
Google™
10

GWT + HTML5 Can Do What!?

Ray Cromwell, Stefan Haustein, Joel Webber
May 2010

View live notes and ask questions about this session on Google Wave

http://bit.ly/io2010-gwt6

Overview

- HTML5 and GWT
- Demos
    1. Eyes
    2. Ears
    3. Guns

What is HTML5

- Formal definition
  - Best practices for HTML interpretation
  - Audio and Video elements
  - Other additional elements

- Colloquial meaning
  - Canvas
  - WebGL
  - WebSockets
  - CSS 3
  - LocalStorage
  - et al

GWT support for HTML5

- Very easy to build Java wrappers
- Many already exist in open-source projects
- Will be moving many of these into GWT proper (~2.2)
- Not part of GWT core yet
- GWT has always strived to be cross-browser
- Most new features are not available on all browsers

WebGL

- OpenGL ES 2.0, made Javascript-friendly
- Started by Canvas3D work at Mozilla
- Spread to Safari and Chrome via WebKit
- Canvas.getContext("webgl");

WebGL

Differences to OpenGL 1.x

- No fixed function pipeline (no matrix operations, no predefined surface models)
- Supports the GL Shader Language (GLSL)
  - Extremely flexible
  - Can be used for fast general computation, too
- Distinct concepts of native arrays and buffers
  - Buffers may be stored in graphics card memory
  - Arrays provide element-wise access from JS
  - Data from WebGL Arrays needs to be copied to WebGL buffers before it can be be used in graphics operations

Eyes: Image Processing

Image Processing

Photoshop Filters in the Browser

- Work on megapixel images
- At interactive frame rates
- Provide general purpose operations
  - scale, convolve, transform, colorspace ops
- Leverage native acceleration where possible

Image Processing

Pipeline / Tree Architecture

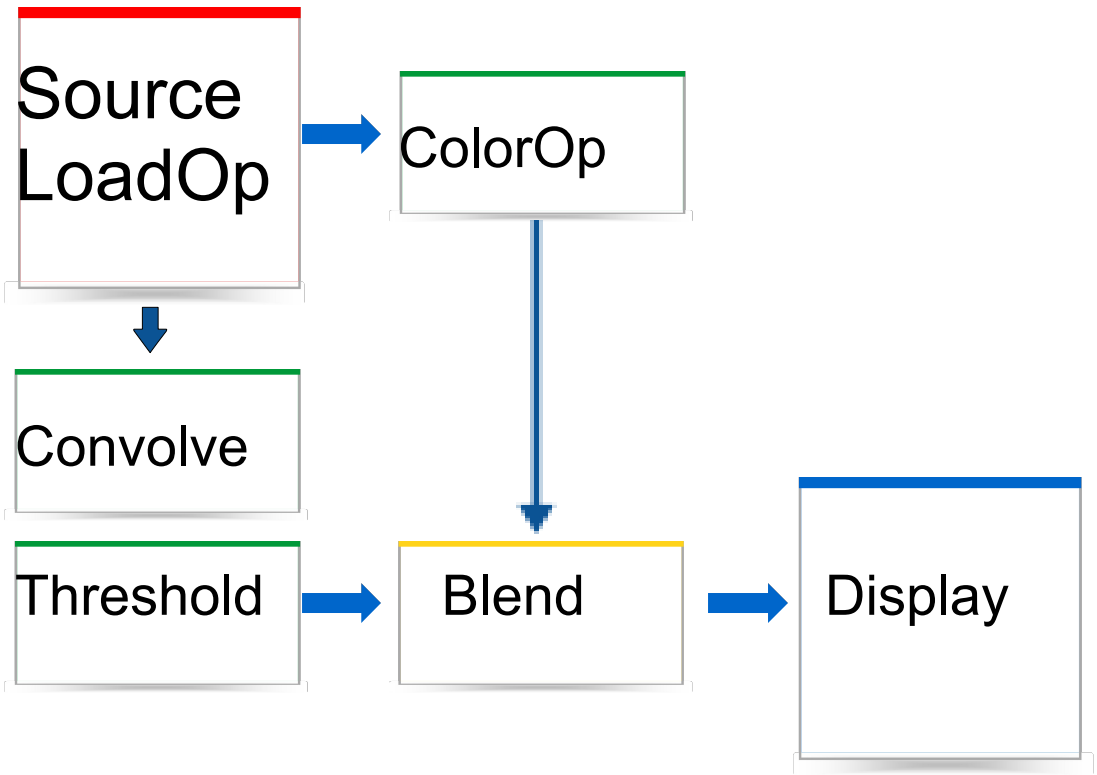


Image Processing
Two implementations

- 2D CanvasPixelArray "software" based (JS)
  - Likely slow on un-scaled megapixel images
  - Will need a "Preview" mode for interactivity
- WebGL Pixel Shader "hardware" version
  - Hopefully runs blisteringly fast

Image Processing
Example Operation: Convolution

- Weighed sum of nearby NxN pixels
- Many photoshop filters based on convolutions
- Sharpen Filter

[+0 -1 +0]
[-1 +5 -1]
[+0 -1 +0]

- Multiply upper left pixel by 0, upper pixel by -1, upper right by 0, multiply current pixel by 5, etc
- Add everything together, write result pixel back
- Notice all weights sum to 1 (-1 -1 -1 -1 5). This properly preserves brightness
- Other common convolutions: Gaussian Blur (good for drop shadow, glow), Unsharpen, Emboss/Edge Detect

Image Processing
Pure Java Implementation

- NxN convolve requires N*N multiplies, and adds
- For a 5x5 convolve on a 1024x1024 image
  - 25 million multiplies, adds, loads, and stores
  - TIMES 3 (red, green, blue, sometimes 4 for alpha)
  - up to 100 million operations
  - In Javascript!?
- 2048x2048 (4 megapixel) image is 4 times worse
- However, a 640x480 image is ~4x better

WebGL Implementation

- From GPU Gems and NVidia SDK
  - developer.nvidia.com/GPUGems/gpugems_ch27.html
- Load image into <canvas> or <img>
- Bind as texture
- Render screen-aligned rectangle with texture
  - to offscreen framebuffer
- Attach pixel shader with filter (e.g. convolve)
- Rinse / Wash / Repeat
- Copy final result back to <canvas>
- **DEMO TIME!**

Summary

- IMGwt Imaging API
  - Pure Java and WebGL impl available
  - Can run on AppEngine for cloud imaging
  - Android port also possible
  - Compact DSL:
    Imaging.load(url).convolve(array).to(widget);
- **WebGL filters are 1,700 times faster than JS**
  - **7000ms vs 4ms for 1024x1024 image, 3x3 kernel**

Ears: Audio Processing

Demo First!

Port of Commodore 64 Java emulator using GWT

- Emulates cycle-accurate C64 hardware @ 1mhz
- That means, 1 million times per second, it is
  - Loading an opcode and emulating CPU
  - Emulating memory fetch semantics
  - For I/O chips, it is calculating next values of every register
  - For Sound chip, there are 3 voices / waveform generators
  - In addition to emulating digital logic of the SID chip, it emulates analog filter circuitry

Port of Commodore 64 Java emulator using GWT

- SID music is not data, it is 6502 assembly code!
- Can't be played back perfectly without such emulation
- Output of emulation step is 1 16-bit 44khz sample, collected into a buffer
- When buffer fills, it is PCM-encoded into audio/wave format
- Result is then base64 encoded into dataURI
- Handled off to <audio> element

Guns: Quake II in the Browser

Quake II in the Browser

The Idea

- There is a Java port of the Quake II C source code called "Jake2"
- Could we use GWT and WebGL to cross-compile Jake2 to Javascript -- and run it in a browser without any plugins?
- Let's see...

Demo time!

Quake II in the Browser

Problems porting Jake2 to GWT

- Rendering
  - Lightweight Java OpenGL APIs (LWJGL) is based on OpenGL 1.x and uses Java nio buffers for data transfer
  - WebGL is based on OpenGL ES 2.0 (No fixed function pipeline) and uses WebGL arrays and buffers
  - No development mode support for WebGL in GWT
- Resource Loading
  - No file system in the browser
  - XHRs are asynchronous -- do not fit into control flow
  - LocalStore is synchronous but has limited capacity
- Networking: No UDP support
- Some Java APIs used in Jake2 are missing in GWT
- Performance: We had *no idea* if it would be fast enough

Quake II in the Browser

Renderer port: Drop-in LWJGL emulation?

It would be possible to emulate the missing GL 1.x fixed function pipeline, but...

- A pure drop-in LWJGL emulation would not take advantage of the DOM based image loading capabilities of WebGL
- Handling images at a pixel level for transfer to WebGL seemed too expensive in Javascript
- We needed slightly different signatures to support WebGL buffers (will come back to this later)

Quake II in the Browser

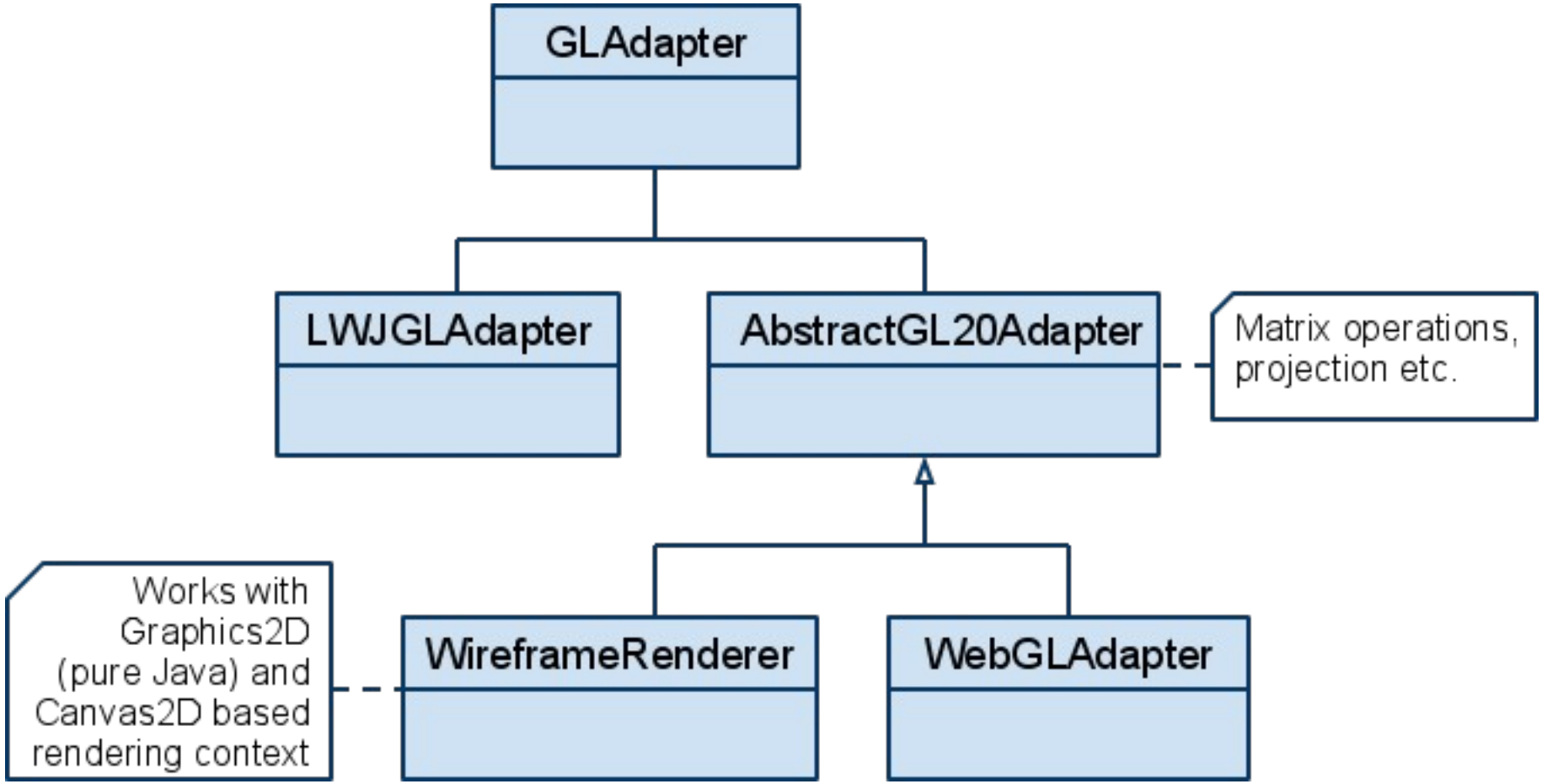
Renderer port

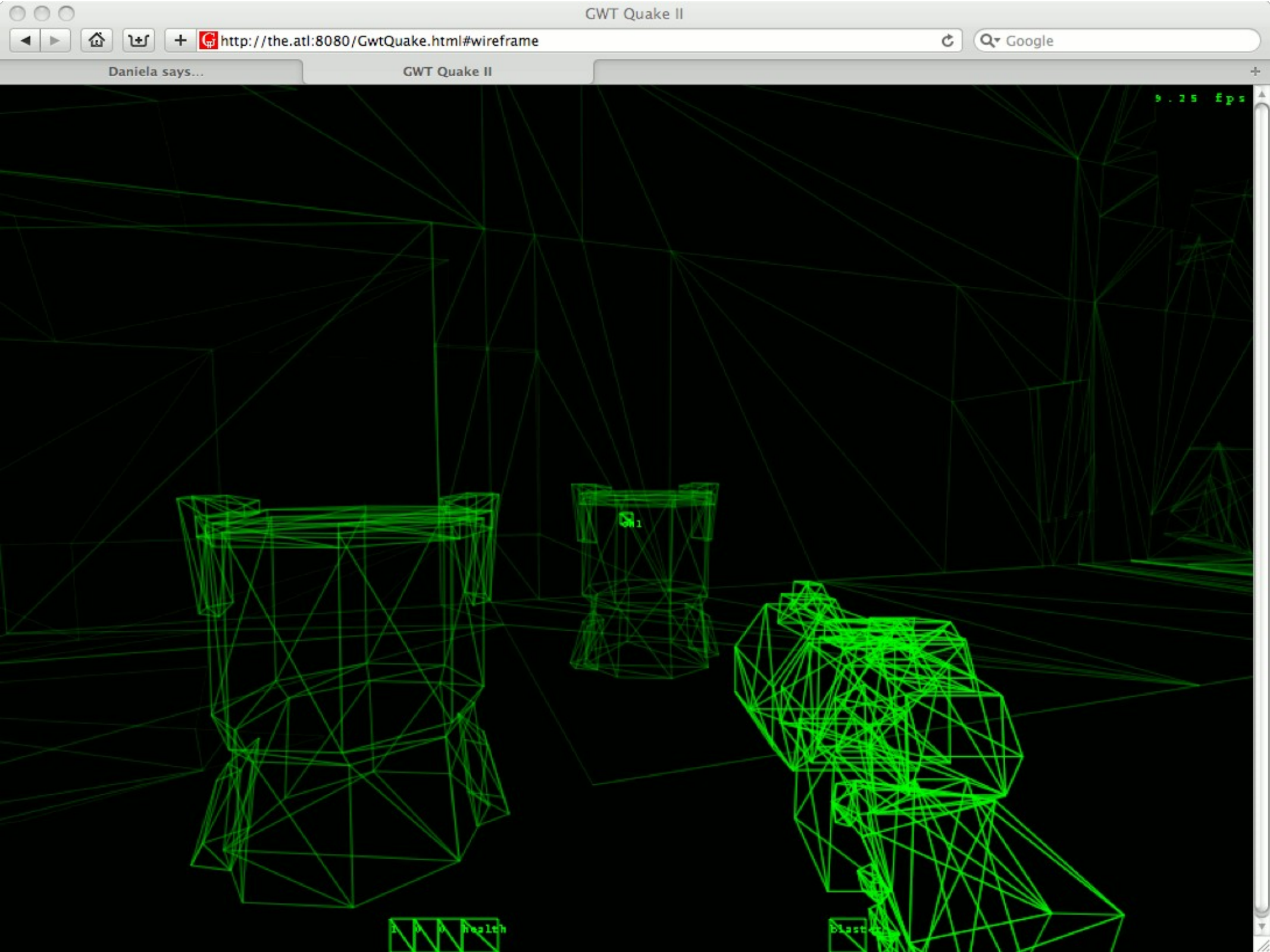
- Inserted an intermediate abstraction layer for Java and GWT that is close to the LWJGL API and still uses nio buffers (class AbstractGLAdapter)

- Java nio buffers were simple to emulate using WebGL arrays
  - Nice property of dev mode:
    The nio buffer emulation falls back to pure Java nio in dev mode

Quake II in the Browser

Renderer port




GWT Quake II
http://the.atl:8080/GwtQuake.html#wireframe
Google
Daniela says...
GWT Quake II
9.25 fps

Quake II in the Browser

WebGL buffer support

- Trivial translation from GL 1.x to WebGL is to turn all calls that set vector data into sequences of `glBindBuffer()`, `glBufferData()`, `glVertexAttribPointer()`
- To make this faster, we changed in the code to make sure that static data (level geometry, static lightmap coordinates) is only copied to a buffer once, and then reused

- In the code, we did this by assigning an unique id to reused buffers, so we can identify and skip the copy step for WebGL, but still run the code with LWJGL in pure Java (see glVertexAttribPointer in WebGLAdapter)

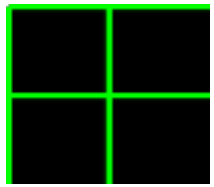
Quake II in the Browser

Other rendering improvements

- Ported GL_DrawAliasFrame in the Mesh class from glDrawElements() to glDrawArrays() to reduce number of data transfer calls
- Re-use part of the GL setup when rendering strings of characters, reducing the number of GL calls

Quake II in the Browser

Porting resource loading to async IO



- Texture Loading
  - Defer the corresponding GL calls to the onload event, using a "holodeck" texture in the meantime (see ImageLoader in GwtWebGLRenderer)
  - Straightforward, little impact on the control flow
  - The sizes of all images (needed when building the model) are sent to the client as JSON in index.html

- Models and Maps
  - Quake code assumes synchronous resources
  - Required significant refactoring to load models and maps asynchronously
  - The main trick was to skip frames while resource loading XHRs are pending

Quake II in the Browser

Networking and multiplayer

- Replaced UDP with WebSockets
- With the renderer, async IO and networking implementations, we got multiplayer games running.

Quake II in the Browser

Single player and demos

- Ironically, multiplayer support was the easy part
  - Did not need to port the server logic
- We wanted to get the demo running, so we would have something that we could use as a reproducible benchmark
- After making sure the required additional classes would compile, neither single player nor the demos worked
- The error must be somewhere in our changes, but diff showed changes in every single source file...

Quake II in the Browser

How do we find the bugs for single player and demos?

- Instead of trying to fully understand what goes wrong in one of the >200 classes, we tried to reduce the differences
- Reverted all unnecessary changes such as import cleanups, package changes and formatting changes
- Replaced other changes in the code with emulations where possible
  - Added a simple file system simulation, based on the local storage API (File, RandomAccessFile, LocalStorage) for saving preferences and games
  - Ported some additional parts of java.util
  - Pushed down networking changes from the Netchan and Msg classes to the lowest level (NET)

Quake II in the Browser

Single player and demo result

- Reduced the number of classes with differences to about 60*
  - Better manageable set of classes
- Also reduced the amount of changes in the remaining classes

- This did not provide much insight into the nature of the bugs we had in our code, but made them go away :)

* Before copyright header cleanup

Quake II in the Browser

Remaining Optimization

- Mesh animation:
  - Quake II animations based on keyframes
  - Point-cloud linear interpolation: terrible in Javascript
  - Move this to the vertex shader
- GWT compiler optimizations:
  - Initializing small arrays is too complex
  - Lazy static initializers hit too often
- Browser improvements:
  - Parsing binary data is *incredibly* slow in Javascript
  - Make WebGL TypedArrays general mechanism for operating on binary data
  - Spec and implementation already begun
  - FPS depending on window size hints at potential frame buffer handling improvements

Wrapping Up

- HTML5 features unlock incredible new functionality
- ... with the help of huge leaps in script performance
- These demos push the bleeding edge, but we'll be seeing more and more "native" apps in the browser:
  - Games
  - Media and design tools
  - Development tools
- Work remains to get all the APIs and tools right
  - Only way to know how is to try

Links and Thanks

Id Software (Quake II): idsoftware.com
Bytonic Software (Jake II): bytonic.de/html/jake2.html
Khronos (OpenGL, WebGL): khronos.org

Thanks to Vladimir Vukićević, who got the whole Canvas3D story started

Special Thanks to Lasse Oorni of Covert BitOps for the Commodore 64 music.

View live notes and ask questions about this session on Google Wave

http://bit.ly/io2010-gwt6


Google™
10