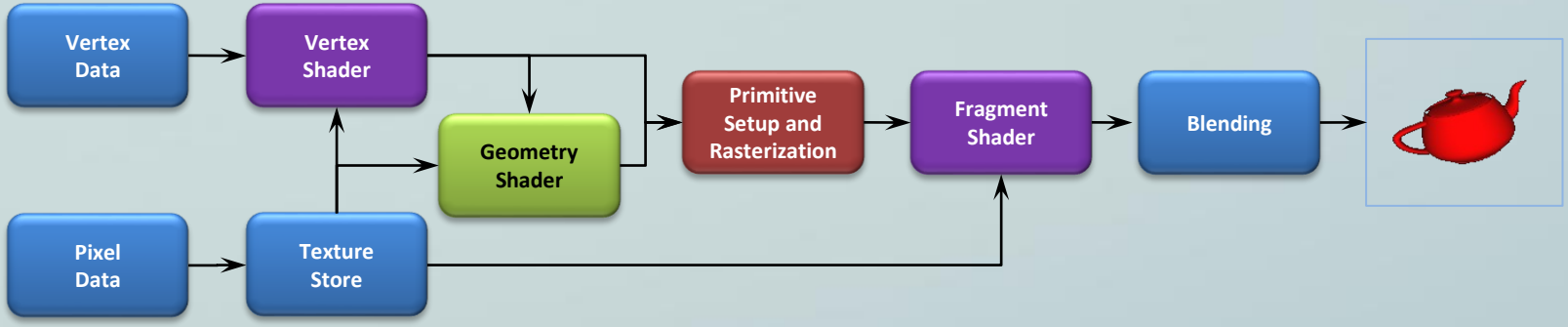
- OpenGL 3.1 removed the fixed-function pipeline
  - programs were required to use only shaders



- Additionally, almost all data is GPU-resident
  - all vertex data sent using buffer objects

More Programmability

- OpenGL 3.2 (released August 3rd, 2009) added an additional shading stage – geometry shaders
  - modify geometric primitives within the graphics pipeline



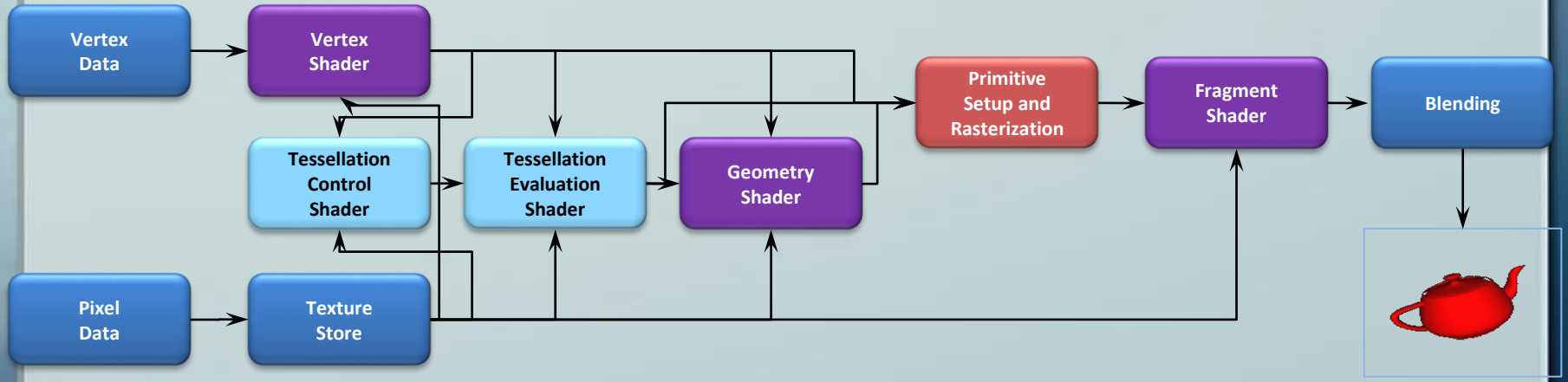
More Evolution – Context Profiles

- OpenGL 3.2 also introduced *context profiles*
  - profiles control which features are exposed
  - currently two types of profiles: *core* and *compatible*
- An implementation was only *required* to define core, so compatibility is not guaranteed to be available

Context Type	Profile	Description
Full	core	All features of the current release
	compatible	All features ever in OpenGL
Forward Compatible	core	All non-deprecated features (exclude all future deprecated features)
	compatible	Not supported

The Latest Pipelines

- OpenGL 4.1 (released July 25th, 2010) included additional shading stages – *tessellation-control* and *tessellation-evaluation* shaders
- Latest version is 4.6 (as of July 31, 2017)

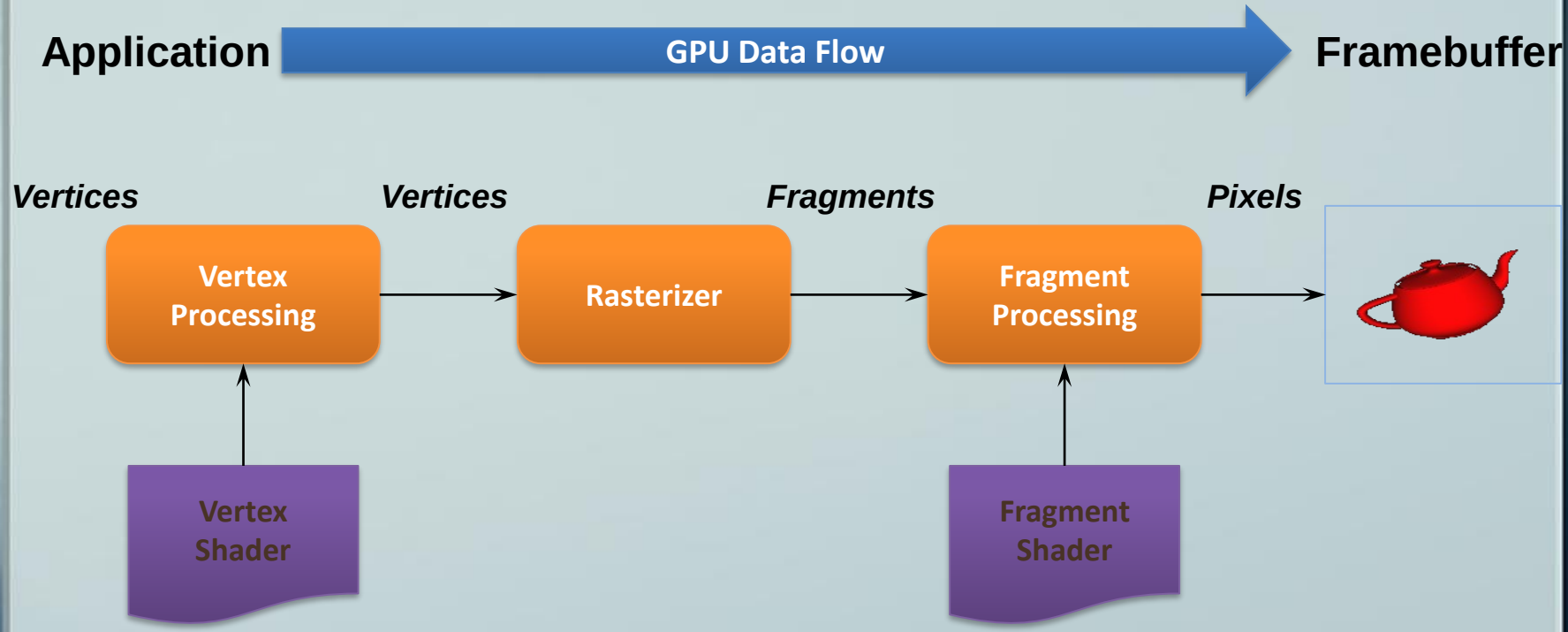


OpenGL Application Development

SIGGRAPH 2013

The 40th International Conference and Exhibition on Computer Graphics and Interactive Techniques

A Simplified Pipeline Model

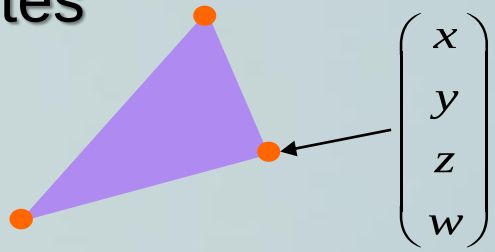


OpenGL Programming in a Nutshell

- Modern OpenGL programs essentially do the following steps:
  - Create shader programs
  - Create buffer objects and load data into them
  - “Connect” data locations with shader variables
  - Render

Representing Geometric Objects

- Geometric objects are represented using *vertices*
- A vertex is a collection of generic attributes
  - positional coordinates
  - colors
  - texture coordinates
  - any other data associated with that point in space



- Position stored in 4 dimensional homogeneous coordinates
- Vertex data must be stored in vertex buffer objects (VBOs)
- VBOs must be stored in vertex array objects (VAOs)

Shaders and GLSL

SIGGRAPH 2013

The 40th International Conference and Exhibition on Computer Graphics and Interactive Techniques

GLSL Data Types

- Scalar types: `float, int, bool`
- Vector types: `vec2, vec3, vec4`
  `ivec2, ivec3, ivec4`
  `bvec2, bvec3, bvec4`
- Matrix types: `mat2, mat3, mat4`
- Texture sampling: `sampler1D, sampler2D, sampler3D, samplerCube`
- C++ Style Constructors

```
vec3 a = vec3(1.0, 2.0, 3.0);
```

Operators

- Standard C/C++ comment styles
- Standard C/C++ arithmetic and logic operators
- Overloaded operators for matrix and vector operations

```
mat4 m;
vec4 a, b, c;

b = m*a;
c = a*m;
```

Components and Swizzling

- Access vector components using either:
  - [ ] (c-style array indexing)
  - `xyzw`, `rgba` or `stpq` (named components)

- For example:

```
vec3 v;
```

`v[1], v.y, v.g, v.t` - all refer to the same element

- Component swizzling:

```
vec3 a, b;
a.xy = b.yx;
```

Qualifiers

- Used with global variables
- in, out (varying, attribute used in version <= 1.2)
  - Copy vertex attributes and other variable into and out of shaders

```
in  vec2 texCoord;
out vec4 color;
```

- uniform
  - shader-constant variable from application

```
uniform float time;
uniform vec4 rotation;
```

Functions

- Built in
  - Arithmetic: `sqrt`, `pow`, `abs`
  - Trigonometric: `sin`, `asin`
  - Graphical: `length`, `reflect`
- User defined

Built-in Variables

- `gl_Position`
  - (required) output position from vertex shader
- `gl_FragCoord`
  - input fragment position in window space, and depth
- `gl_FragDepth`
  - output variable to explicitly set the depth value for the fragment
- `gl_FragColor`
  - (not required) output color given to fragment in fragment shader

A Simple Vertex Shader

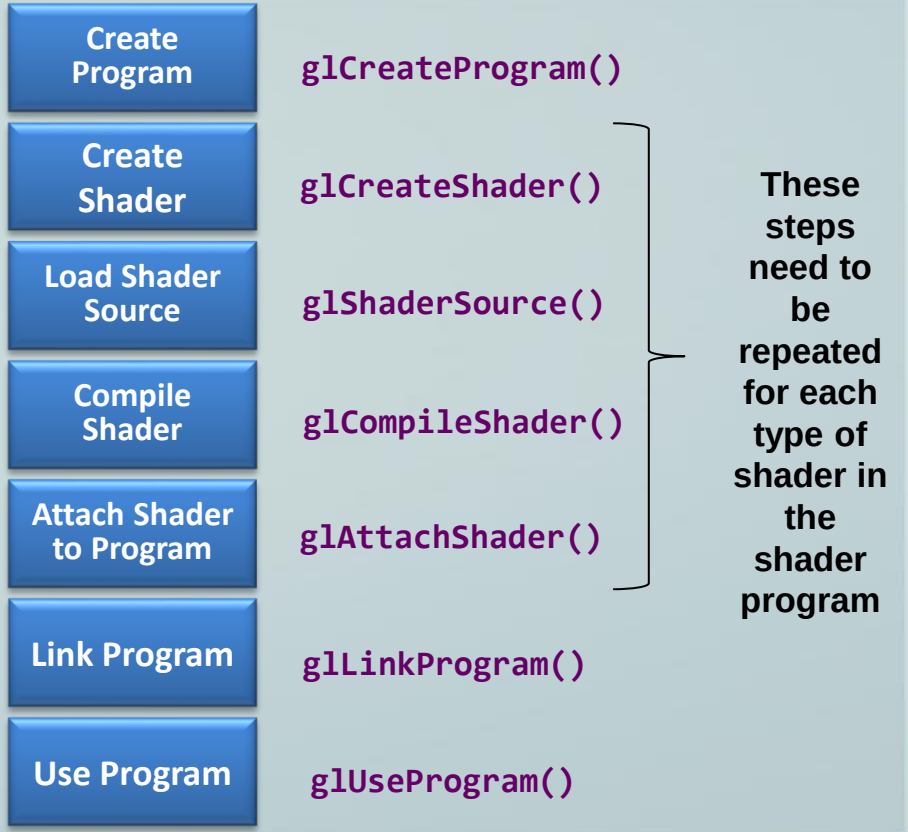
```
#version 430

in vec4 vPosition;
in vec4 vColor;

out vec4 color;

void main()    // entry point for the shader, in contrast
{              // in standard C, entry point for the program
    color = vColor;
    gl_Position = vPosition;
}
```

A Simple Fragment Shader

```
#version 430

in vec4 color;

out vec4 fColor; // fragment's final color

void main()
{
    fColor = color;
}
```

Getting Your Shaders into OpenGL

- Shaders need to be compiled and linked to form an executable shader program
- OpenGL provides the compiler and linker
- A program must contain
  - vertex and fragment shaders
  - other shaders are optional



A Simpler Way

- We will create a routine to compile and link our shaders

```
compileShaders(vertex_code, fragment_code)
```

- `compileShaders` takes two string
  - `vertex_code` is a source code string of vertex shader
  - `fragment_code` is a source code string of fragment shader
- Fails if shaders don't compile, or program doesn't link
- Otherwise, `compileShaders` returns program object id

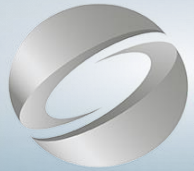
Vertex Shaders

- A vertex shader is initiated by each vertex input, e.g. by `glDrawArrays()`
- The vertex shader runs once for each vertex provided by the vertex array or other vertex input methods
- The vertex shader should output a position in clip coordinates to the rasterizer
- Basic uses of vertex shaders
  - Transformations
  - Lighting
  - Moving vertex positions

Fragment Shaders

- A shader that is executed for each “potential” pixel
  - fragments still need to pass several tests before making it to the framebuffer
- There are lots of effects we can do in fragment shaders
  - Per-fragment lighting
  - Texture and bump Mapping
  - Environment (Reflection) Maps

Resources



SIGGRAPH 2013
The 40th International Conference and Exhibition
on Computer Graphics and Interactive Techniques

Books

- Modern discussion
  - The OpenGL Programming Guide, 8th Edition
  - Interactive Computer Graphics: A Top-down Approach using OpenGL, 6th Edition
  - The OpenGL Superbible, 5th Edition
- Older resources
  - The OpenGL Shading Language Guide, 3rd Edition
  - OpenGL and the X Window System
  - OpenGL Programming for Mac OS X
  - OpenGL ES 2.0 Programming Guide
- Not quite yet …
  - WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL

Online Resources

- The OpenGL Website: www.opengl.org
  - API specifications
  - Reference pages and developer resources
  - Downloadable OpenGL (and other APIs) reference cards
  - Discussion forums
- The Khronos Website: www.khronos.org
  - Overview of all Khronos APIs
  - Numerous presentations

Thanks!

- Feel free to drop us any questions:

angel@cs.unm.edu
shreiner@siggraph.org

- Course notes and programs available at

www.daveshreiner.com/SIGGRAPH
www.cs.unm.edu/~angel