A New Object Model for OpenGL

Michael Gold
NVIDIA Corporation
Why do we need a new object model?

• Existing object model evolved, was never designed
  - Began with display lists
  - Texture objects started down wrong path
  - New semantics with each extension

• Optimized for object creation
  - Most apps create once, reference often
  - Name / pointer translation typically 3-5% of runtime

• Bind to edit / bind to use
  - Complex “side effects” cause application errors

• Object mutability too flexible
  - Burden on implementation to support
  - Application misuse / thrashing

• Sharing is problematic
  - “All or nothing” lacks useful granularity
  - Dangerous race conditions
  - Ill-defined orphaning behavior
New object model basics

• Object ID generated by the implementation
  - No more “push” model
  - Immediate feedback on success of creation
  - Eliminates ambiguous “orphaning” behavior

• New data types
  - Strong type checking
  - May be pointer (faster lookup, faster error detection!)

• Manipulator functions take an object ID
  - No more “bind to edit”
  - Immutability removes most editing anyway
  - More intuitive / fewer side effects

• Per-object sharing
  - No more “all or nothing”
  - Objects are not shared by default
Immutability

- Object structure fixed at creation time
  - Create new instead of edit existing
  - Contents may be modified
- Atomic creation removes “guesswork”
- Read-only structure eliminates dangerous race conditions
- Easier to program
  - Fewer points of failure
- Easier to optimize
  - No driver heuristics necessary
  - What’s good for us is good for you!
Attribute Objects

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

// Create an attribute object, initialized to defaults
GLattribute attrib = glCreateAttrib(WIDGET_OBJECT);
// Override defaults as necessary
glSetAttribi(attrib, PROPERTY, value);
...
// Create the actual object
Object = glCreateWidget(attrib);
// Destroy attribute object
glDestroyObject(attrib);
Image Objects

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

```c
GLattribute attrib = glCreateAttrib(GL_IMAGE_OBJECT);
glSetAttribo(attrib, GL_FORMAT, format);
glSetAttribi(attrib, GL_WIDTH, width);
glSetAttribi(attrib, GL_HEIGHT, height);
glSetAttribi(attrib, GL_LEVELS, levels);
GLimage image = glCreateImage(attrib);
glDestroyObject(attrib);

- or -

image = gluCreateImage2D(format, width, height, levels);
```
Sampler Objects

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

```
GLattribute attrib = glCreateAttrib(GL_SAMPLER_OBJECT);
glSetAttribo(attrib, GL_MIN_FILTER, min_filter);
glSetAttribi(attrib, GL_MAG_FILTER, mag_filter);
glSetAttribi(attrib, GL_WRAP_S, wrap_s);
glSetAttribi(attrib, GL_WRAP_T, wrap_t);
GLsampler sampler = glCreateSampler(attrib);
gIDestroyObject(attrib);

or

sampler = gluCreateSampler2D(min_filter, mag_filter,
                           wrap_s, wrap_t);
```
Program Objects

• Shader object is compiled representation of text string
  - Trivial “immutable” object

  // shaderType is VERTEX or FRAGMENT; source is text string
  shader = gluCreateShader(shaderType, source);

• Program links together one or more shader(s)
  - No incremental relinking (create a new program instead)

  glSetAttribov(attrib, GL_SHADERS, count, shader_list);

• Uniforms may be declared “const” prior to linking
  - Hint may allow greater optimization
  - Comparable to #define
  - Optimize “uber shaders”

  glSetNamedAttribf(attrib, GL_UNIFORM_CONSTANT, “decay”, 0.2f);
Uniform objects

- Uniform storage separate from program object
- Uniform values are data (mutable)
- Enables rapid updates of uniform values
  - Toggle between multiple state vectors
- Create “default” uniform object from linked program
  - Contains exact set of uniforms required for the program
- Create custom uniform groups prior to linking
  - Uniform objects sharable between programs
- Image and Sampler objects bind to uniform objects
  - Solves the 32 texture limit
  - No more ActiveTexture / BindTexture
Other object types

• Format objects
  - More expressive than simple enumerants

• Framebuffer objects
  - Includes “format compatibility” query mechanism
  - Always complete!

• Buffer objects
  - Now, with 50% more immutability!
  - Usage no longer a hint

• Vertex Array objects
  - Encapsulates all array state for fast state changes
  - Immutability removes validation overhead

• Synchronization objects
  - Asynchronous description

• Display lists?
  - Stay tuned…