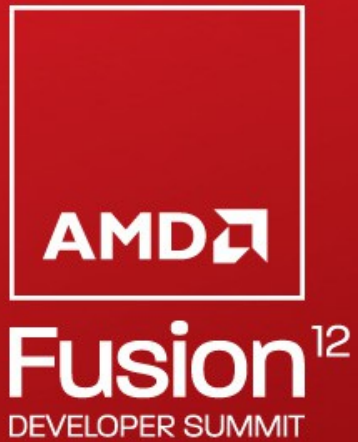
AMD

Fusion[12]

DEVELOPER SUMMIT



GPU Acceleration in Chrome

Kenneth Russell
Software Engineer
Google, Inc.

Motivation

- The web is hosting an increasing amount of rich media
  - HTML5 video
  - CSS 3D transforms
  - SVG and CSS filters
  - Canvas 2D
  - WebGL
  - CSS Shaders
- It's increasingly important and necessary to use the GPU to accelerate the rendering of this content

Agenda

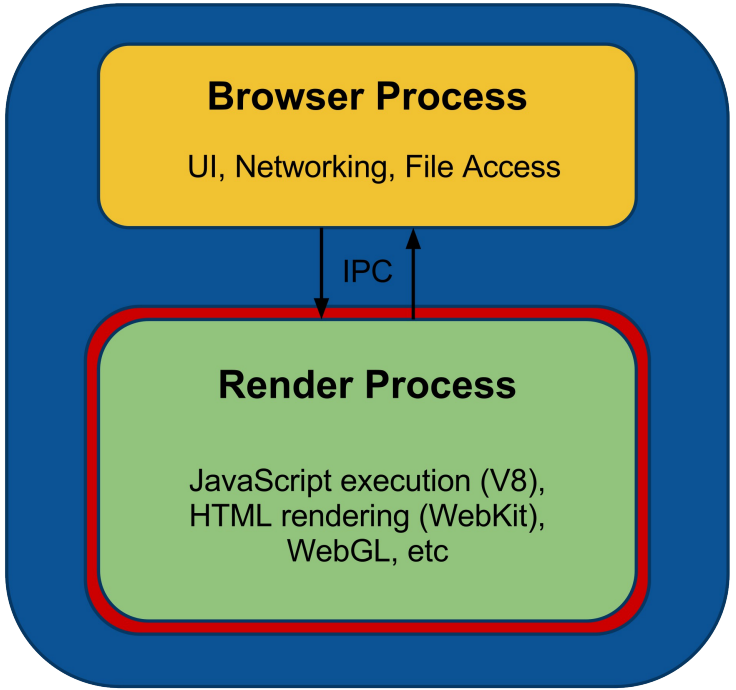
- Chrome browser architecture and security features
- Challenges of enabling GPU acceleration within the security framework
- How Chrome accesses the GPU
- GPU accelerating rendering of web pages
- Accelerated compositing
- How WebGL and other content fit within the compositing architecture
- Getting rendering results on to the screen
  - Windows, Mac OS X, Linux
- Current GPU acceleration challenges
  - Texture upload performance
  - Fully GPU-accelerated HTML rendering
  - Responsiveness and Robustness
- Conclusion
- Q&A

Design Principles of the Chrome Browser

- Speed, Simplicity, Security
- Security was a key focus of the project from the beginning
- The security architecture has many implications for the design of the browser

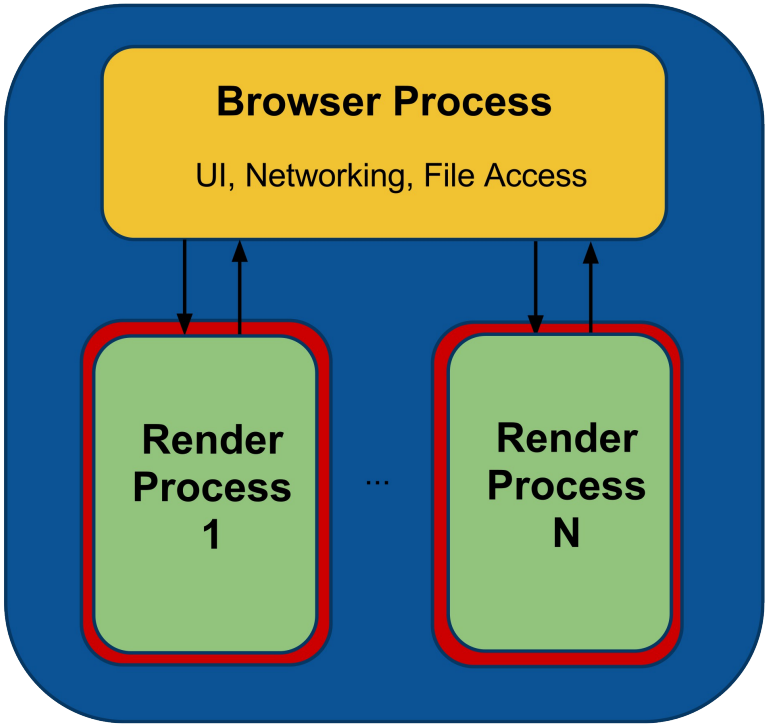
Security Architecture

- In Chrome, web content is executed in an subprocess sandboxed at the OS level
- No access to
  - File system
  - Network
  - On-screen windows
  - Graphics devices
  - …
- Even if an exploit is found allowing an attacker to take control of the process, the amount of damage that can be done is limited
- Defense in depth



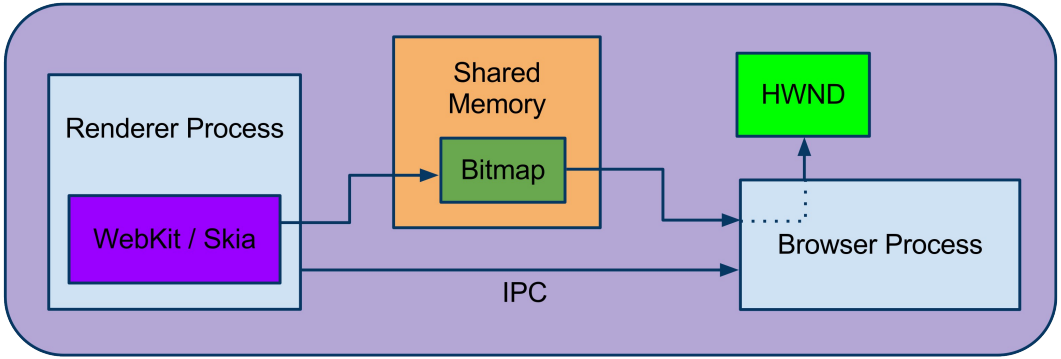
Multi-Process Architecture

- One main "browser" process drives all on-screen windows
- Launches multiple "renderer" sub-processes running the WebKit rendering engine
  - Typically, one per web page or one per domain
  - Reduces possibility of cross-site attacks
- Renderer processes are sandboxed
- All requests for privileged information are made via IPC to the browser process
  - Fetch network resources
  - Access disk cache
- Renderer process is untrusted
  - All inputs to browser process from renderers are validated



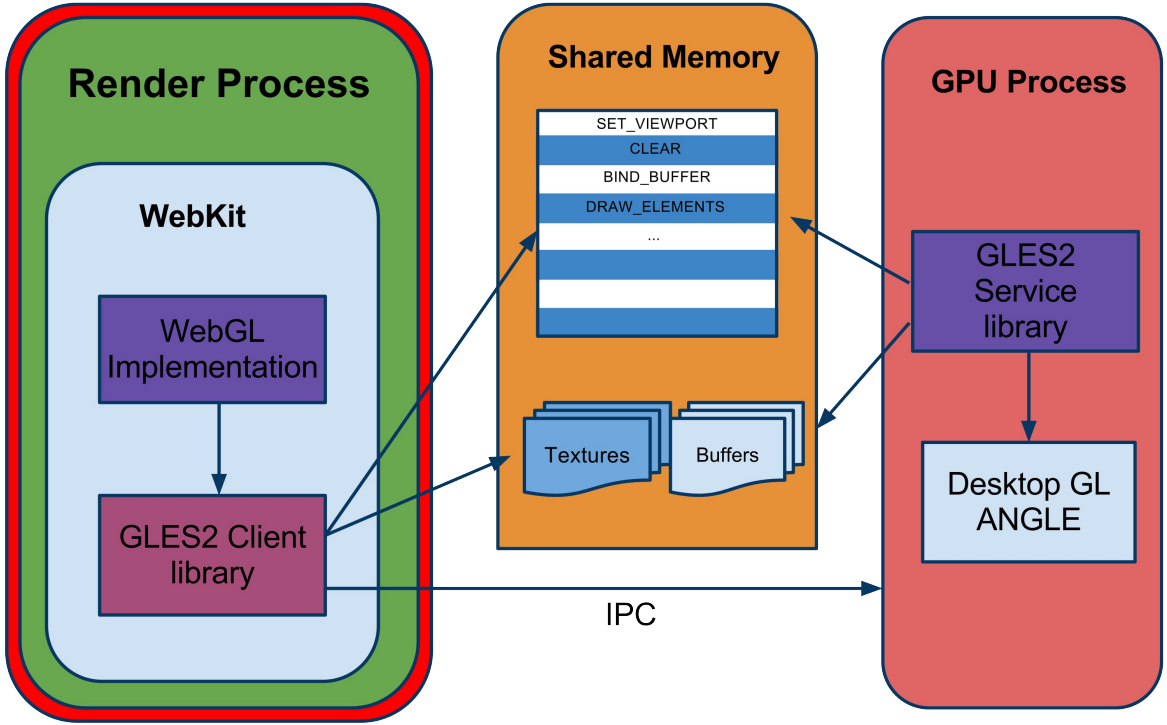
Multi-Process Rendering

- Chrome uses the open-source Skia 2D graphics library to render web pages
  - Text, images, geometry
- Chrome uses Skia not just as a library but also as an abstraction layer
  - For example, delegates to the platform's native text rendering APIs such as GDI and Core Text
  - To achieve parity with platform's text quality, look and feel
- Historically, rendering was done into shared memory using Skia's software backend
- Browser process received rendering results, maintained backing store and drew into on-screen windows



GPU Infrastructure

- When GPU acceleration was first added to Chrome, a "GPU" sub-process was introduced
- Renderer process issues what look like OpenGL ES 2.0 calls
- Commands are actually written into shared memory
- Interpreted and executed by the GPU process
- Similar to a graphics driver
- Architecture supports restarting of the GPU process
  - Sends lost context events to renderer processes



Command Buffer Subsystem

- Performs extensive validation to ensure security
  - Inputs from renderer process are untrusted
  - Avoids sending bad inputs such as out-of-range indices to graphics driver
  - Prevents uninitialized GPU memory from being viewed
- Enforces OpenGL ES 2.0 semantics everywhere
  - Limited NPOT texture support, vertex attribute 0 behavior, ...
  - Even on top of desktop OpenGL
  - Provides uniform behavior
- Increases parallelism on multi-core machines
  - Assuming the renderer process doesn't do any readbacks
  - Main cost is memory bandwidth
- Beneficial even on mobile devices
  - Used in Chrome on Android
- On Windows, uses open-source ANGLE project to provide OpenGL ES semantics on top of Direct3D 9
  - Considering using OpenGL ES 2.0 drivers on Windows where available

GPU Accelerating Web Page Rendering

- Several issues with rendering the entire web page using OpenGL
  - Quality and fidelity of text rendering
  - Quality and performance of path rendering
  - Overall performance of GPU accelerated 2D backend

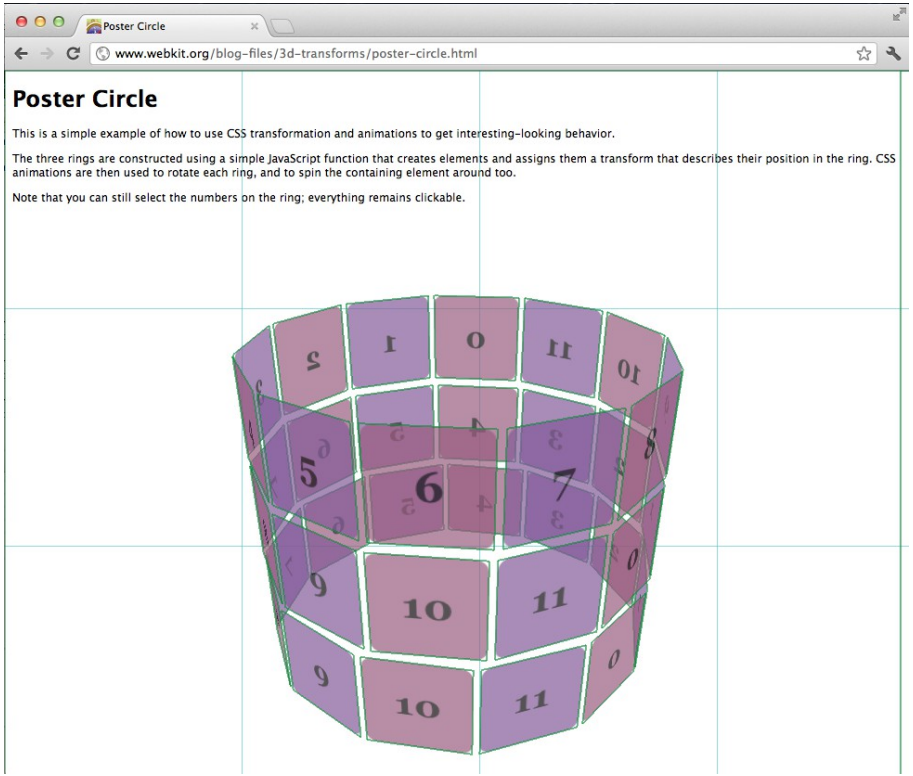
Incremental Approach

- New OpenGL backend developed for Skia (“Ganesh”)
- Supports the entire Skia API: text, images, paths, fills, filters, compositing modes
  - Translates 2D vector commands into OpenGL calls in the renderer process
  - Sent via command buffer to GPU process for rendering
  - Still delegates to platform's text APIs for sub-pixel antialiasing
  - Software fallbacks where absolutely necessary
- Some areas are significantly sped up over software rendering; some areas need more work
- Plan: continue to use Skia's software backend for majority of web page rendering
  - Begin using Skia's OpenGL backend in limited areas
  - Biggest bang for the buck
  - Gain confidence with this backend in the field

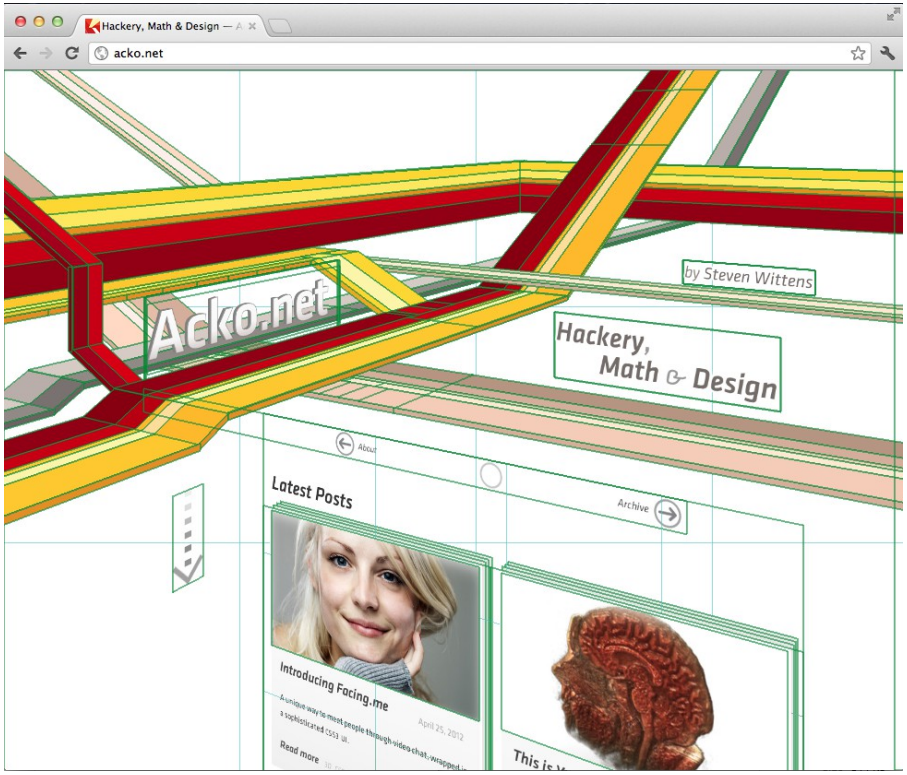
GPU Accelerated Web Page Compositing

- Web page is split into multiple layers
- Layers are rendered independently and composited afterward
- Layers are a concept internal to the browser; not part of the HTML specification
- Many reasons why a region of the page might be promoted to its own layer
  - Element has transformation CSS properties including 3D or perspective
  - <video> element / accelerated video decoding
  - <canvas> element with a 3D context or accelerated 2D context
  - Element uses a CSS animation for its opacity or uses an animated webkit transform
  - …
- Layers effectively map to textures on the GPU
- Texture mapped on to quads during the compositing process

Example Web Pages Illustrating Use of Layers



http://www.webkit.org/blog-files/3d-transforms/poster-circle.html



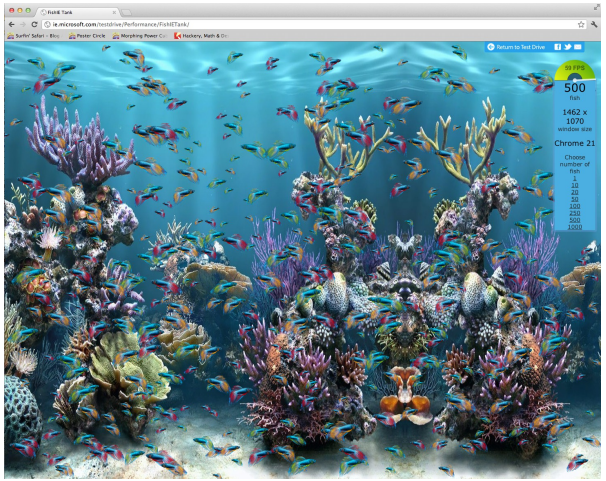
http://acko.net/

Software Rendered Layers

- Most layers containing HTML content are still rendered with software Skia
- Rendering results are uploaded into OpenGL textures
- Layers containing non-accelerated content are generally tiled
  - 256x256 textures
- Only visible tiles are actually rasterized
- Occlusion by fully opaque layers also taken into consideration
  - Avoid rasterization of invisible tiles
  - Sub-pixel antialiasing currently only supported on opaque layers

Accelerated 2D Canvas

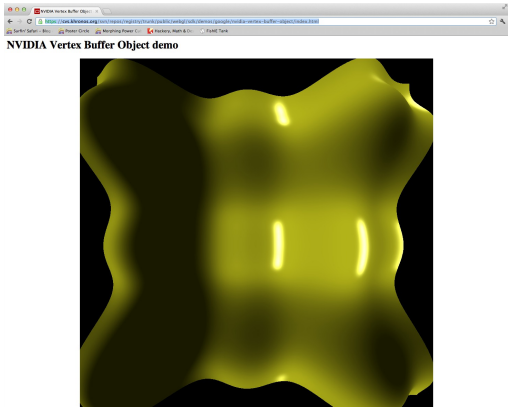
- HTML5 Canvas
  - JavaScript API for immediate mode vector graphics, text, images, bitmap manipulation
- First target for full GPU acceleration in Chrome
- Canvas 2D now implemented with Skia's new OpenGL backend
  - Straightforward mapping between Canvas API and Skia APIs
- Renders into a texture drawn by the compositor



http://ie.microsoft.com/testdrive/Performance/FishIETank/

WebGL

- Exposes the entire OpenGL ES 2.0 API to JavaScript
- JavaScript calls go more or less directly down to OpenGL
  - Some input validation is done in WebKit code
  - For security reasons, command buffer can not trust these results
  - Really expensive validation (like index validation) is done only by the command buffer
- WebGL uses its own OpenGL context
  - Renders into texture, possibly with intermediate multisampled renderbuffers
  - Shares resources with compositor's context to draw results to screen



https://cvs.khronos.org/svn/repos/registry/trunk/public/webgl/sdk/demos/google/nvidia-vertex-buffer-object/index.html

Getting Results on to the Screen

- The compositor runs in a renderer process and issues OpenGL calls to draw the various layers
- The GPU process executes the OpenGL commands coming from the renderer processes
- The browser process owns the on-screen windows
- How to get the compositor's results on to the screen?
- Different solution needed on each platform

Windows

- Originally the GPU process added a child window into the browser window's hierarchy
- Two problems
  - Wanted to sandbox GPU process as much as possible; deny access to on-screen windows
  - Synchronous window messages like focus transfer introduced possibility of deadlock
- Now a Direct3D texture is shared between the GPU and browser processes
- During buffer swap operation, GPU process inserts and waits for fence
  - Implemented on top of Direct3D Event Query
- Upon completion, browser process blits the texture to the screen via StretchRect into the swap chain
- Would be desirable to find better synchronization mechanism
  - In particular, GPU process issues query, but browser process waits for it

Mac OS X

- No notion of a cross-process window hierarchy
- GPU process renders into an IOSurface (10.6+)
- Browser process draws that IOSurface to the screen
- Both sides use OpenGL to bind the IOSurface to a texture
- During buffer swap operation, GPU process simply issues a glFlush
  - In conjunction with application level IPC, appears sufficient to guarantee order of execution of OpenGL commands between GPU and browser processes
- Currently only single-buffered
  - Have not seen any rendering artifacts such as tearing or incomplete rendering
  - IOSurface API very easy to use, but semantics are not well defined
- GPU process has been sandboxed on Mac OS X since its introduction

Linux with X11

- Renders directly into X window provided by browser process
  - Deadlock possibilities on Windows don't exist in X11
- GPU process is sandboxed as of Ubuntu 12.04
  - Tested with multiple vendors' graphics drivers, both proprietary and open-source
  - Enough system calls are whitelisted to allow the drivers to run
- Alternate rendering path exists using combination of XComposite and GLX_EXT_texture_from_pixmap
  - Not currently used; no security or functionality advantage

Linux with EGL

- Currently uses EGLImages to transfer rendering results across processes
- Each side binds the image to an OpenGL texture via glEGLImageTargetTexture2DOES
- Uses same fence-based synchronization during buffer swaps as on Windows
  - Would like to do better
  - Intend to switch to EGLStream if and when that is possible

Current GPU Acceleration Challenges

- Texture upload performance
- Fully GPU-accelerated HTML rendering
- Responsiveness issues with acceleration
- Robust out-of-range buffer access behavior on GPU

Texture Upload Performance

- Uploading of texture tiles to the GPU is expensive, in particular on low-end GPUs
- Can take tens of milliseconds in some situations
  - Texture swizzling during upload
- Eliminating all data copies is a goal
- Not currently possible even with pixel buffer objects
  - Mapping happens in the GPU process, not the renderer process
- One option: map/unmap storage for e.g. EGLStream without full access to OpenGL API
  - Only a couple of ioctl's necessary; sandbox can still be restrictive
  - Renderer process could draw software Skia directly into GPU memory

Fully GPU-accelerated HTML Rendering

- Skia's OpenGL backend currently translates 2D vector drawing commands to triangles on client side
- Some disadvantages to this approach
  - Higher bandwidth between renderer and GPU processes
  - Not possible to delegate to platform vector 2D APIs like Direct2D or OpenVG
- Work underway to send higher level 2D drawing commands from renderer to GPU process
  - SkPicture, SkGPipe
  - Defer and replay rendering of command stream
  - Avoid using textures as the representation of rendering results
  - Separate command streams for independent layers
- Issues with this approach
  - Synchronization between command streams
  - Maintaining correctness of HTML5 rendering model
  - HTML5 Canvas semantics allow unbounded growth of command stream

Fully GPU-accelerated HTML Rendering

- Text would still need to be rendered with platform APIs
  - Sub-pixel AA support still required on desktop and laptop displays
    - Not currently feasible to translate glyphs to paths and render them as paths
  - High-DPI displays simplify the problem
    - Avoid need for sub-pixel AA
  - Interoperability needed between platform text APIs and GPU path rendering
    - Supported on Windows via DirectWrite / Direct2D
    - Questions on other platforms (Core Text / OpenGL, ...)
  - High-quality text support somewhat incompatible with transformation and perspective projection
    - Often knowledge of the pixel grid is taken into consideration during font rasterization
  - Further investigation needed; ideas welcome

Responsiveness Issues with Acceleration

- Long-running draw calls can cause entire machine to become unresponsive
- More often occurs accidentally rather than maliciously
- Widely reported as a problem with WebGL, but not restricted to WebGL
- Possible with any API allowing direct or indirect access to GPU
  - Possible with just fixed function pipeline — no shaders needed
  - Possible with just lots of layers in a web page

Responsiveness Issues with Acceleration

- GL_ARB_robustness defines opt-in "kill switch"
  - LOSE_CONTEXT_ON_RESET context creation flag
  - Imposes implementation-defined timeout on OpenGL calls
  - This is the primitive needed for the browser to implement security policies
  - Some GPU vendors already implementing this; need all to do so
  - GL_EXT_robustness already spec'd for OpenGL ES
- On Windows platform, Timeout Detection and Recovery defines behavior
  - ANGLE implements GL_EXT_robustness in terms of TDRs
- Need to also prevent WebGL from affecting other OpenGL contexts on the system
  - Opt-in GL_ARB_robustness_isolation forthcoming (mainly for Linux, Mac OS)
  - Browser would use this to ensure it could not affect other OpenGL applications
  - Will almost certainly be some performance cost

Robust Out-of-Range Buffer Access Behavior

- Desire to delegate responsibility for out-of-range buffer access handling to GPU
  - Required in order to remove expensive index validation from Chrome's command buffer code
  - Would provide big speedup for certain kinds of applications
- GL_ARB_robust_buffer_access_behavior clarifies undefined areas in GL_ARB_robustness
  - Direct3D 10 capable hardware defines similar behavior
  - Intent is to specify behavior the hardware already supports
  - ANGLE would likely expose a compatible EXT_ extension

Conclusion

- Web browsers in general, and Chrome specifically, exercise GPU and graphics APIs differently than most graphics applications and games
- Much simpler in some ways, more complex than others
- Imposes security requirements on GPU not previously understood
- Looking forward to robust deployment of advanced web content on all devices

Q&A

GPU Acceleration in Chrome

AMD Fusion 12 DEVELOPER SUMMIT

Disclaimer & Attribution

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. There is no obligation to update or otherwise correct or revise this information. However, we reserve the right to revise this information and to make changes from time to time to the content hereof without obligation to notify any person of such revisions or changes.

NO REPRESENTATIONS OR WARRANTIES ARE MADE WITH RESPECT TO THE CONTENTS HEREOF AND NO RESPONSIBILITY IS ASSUMED FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

ALL IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED. IN NO EVENT WILL ANY LIABILITY TO ANY PERSON BE INCURRED FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD, the AMD arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. All other names used in this presentation are for informational purposes only and may be trademarks of their respective owners.

The contents of this presentation were provided by individual(s) and/or company listed on the title page. The information and opinions presented in this presentation may not represent AMD's positions, strategies or opinions. Unless explicitly stated, AMD is not responsible for the content herein and no endorsements are implied.