OpenGL 4 Update

Barthold Lichtenbelt
OpenGL ARB WG Chair
Agenda

• **Open Standards Bringing 3D Everywhere**
  - Jon Peddie, Jon Peddie Research

• **OpenGL 4 Update**
  - Barthold Lichtenbelt, NVIDIA

• **GLSL 4 tips and tricks**
  - Bill Licea-Kane, AMD

• **OpenGL Ecosystem update**
  - Jon Leech, Khronos

• **Graphics benchmarking goes to 11**
  - Ian Williams, NVIDIA

• **Mixed OpenGL and OpenCL debugging and profiling using gDEBugger**
  - Yaki Tebeka, Graphic Remedy
Prizes

- 5 SuperBible 5th Edition
  - Pearson (Addison-Wesley)
- 1 ATI FirePro V5800 workstation GPU
  - AMD
- 1 Quadro Graphics 5000 Card
  - NVIDIA
- 3 gDEBugger GL one-year license
  - Graphic Remedy
- Reference card for everyone
  - Khronos
Our corporate sponsors
Our special sponsor: Rob Barris
OpenGL Strategy

- Serve developers
- Enable platforms
- Expose hardware
- Encourage innovation
OpenGL Tactics

- Align with OpenGL ES
- Features driven roadmap
- Adopt proven extensions
- Schedule driven roadmap
- Streamline the API
- Core and Compatibility profiles
- Embrace OpenCL interop
- Simplify DX porting
OpenGL Ecosystem

OpenGL ES 2.0 on desktop as subset of OpenGL 4.1 for content mobility

Only OpenGL provides graphics functionality on all platforms

High-end functionality developed first on desktop

• Different platforms needs -> API overlap not convergence
• Cooperation at Khronos -> consistency and synergies

WebGL drives need for security and pervasive OpenGL ES 2.0
Visual Computing Ecosystem

Mobile Visual Computing
Compute, graphics and AV APIs interoperate through EGL

Desktop Visual Computing
OpenGL and OpenCL have direct interoperability

OpenCL Specification includes embedded profile to enable sooner deployment on mobile devices
OpenGL for Each Hardware Generation

- Fixed Function
- Vertex and Fragment Shaders
- Geometry Shaders
- Tessellation and Compute

- OpenGL 4.0
- OpenGL 3.0
- OpenGL 2.0
- OpenGL 1.0
Unigine Heaven Benchmark

- OpenGL
- Tessellation
Accelerating OpenGL Innovation

- OpenGL 4.1 and GLSL 4.10 specifications available!
  - Support for the latest generation of programmable hardware
  - Superset of DX11 functionality

- OpenGL increased pace of innovation
  - Six new spec versions in two years
  - Actual implementations following specifications closely

© Copyright Khronos Group, 2010 - Page 14
What is in OpenGL 4.1?

- **ARB_get_program_binary**
  - Query and load a binary blob for program objects
- **ARB_separate_shader_objects**
  - Ability to bind programs individually to programmable stages
- **ARB_ES2_compatibility**
  - Pulling missing functionality from OpenGL ES 2.0 into OpenGL
- **ARB_shader_precision**
  - Documents precision requirements for several FP operations
- **ARB_vertex_attrib_64 bit**
  - Provides 64-bit floating-point component vertex shader inputs
- **ARB_viewport_array**
  - Multiple viewports for the same rendering surface, or one per surface
- **All functionality also available as ARB extensions**
New ARB only Extensions

• **ARB_robustness**
  - Robust features to grant greater stability when using untrusted code

• **Create_context_robustness (WGL and GLX)**
  - Create a context with robust features enabled

• **ARB_debug_output**
  - Callback mechanism to receive errors and warning messages from the GL

• **ARB_shader_stencil_export**
  - Set stencil reference value in fragment shader

• **ARB_cl_event**
  - Create OpenGL sync objects linked to OpenCL event objects
OpenGL 4 – DX11 Superset

• Interop with a complete compute solution
  - OpenGL is for graphics – OpenCL is for compute

• Get_program_binary
  - Ability to query a binary, and save it for reuse later

• Flow of content between desktop and mobile
  - All of OpenGL ES 2.0 capabilities available on desktop
  - EGL on Desktop in the works
  - WebGL bridging desktop and mobile

• Cross platform
  - Mac, Windows, Linux, Android, Solaris, FreeBSD
  - Result of being an open standard
Specifications Available Now

- Three new OpenGL 4 specifications:
  - OpenGL 4.1 with core profile
  - OpenGL 4.1 with compatibility profile
  - GLSL 4.1 specification
  - http://www.opengl.org/registry

- Plus a set of ARB extensions
Core OpenGL 4.1
Get_program_binary

- Retrieve and set binary for a program object
  - Can be created by offline compilers

- Loading a binary can fail
  - Link status will be FALSE

- Loading a binary will
  - Replace the current binary for the program object
  - Initialize uniform values
  - Restore vertex shader inputs and fragment shader outputs

- Does not mandate a binary format
  - Left up to the GL implementation

- Once loaded, binary is valid for all legal GL state vectors
  - GL implementation deals with state dependencies

- Use PROGRAM_BINARY_RETRIEVABLE_HINT!
  - The GL can cache executable variations in the binary
Separate_shader_objects

- "Mix and match" approach to specifying shaders independently
  - For each shader stage there can be a program object

- Allow multiple program objects to be bound at once
  - Bound to a pipeline stage

- Program Object can contain shaders for one or more stages
  - Still need to be linked
  - Set PROGRAM_SEPARABLE program object flag

- Introduction of Program Pipeline Object
  - Container that contains program objects
  - Has five binding points, one for each shader stage
  - Does not need to be linked!

- Bind a program pipeline object to the context
Separate_shader_objects

• Unwritten input varyings for a stage are undefined
• Extra outputs in a shader stage are ignored in the next stage
• Interfaces between program objects need to match
  - Use Layout qualifier OR
  - Use interface blocks. Declare variables with same name, type and qualification
• **Introduction of ActiveShaderProgram()**
  - Redirect Uniform* API commands
• **Introduction of ProgramUniform*()**
  - Same as Uniform*() but takes a program object as parameter as well
• **Introduction of CreateShaderProgramv()**
  - Create a program object from source strings
• **Introduction of ValidateProgramPipeline()**
  - To figure out if it is all going to work together
ES2_compatibility

- ShaderBinary()
- GetShaderPrecisionFormat()
- ReleaseShaderCompiler()
- DepthRangef() and ClearDepthf()
  - OpenGL only has DOUBLE versions of these entry points
- FIXED type to specify vertex attributes
- Few details related to FBO completeness, reading / writing a buffer
- #version 100 -> 1.00 of OpenGL ES Shading Language
- Vector based uniform limits added
- OpenGL ES 2.0 rules for packing varying and uniform variables
Viewport_array

- Lifts restriction of one viewport per context
- Geometry shader can indicate viewport
  - Write to gl_ViewportIndex
- Separate scissor rectangle per viewport
- Viewport bounds are now floating point
Vertex_attrib_64bit

- Specify 64-bit floating-point components
- Introduces VertexAttribL*<d>()
- Introduces VertexAttribPointer()
- No automatic type conversion between attribute and shader variables
Shader_precision

- Restricts precision requirements for GLSL
- Arithmetic operators (‘+’, ‘/’ etc)
- Transcendentals (log, exp, pow etc)
- When NaNs and INFs are supported and generated
- Denorm flushing behavior
New ARB Extensions
ARB_shader_stencil_export

• Adds ability to generate stencil reference value in fragment shader
  - gl_FragStencilRef

• Can therefore write to stencil buffer from fragment shader
  - glStencilFunc(GL_ALWAYS, 0, 0xFFFF);
  - glStencilOp(GL_REPLACE, GL_REPLACE, GL_REPLACE);
ARB_robustness

• **Mechanism to develop more robust and secure applications**
  - WebGL

• **New Query entry points that specify number of bytes to write**
  - Instead of computed from a set of GL state
  - Example: ReadnPixelsARB(..., sizei bufSize, void *data)

• **Out of bound GPU accesses from buffer objects are robust**
  - No program termination anymore

• **Detect a graphics hardware reset**
  - Context will be unusable, application needs to re-create context
  - “Opt in” detection mechanism
ARB_create_context_robustness

- Robust buffer access is a context property
- Use CreateContextAttribsARB() and set
  - (CONTEXT_FLAGS_ARB, CONTEXT_ROBUST_ACCESS_BIT_ARB)

- Context Reset Notification is a context property also
- Use wglCreateContextAttribsARB() and set
  - (CONTEXT_RESET_NOTIFICATION_STRATEGY_ARB, LOSE_CONTEXT_ON_RESET_ARB)

- Once per frame, call GetGraphicsResetStatusARB() to find out the state
  - GUilty_CONTEXT_RESET_ARB
  - INNOCENT_CONTEXT_RESET_ARB
  - UNKNOWN_CONTEXT_RESET_ARB
  - NO_ERROR
ARB_debug_output

- Standardized mechanism to notify you when events occur
- Human readable string
- Callback mechanism or message log
- Source: OpenGL API, windowing system, compiler, debuggers, application
- Type: Errors, performance, undefined, portability, deprecated, other
- ID: distinguish within a (source, type) pair.
  - Example: INVALID_ENUM within source: OpenGL API
- Filter by severity: High, medium, low
- Filter by ID, or (source, type)
- Spec only requires GL Error messages from the OpenGL API
ARB_cl_event

- sync CreateSyncFromCLEventARB()

OpenCL event ➔ OpenGL Sync Object

- OpenCL event ➔ CL_COMPLETE ➔ Sync Object ➔ GL_SIGNALED

  CL_RUNNING

- OpenCL event ➔ CL_QUEUED ➔ Sync Object ➔ GL_UNSIGNALED
  CL_SUBMITTED
BACKGROUND SLIDES
New OpenGL 4 Pipeline

Array Element Buffer
Vertex Array Buffer Object (VAO)
Transform Feedback Buffers
Uniform Buffer Object (UBO)

Vertex data

Vertex Puller
Vertex Shading
Tessellation
Geometry Shading
Transform feedback
Fragment Shading

Pixel data
Texture Buffer Object (TexBO)
Pixel Unpack Buffer
Pixel Pack Buffer

Parameter data
Framebuffer

Texturing

© Copyright Khronos Group, 2010 - Page 34
More Detail – Vertex and Tessellation

- **Vertex position**
- **Current values**
- **Vertex position (clip coords)**
- **Associated data**
- **Primitive data (point, line, triangle, patch)**
- **Core Profile**
- **Compatibility**
- **OpenGL 4.0**

- **Lighting Xform Texgen**
- **Rasterpos (clip, project)**
- **Prim. Draw Mode state**

- **Primitive data**
  - **(point, line, triangle, patch)**

- **Per patch data**
- **Vertices**
- **Tessellation**
- **control shader**
- **Tessellation Evaluation Shader**
- **Primitive Assembly**
- **Vertices**
- **Tessellation Primitive Generator**
- **Primitive data (vertex pos, colors, other assoc. data)**

- **Tess parameters**
- **Connectivity**
- **Tess levels**
- **(u,v,w)**

- **Primitive data bypass**
More detail – Geometry and Follow-on

Primitive data (vertex pos, colors, other assoc. data) → Geometry shader → Primitive Assembly

Primitive mode: Bypass to stream 0

Stream 3 → Stream 2 → Stream 1 → Stream 0 → Transform feedback → Buffer object Binding points

Vertex position → Other assoc. data → Colors

Core Profile
Compatibility
OpenGL 4

To Rasterization

Vertex position (clip coords) → User clip planes

Other assoc. data → Flat shading

Clip Coords → Perspective Divide → Viewport transform → Front / Back Face selection

Normalized Coords → Window Coords

Final color processing → Other assoc. data

Colors → Other associated data
Feature adoption