OpenGL 2.0 and New Extensions

Cass Everitt

NVIDIA Corporation

Overview

- OpenGL 2.0 additions
- New ARB extensions
- Applications

Brief History of OpenGL

1983	IRIS GL ships with SGI IRIS 1000 terminal
1987	SGI and Pixar consider joint API development
1991	OpenGL ARB created
1992	OpenGL 1.0 completed (June 30)
1995	OpenGL 1.1 released (vertex array, texture objects, new texenv modes)
1997	Fahrenheit agreement between SGI and Microsoft
1998	OpenGL 1.2 released (3D textures, separate specular, imaging)
1999	OpenGL 1.2.1 released (multi-texture)
2001	OpenGL 1.3 released (compressed texture, cube maps, multi-sample, dot3)
2002	OpenGL 1.4 (mip-map generation, shadows, point parameters)
2003	OpenGL 1.5 (vertex buffer objects, occlusion query) ARB extensions: OpenGL Shading language, ARB_vertex_program, ARB_fragment_program
2004	OpenGL 2.0

Life of an Extension

- GL_NVX_foo – eXperimental
- GL_NV_foo – vendor specific
- GL_EXT_foo – multi-vendor
- GL_ARB_foo
- Core OpenGL

OpenGL 2.0

- OpenGL Shading Language
- Multiple Draw Buffers
- Non-Power-Of-Two Textures
- Point Sprites
- Separate Stencil
- Separate Blend Functions and Equations
- Other minor changes

OpenGL Shading Language

- **Shader objects**
  - mechanism to manage shader and program objects
  - from ARB_shader_objects extension
- **Shader programs**
  - ARB_vertex_shader / ARB_fragment_shader
- **OpenGL Shading Language**
  - ARB_shading_language_100
  - OpenGL 2.0 must support 1.10 of GLSL
- **Minor changes to shader APIs**
  - use GLuint instead of handleARB type

Multiple Draw Buffers

- Allows writing multiple colors to different color buffers at once
  - Reduces number of rendering passes
  - Based on ATI_draw_buffers extension
- In the C code:

```
GLenum buffers[] = { GL_AUX0, GL_AUX1 };
glDrawBuffers( 2, buffers );
```

- In the shader:

```
OP result.color[0], src0, src1, …;
OP result.color[1], src0, src1, …;
```

Non Power-of-Two Textures

- Relaxes the requirement that texture dimensions must be power-of-two
  - Useful for video, image processing
- Promoted from ARB_texture_non_power_of_two
- Mip level dimensions use “floor” convention
  - $w_i = \mathtt{floor}(w_0 / 2^i)$

Point Sprites

- **Points with varying texture coordinates**
  - **Useful for particle systems etc.**
- **Promoted from ARB_point_sprite extension**
- **Can change orientation using POINT_SPRITE_COORD_ORIGIN**
  - **UPPER LEFT (default) or LOWER LEFT**
  - **most hardware supports upper-left origin**
  - **import from Direct3D**

Two-Sided Stencil

- **Provides separate stencil functions and operations depending on facing**
- **Unified form of the functionality in**
  - **EXT_stencil_two_side / ATI_separate_stencil**
- `void` **`StencilFuncSeparate`**`( enum` *`face`*`, enum` *`func`*`, int` *`ref`*`, uint` *`mask`* `);`
- `void` **`StencilOpSeparate`**`( enum` *`face`*`, enum` *`sfail`*`, enum` *`dpfail`*`, enum` *`dppass`* `);`
- **Helpful for single-pass stenciled shadow volume rendering.**
  - **front faces incr, back faces decr, happens simultaneously**

Separate Blend Functions and Equations

- **More flexible blending**
- **Promoted from EXT_blend_func_separate and EXT_blend_equation_separate**
- `void` **`BlendFuncSeparate`**`( enum` *`srcRGB`*`, enum` *`dstRGB`*`, enum` *`srcAlpha`*`, enum` *`dstAlpha`* `);`
  - **allows separate srcFactor and dstFactor for RGB and A**
- `void` **`BlendEquationSeparate`**`( enum` *`modeRGB`*`, enum` *`modeAlpha`* `);`
  - **allows separate blend equations for RGB and A**

Minor changes

- Several minor revisions and corrections to the OpenGL 1.5 specification were made:
- In section 2.7, SecondaryColor3 was changed to set A to 1.0 (previously 0.0), so the initial GL state can be restored.
- In section 2.13, transformation was added to the list of steps not performed byWindowPos.
- Section 3.8.1 was clarified to mandate that selection of texture internal format must allocate a non-zero number of bits for all components named by the internal format, and zero bits for all other components.
- Tables 3.22 and 3.23 were generalized to multiple textures by replacing Cf with Cp.
- In section 6.1.9, GetHistogram was clarified to note that the Final Conversion pixel storage mode is not applied when storing histogram counts.
- The FOG COORD ARRAY BUFFER BINDING enumerant alias was added to table H.1.

Stuff That Didn't Make OpenGL 2.0

- **ARB_vertex_program & ARB_fragment_program**
  - **general lack of interest in making the ASM interfaces core**

Pixel Buffer Object (PBO)

- Now ARB_pixel_buffer_object
- VBO “buffer objects” are just arrays of bytes managed by the driver
- PBO uses same API, but has binding points for
  - `PIXEL_PACK_BUFFER`
    - glReadPixels(), glGetTexImage(), etc.
  - `PIXEL_UNPACK_BUFFER`
    - glTexImage*(), glDrawPixels(), etc.
- Allows more efficient texture downloads, and asynchronous read-back

Floating Point Pipeline

- Allow general-purpose rendering to and texturing from fp buffers and textures
- Incorporates several extensions:
  - ARB_color_buffer_float
  - ARB_texture_float
  - ARB_half_float_pixel

ARB_color_buffer_float

- Adds pixel formats / visuals with floating point RGBA color components
- Can also enable or disable [0, 1] clamping for various operations:
  - vertex colors (after lighting)
  - fragment color
  - pixel reads

  - `ClampColorARB(enum target, enum clamp)`
    - `CLAMP_VERTEX_COLOR_ARB`
    - `CLAMP_FRAGMENT_COLOR_ARB`
    - `CLAMP_READ_COLOR_ARB`

ARB_texture_float

- Floating point internal formats
  - RGBA32F
  - RGB32F
  - ALPHA32F
  - INTENSITY32F
  - LUMINANCE32F
  - LUMINANCE_ALPHA32F
  - RGBA16F
  - RGB16F
  - ALPHA16F
  - INTENSITY16F
  - LUMINANCE16F
  - LUMINANCE_ALPHA16F
- Also supports queries to determine component type

ARB_half_float_pixel

- "external format" for fp16 pixel data from the CPU
- s1e5m10
  - 1 sign bit
  - 5 exponent bits (bias of 15)
  - 10 mantissa bits
  - special numbers undefined
  - denorms defined

ARB_texture_rectangle

- No power-of-two constraints
- But, no mip-mapping, no repeat, no borders
- Coordinates are not normalized
  - (0..w, 0..h)
- More restricted functionality, but wider hardware support than ARB_texture_non_power_two

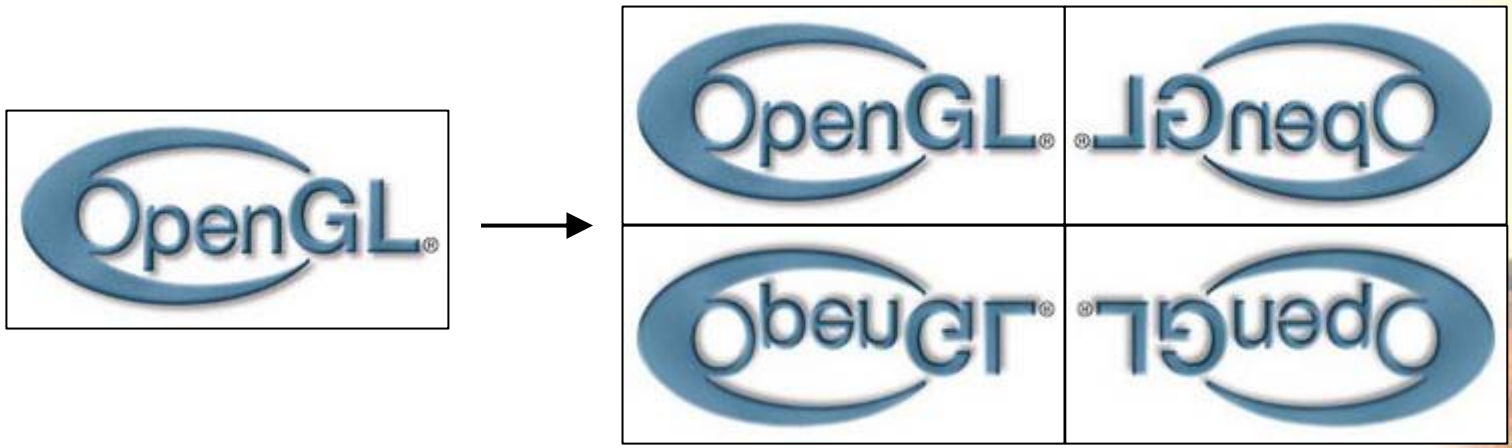
ARB_fragment_program_shadow

- Adds new texture targets for doing shadow map comparisons in fragment programs:
  - SHADOW1D, SHADOW2D, SHADOWRECT
- Textures must have depth format and have depth comparison mode set

```
!!ARBfp1.0 OPTION
ARB_fragment_program_shadow;
TEMP Result;
TEX Result, fragment.texcoord[0], texture[0], SHADOW2D;
END
```

ARB_texture_mirrored_repeat

- Introduces new wrap mode that effectively creates a texture twice the size which is mirrored around the center in each axis



Parting Queries

- **Where does the ARB need to focus efforts?**
  - **demos, whitepapers?**
  - **more rapid spec revisions?**
  - **developer conference?**
- **How do ISVs provide feedback?**
  - **Is something less formal than the participant undertaking desirable?**

Questions?

- cass@nvidia.com