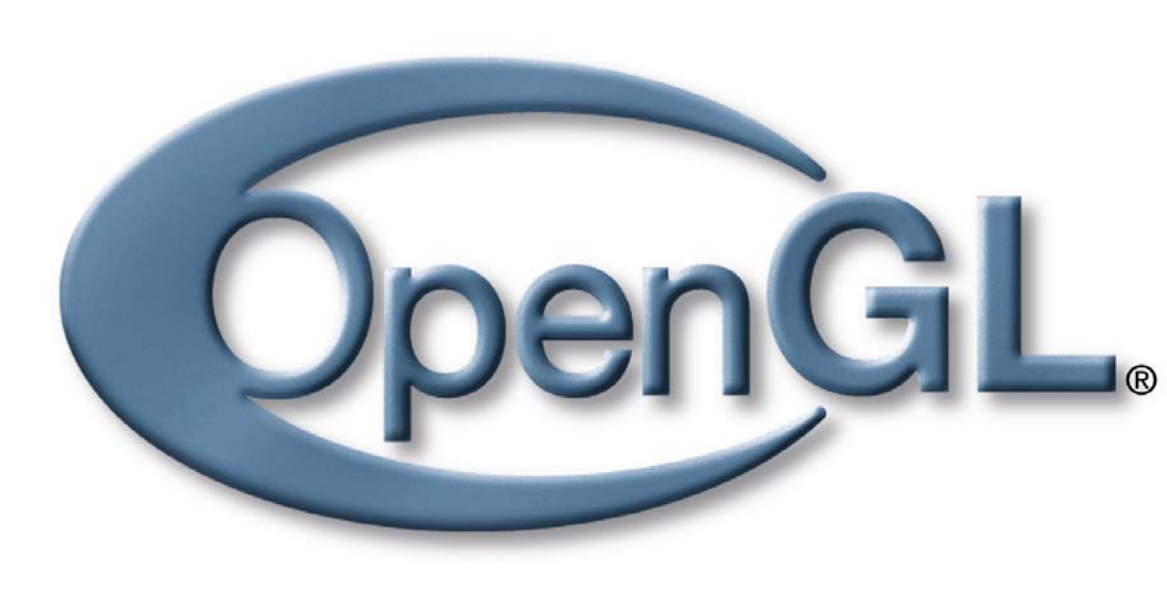


OpenGL Longs Peak

Michael Gold
NVIDIA Corporation

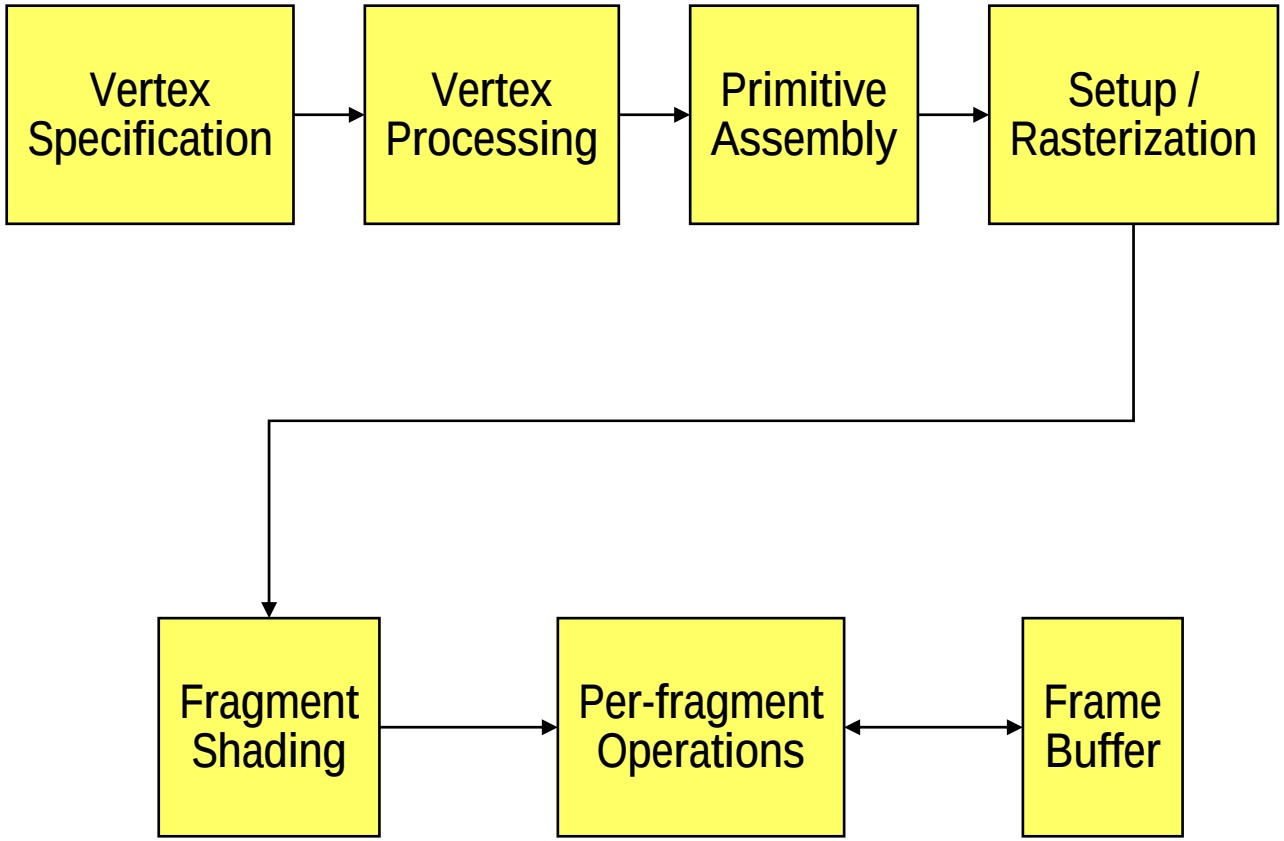
New Interface

- Hardware has evolved, OpenGL has not
  - DMA vs direct writes
  - Programmable vs fixed function
  - Multitexture!
- API burdened by extension without design
  - Too many cooks spoil the broth
  - Too many comparable means to the same end
  - Requires expertise to achieve efficiency
- Lessons learned
  - Fifteen years experience to draw upon

Goals

- Simplify the task for developers
- Simplify the task for implementors
- Design overhead and uncertainty out of the spec
- No “important” functional regressions
- No false promises
  - Some APIs were rarely/never accelerated

OpenGL Pipeline block diagram



GL2 vs. LP: Vertex specification

- Legacy GL options
  - Immediate Mode (Begin/End)
  - Display Lists
  - Vertex Arrays
  - Vertex Buffer Objects
- Problems
  - Difficult to optimize all paths
  - Flexibility led to inefficiency
  - Unclear to developers which is “fast path”
  - VBO layered on Vertex Arrays is inefficient and error-prone
    - Hidden performance cost when mixing VBO and non-VBO arrays
    - State changes are expensive
- Longs Peak solutions
  - Vertex array objects w/ Buffer objects
    - No client arrays
    - Rapid state changes
  - Geometry-only display lists (tentative)

GL2 vs. LP: Vertex processing

- Legacy GL options
  - Fixed function
  - Vertex program
- Problems
  - No hardware support for fixed function
    - Driver generates vertex program under the covers
  - No hardware support for matrix operations
    - Driver emulates in software
  - Fixed function is limiting
    - Developers fit code to the interface
  - Uniform updates expensive
    - Data must be copied into program object
- Longs Peak solution
  - Eliminate fixed function
    - Can be layered
    - Encourage developer to tailor programs to specific needs
  - Uniform partitions
    - Rapid state changes

GL2 vs. LP: Fragment shading

- Legacy GL options
  - Texenv
  - Texenv_combine
  - Texenv_crossbar
  - Extensions! (register_combiners, texture_shader, etc.)
  - Fragment programs
- Problems
  - No hardware support (driver generated shaders)
  - Limited utility
  - Inefficient state changes
- Longs Peak solutions
  - Eliminate fixed function
    - Can be layered
  - Program object is a container
    - Efficient state changes

GL2 vs. LP: Framebuffer objects

- Legacy GL options
  - Framebuffer objects, v1
- Problems
  - Texture consistency
  - Framebuffer completeness
  - Validation overhead
- Longs Peak solution
  - Framebuffer objects, v2
  - Image objects
  - Format objects

Object Model basics

- Immutable
  - All structure properties fixed at object creation
  - Only data and attachments may be modified
  - Strong usage hints
- GL allocates handles
  - Efficient lookup
  - Hybrid push model
- Direct editing
  - Avoids polluting state cache
  - Eliminate mysterious side effects
- Efficient state grouping
  - Minimizes API required to change state
- Efficient sharing
  - Per object state
  - Avoids dangerous race conditions

State objects

- Fully immutable
- May be shared
- Types
  - Format
  - Texture Filter
  - Shader

Data Objects

- Immutable structure
- Mutable data
- May be shared
- Types
  - Image (texture, renderbuffer)
  - Buffer (VBO, PBO, Uniform block)
  - Sync objects
  - Query objects

Container objects

- Mutable attachments
- Immutable attachment properties
- May not be shared
- Types
  - Program
  - Vertex Array
  - Framebuffer

Templates

- Client state
- Mutable
  - May be reused for multiple objects
- Extensible
  - Opaque data structure
- Watch for memory leaks

```
// Create an attribute object, initialized to defaults
GLtemplate temp = glCreateAttrib(WIDGET_OBJECT);
// Override defaults as necessary
glTemplateAttribt_i(temp, PROPERTY, value);
…
// Create the actual object
Object = glCreateWidget(temp);
// Destroy attribute object
glDestroyTemplate(temp);
```

Image Objects

- Replace textures and render buffers
- Size, shape, format fixed at creation time
  - Data is mutable
- Always “complete”

```
GLtemplate temp = glCreateAttrib(GL_IMAGE_OBJECT);
glTemplateAttribt_o(temp, GL_FORMAT, format);
glTemplateAttribt_i(temp, GL_WIDTH,  width);
glTemplateAttribt_i(temp, GL_HEIGHT, height);
glTemplateAttribt_i(temp, GL_LEVELS, levels);
glTemplateAttribt_i(temp, GL_RENDER, GL_TRUE);
GLbuffer image = glCreateImage(temp);
glDestroyTemplate(temp);

 - or -

image = gluCreateTexture2D(format, width, height, levels);
```

Texture Filter Objects

- Replaces TexParameter
- Separate from Image Object
  - Samplers may be bound to multiple image objects
  - Images may be bound to multiple sampler objects

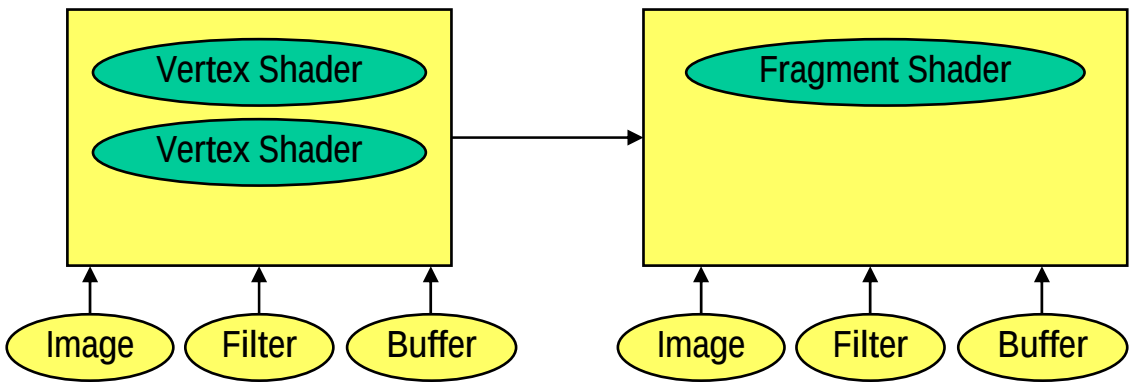
```
GLtemplate temp =
    glCreateTemplate(GL_TEXTURE_FILTER_OBJECT);
glTemplateAttribt_i(temp, GL_MIN_FILTER, min_filter);
glTemplateAttribt_i(temp, GL_MAG_FILTER, mag_filter);
glTemplateAttribt_i(temp, GL_WRAP_S, wrap_s);
glTemplateAttribt_i(temp, GL_WRAP_T, wrap_t);
GLtextureFilter filter = glCreateTextureFilter(temp);
glDestroyTemplate(temp);

 - or -

filter = gluCreateTextureFilter2D(min_filter, mag_filter,
                                  wrap_s, wrap_t);
```

Buffer objects

- Size, usage fixed at creation
- Store vertex and pixel data, uniform values
- Unmappable by default
  - Setting this attribute may affect performance
- Similar to current VBO and PBO

Program Objects

- Shader object is compiled representation of text string
  - Trivial “immutable” state object
  - Shader may represent complete or partial stage
- Program links together one or more shaders
  - No incremental relinking (create a new program instead)
- Program may include one or more stages
  - Complete set of programs bound atomically
- Program is a container object
  - Attach images, texture filters, uniform buffers directly to program



Uniform Blocks

- Uniform storage in buffer objects
- Efficient editing
  - BufferData, MapBuffer
- Atomically swap entire set of uniform values
  - Toggle between multiple state vectors without a copy
- Common blocks
  - Shader text defines partitions
  - May be shared between shaders

Vertex Array objects

- Encapsulate complete set of VBOs
- Attachments are mutable
  - Buffer object, offset
- Set of attachments is immutable
- Attachment properties are immutable
  - Type, stride, etc
- No more client arrays!
- No more client enables!
- Rapid state changes
  - One API call instead of forty-eight

Framebuffer objects

- Format compatibility determined at FBO creation
- Attachment compatibility strictly enforced
- Reduced validation overhead
- New features
  - Mixed size attachments
  - 1 – 2 component rendering

Other object types

- Format
- Sync
- Query
- Display lists (?)

Backward Compatibility

- Goals
  - Complete backward compatibility
  - Enable incremental migration
- Proposal
  - Longs Peak requires a new context
  - Both legacy and Longs Peak contexts may bind to same drawable
  - Image objects may bind to texture and renderbuffer objects
- Benefits
  - Longs Peak not burdened by legacy requirements
  - Legacy code and LP code may peacefully co-exist
  - Application developers may move at their own pace
    - Incentive: GL2 unlikely to evolve

Questions?