WebGL 2.0

Zhenyao Mo, Kai Ninomiya, Ken Russell
Google, Inc.

Silicon Valley WebGL Meetup
March 2017
Background on WebGL

- WebGL is being used for all kinds of 3D visualization and presentation on the web today.
  - Artists showcase their work on Sketchfab, with over 1 million models shared and embedded across the web
  - The New York Times publishes visualizations in Three.js
    - Jupiter and Its Moons
    - The Dawn Wall, El Capitan’s Most Unwelcoming Route
  - Create amazing, fast, and interactive mathematical visualizations using MathBox
WebGL in Games

● Everyone has a web browser
  ○ WebGL “just works” for almost everyone!
    ■ WebGL Stats - 96%
    ■ caniuse - 91%

● **Unity - HTML5 target**
  ○ Publish Unity games straight to the Web with HTML5 and WebGL

● Open-source, pure-web 3D and game engines
  ○ PlayCanvas, Turbulenz, BabylonJS, Three.js
  ○ WebGL-accelerated 2D game engines
  ○ …
WebGL 2.0 + WebVR
PlayCanvas: After the Flood [Video]
PlayCanvas: *After the Flood*

- **Techniques used**
  - HDR+MSAA rendering, using z-buffer as a texture, procedural clouds using 3D textures, hardware PCF, alpha to coverage and transform feedback particles.

- **Demo features**
  - procedural clouds, procedural water ripples and reflection, leaf particles, animated trees, mirrors, dynamic lights, compressed textures, asset streaming, runtime lightmap baking, and more.
WebGL 2.0: Feature Overview

- WebGL 2.0 exposes the OpenGL ES 3.0 feature set
  - Brings desktop and mobile platforms’ features much closer together
- Features
  - Many sized texture formats
    - integer/float textures
  - 3D textures, 2D texture arrays
  - Immutable textures
  - Full non-power-of-two texture support
  - Instanced drawing*
- Multiple render targets*
- Transform feedback
- True integer vertex attributes
- Multisampled renderbuffers
- Many shading language upgrades
  - texture level-of-detail sampling control (important for physically-based rendering)
  - uniform blocks
  - in/out variables instead of “attribute”, etc.
  - control over layout in the shader

*Available via extensions in WebGL 1.0
WebGL 2.0: Feature Overview (continued)

**Features**

- **Seamless cubemaps**
  - Important for physically based rendering pipelines; can finally use mipmap generation effectively

- **Performant GPU-side copy/compute operations**
  - Upload textures from pixel unpack buffers, read pixels into pixel pack buffers, copying between buffers
  - Transform feedback
  - When used in conjunction with 3D textures and 2D texture arrays, have much more computation ability than WebGL 1.0

- **Uniform buffers, Vertex array objects**, Sampler objects
- **Query objects, Sync objects**

*Available via extensions in WebGL 1.0*
WebGL 2.0 is here!

- WebGL 2.0 now available in Chrome/Firefox!
  - Chrome: Released to desktop platforms - soon on Android
  - Firefox: Released on all platforms
  - Edge/Safari: plan to ship WebGL 2.0
  - Desktop support at 38% - to continue rising [WebGL Stats]

- WebGL 2.0.0 conformance updates
  - Test suite on track for ratification
    - Very thorough - run 10x longer than WebGL 1.0 tests
  - Chrome
    - Windows/macOS/Linux: Passing 100% in Chrome 56 (January)
    - Android: Passing 100% in Chrome 58 (April)
  - Firefox
    - Windows/macOS/Linux/Android pending; very nearly conformant
Looking Forward

● What’s in the future?
  ○ Compute shader extension
    ■ For advanced effects and GPGPU computation
    ■ Parallel data structures & algorithms
    ■ Physics simulations
    ■ Ray tracing
    ■ AI & neural networks
    ■ ...
  ○ A next-generation web graphics API
    ■ Widespread interest from developers and browsers to create a new graphics API
      ● Pre-validated design to reduce draw call overhead
      ● Implementable with excellent performance on Vulkan, Metal, and D3D12
      ● Secure design for the Web
      ● JavaScript & WebAssembly
Conclusion

- The amazing power of the graphics processor is available to web developers
- WebGL 2.0 provides a much needed upgrade to the capabilities
  - OpenGL ES 3.0 feature set
  - Will enable more amazing applications to be built

- Now we need you:
  What will you create with these graphics capabilities?
Many thanks to the many WebGL 2.0 contributors

- **Khronos Group**
  - Geoff Lang, Jamie Madill, Corentin Wallez, Shannon Woods
- **ANGLE team**
  - Olli Etuaho, Kimmo Kinnunen, Amal Prabhu, Barthold Lichtenbelt
- **NVIDIA Mobile Graphics Team**
  - Yunchao He, Qiankun Miao, Yang Gu, Xinghua Cao, Jiawei Shao, Yizhou Jiang, Guanxian Li, Chenglei Ren
- **Intel Web GPU Team**
  - incl. Brandon Jones, Victor Miura
- **Firefox team**
  - Jeff Gilbert
- **Chrome GPU team**
  - incl. Alec Miller, Figma
- **WebGL working group members**
  - Mark Callow, Rafael Cintron (Microsoft), Dean Jackson (Apple)
- **Mobica** (Janusz Sobczak and team)
- **Unity**
  - Jonas Echterhoff, Christophe Riccio, Marco Trivellato, ...
- **WebGL2Samples Pack team**
  - Patrick Cozzi, Trung Le, Shuai Shao, University of Pennsylvania
- **PlayCanvas, Sketchfab, Floored, Google Maps**
- **The Three.js community**
  - esp. Ricardo Cabello
- **...and many more collaborators in the open-source community**
  - Authors of linked demos
  - Alec Miller, Figma
  - Evgeny Demidov
  - Florian Bösch
  - m_panknin
  - Sijie Tian
  - …
Appendix A: Detailed Feature Overview

- With links!
  - Demos
  - WebGL 2 Samples Pack
  - Examples of techniques now possible in WebGL 2.0
Textures: More Formats

- **Floating-point textures, unsigned/signed integer textures**
  - A shader can read from and write to a texture with accurate values
    - No losing precision, no clamping (no need to worry about range)
    - Deferred rendering, scientific computation, etc
  - 8bit, 16bit, 32bit signed/unsigned integer formats; 16bit, 32bit floating-point formats
  - Demo: [Integer textures for RNG in Monte-Carlo simulations](https://example.com) [Evgeny Demidov]
  - Samples Pack: [simple integer texture demo](https://example.com)

- **sRGB formats**
  - Higher quality rendering in linear color space
  - Essential for [high dynamic range rendering](https://example.com) [Floored]
  - Samples Pack: [simple sRGB texture demo](https://example.com)

- **ETC2/EAC compressed textures via extension**
  - (Removed from core spec due to lack of hardware support on desktop platforms.)
Textures: 3D Textures

- **2D texture arrays & 3D textures**
  - Volumetric effects (fire, smoke, fog, etc)
  - Medical imagery (MRI, CT scans, etc)
  - Cached lighting effects
  - NVIDIA’s smoke box demo
  - Has been emulated before in WebGL 1.0, for example in Vicomtech’s demos, but now supported natively
  - Samples Pack:
    - Texture array demo
    - 3D texture demo
Textures: Immutable textures

- Dimensions/format/type will not change once initialized
  - Better performance - driver optimizations
  - When implementing WebGL on top of Direct3D, saves a lot of CPU memory
    - Direct3D doesn’t allow level-by-level texture uploading
    - A copy of all levels have to be kept in CPU for mutable textures

- Demo: Flat Wave (immutable RG32F) [Evgeny Demidov]
- Samples Pack: simple demo
Textures: Others

- Full support for non-power-of-two textures
  - Filtering, wrapping, and mipmapping of non-power-of-two textures
  - Long-requested feature; now it’s portable across devices

- Seamless cube maps
  - Used in physically based rendering pipelines
  - Now supported natively
Instancing

● To draw 1000 soldiers
  ○ WebGL 1: **1000** draw calls, with different position, posture uniforms, etc.
  ○ WebGL 2: Only **1** instanced draw call
  ○ Big performance improvement

● Demos:
  ○ Three.js demos:
    ■ instancing demo (single triangle)
    ■ indexed instancing (single box), interleaved buffers, dynamic updates
  ○ Brandon’s first demo of the extension:
    ■ WebGL instancing with ANGLE_instanced_arrays
  ○ Crowd demo [Github] (image)
  ○ Samples Pack: simple demo
  ○ Google Maps’ 3D mode uses instancing support
    ■ Reduces vertex buffer size by 6x (!) - thanks aappleby@ for the info
Multiple Render Targets

- Allow rendering to multiple framebuffers in one pass
- Critical for deferred shading
  - Lighting calculation on the entire scene once, not per individual object
  - Render depth, color, normals, etc into multiple buffers
  - Render one full-screen quad which performs all lighting calculations per-pixel
  - Big performance gain

Demos
- WebGL Deferred Shading [Sijie Tian, etc.] (image)
- Deferred Irradiance Volumes [Florian Bösch]
- Deferred Rendering [m_panknin]
- Spiral waves in excitable media [Evgeny Demidov]
Integer vertex attributes

● Go hand-in-hand with transform feedback, as well as integer texture support

● Examples
  ○ Maintain a pseudo-random number generator’s state per-vertex

● More generally:
  ○ Can send multiple integers into the vertex shader
  ○ Perform logical operations on them
  ○ Output them
  ○ More?
Transform feedback

- Save the output of the vertex shader directly into a vertex buffer object
  - Optionally skipping rasterization
- Caching and reusing GPU computation
  - Crowd demo [Github]
    - Instanced rendering of many skinned object
    - Skinning computations for just the few variants, instead of per-instance
- Stateful particle systems
  - Interactive particle demo [Github]
    - Per-particle state, particle birth/death
    - Simulates physics each frame
  - Lorenz strange attractor [Evgeny Demidov] (image)
  - Samples Pack: simple demo
- Non-web OpenGL examples:
  - OpenGL 4.0 review [Christophe Riccio], Noise-Based Particles advection shader [Philip Rideout]
Multisampled renderbuffers

● Efficient antialiased offscreen rendering
  ○ User controls number of samples
  ○ Useful for implementing techniques like vector curve rendering
    ■ Three.js vector curve rendering (image)
    ■ Rendering SVG paths in WebGL
    ■ Samples Pack: simple demo
Performant GPU-side copy/compute operations

- Copies between textures, pixel buffer objects
- Transform feedback
- Rendering to 2D texture arrays and 3D textures
  - Rendering to slices of 3D textures [Evgeny Demidov]
  - Samples Pack: rendering to texture array
- Many kinds of volume rendering algorithms now possible in WebGL
- With 3D textures and 2D texture arrays, 3D GPU fluid simulations
Features for performance optimization

- **Uniform buffers (potentially significant speedups)**
  - Allows updating many uniforms in a batch with one API call
  - Samples Pack: [simple demo](#)

- **Vertex array objects (also a WebGL 1.0 extension)**
  - Encapsulates all data that is associated with the vertex processor
    - Instead of the actual data, it holds references to the vertex buffers, index buffer, and layout specification
    - Usually set up once, and then just keep using it

- **Occlusion queries**
  - Used to determine if a render will be visible
    - E.g. occlusion-query a bounding box to see if it’s visible, for culling
  - Samples Pack: [simple demo](#)
Sampler objects

- Stores the sampling parameters for texture access inside of a shader
- Use N samplers with M textures - N+M objects instead of N*M
- Samples Pack: simple demo (image)
  - One texture with 4 different samplers
GLSL ES 3.00: `#version 300 es`

- **Samples Pack:**
  - Texture lookup in vertex shader, e.g. terrain, bump mapping (image)
  - Centroid interpolation
  - Fragment discard
  - Flat/smooth interpolators
  - Non-square matrices
  - Sampling textures by LOD
  - …
And more!

- There are more features not listed here
- And many new effects not yet implemented
  - Anything that’s possible with OpenGL ES 3.0
    - Terrain Rendering with Geometry Clipmaps [ARM Mali Developer Center]
    - ... 

- Dive in with the WebGL 2.0 Spec
  - Check out WebGL 2 Samples Pack for minimal examples
Appendix B: WebGL 2.0 Feature List

- OpenGL Shading Language ES 3.00
- 2D array and 3D textures
- Multisampled renderbuffers
- Transform feedback
- Uniform buffer Objects
- Vertex Array Objects
- Sampler Objects
- Pixel Buffer Objects
- Buffer-to-Buffer Copies
- Boolean occlusion queries
- Instanced rendering
- Multiple render targets
- Texture storage specification
- R and RG textures
- Seamless cube maps
- Non-power of two textures
- Texture LOD clamps
- Mipmap level base offset and max clamp
- At least 32 textures, at least 16 each for vertex/fragment shaders
- 16-bit and 32-bit floating-point textures
- 32-bit, 16-bit and 8-bit signed and
- Unsigned integer format renderbuffers, textures and vertex attributes
- 8-bit sRGB textures and framebuffers
- 11/11/10 floating-point RGB textures
- Shared exponent RGB 9/9/9/5 textures
- 10/10/10/2 unsigned normalized and unnormalized integer textures
- 10/10/10/2 signed and unsigned
- normalized vertex attributes
- 16-bit floating-point vertex attributes
- 8-bit-per-component signed normalized textures
- Sized internal texture formats