JavaScript and the browser as a platform for game development

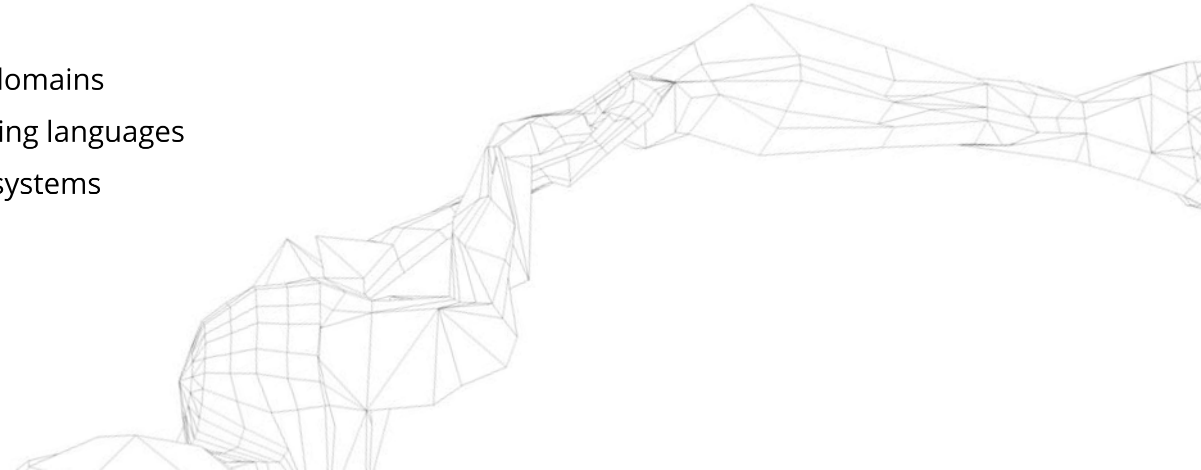
turbulenz

David Galeano

mloc.js 2014

David Galeano

- Co-founder and CTO of Turbulenz Limited since 2009
- Technical Director of EA Tech Graphics
- Team leader of RenderWare Graphics
- Game developer at Dinamic Multimedia
- In the video game industry since 1995
  - Worked on many platforms
  - Worked on many technology domains
  - Worked with many programming languages
  - Worked with many operating systems



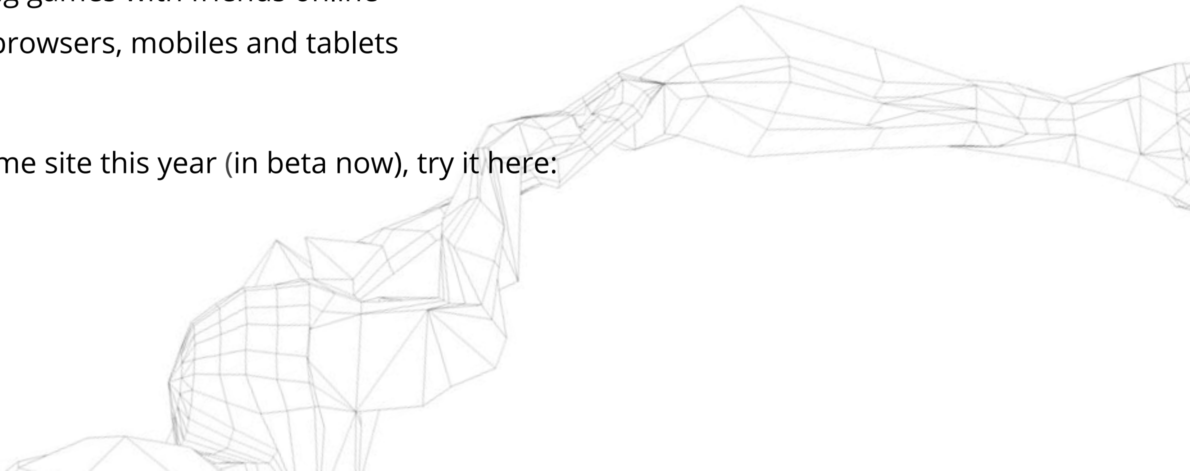
About Turbulenz

We have built the first high performance Internet generation game engine and web service called the Turbulenz platform.

- An industry leading HTML5 JavaScript 2D + 3D game engine
- A web service for social and game APIs accessible across all platforms
- A developer service for the publishing of content online
- An ultra modern destination for playing games with friends online
- Seamlessly working across desktops, browsers, mobiles and tablets

We are relaunching our consumer HTML5 game site this year (in beta now), try it here:

https://ga.me



What is possible today

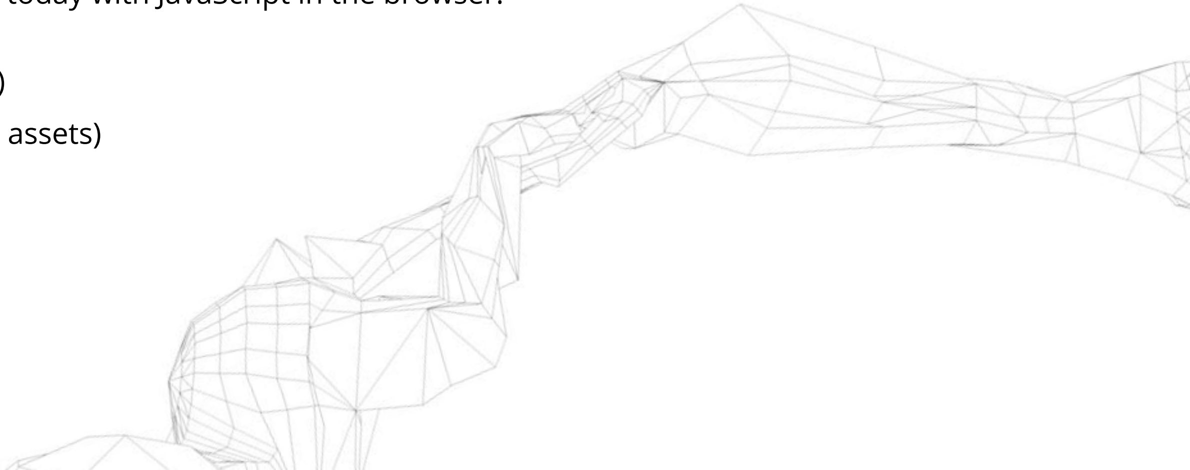
JavaScript in the browser is a viable platform for game development.

We can create high quality games.

The different layers between you and the hardware do affect performance, but not as much as in the past and the GPU is closer than ever.

These are some examples of what is possible today with JavaScript in the browser:

- **Third Person Action Game** (Polycraft)
- **First Person Shooter Demo** (Quake 4 assets)



Polycraft

turbulenz

- Developed by Wonderstruck
- Persistent island game combining arcade and strategy

ga.me/games/polycraft

Polycraft Challenges

turbulenz

- **~200K lines of code (ignoring comments or blank lines)**
  - Misspelled properties, missing arguments, wrong argument type, etc.
  - JSLint, JSHint, IDEs code intelligence help a lot but still not perfect
- **Level meshes built from thousands of small pieces in the in-game editor**
  - Would be CPU limited because of number of draw calls
  - These are merged into 32x32m blocks
- **Hundreds of dynamic renderables for complex bases**
  - Would also be CPU limited because of number of draw calls
  - Uses pseudo-instancing, with up to 32 copies of meshes, to reduce draw calls for common objects such as walls
  - Cut render calls by ~1/3, 10-20% total speedup
- **Mostly GPU limited because of expensive lighting model**

Quake 4 Demo

turbulenz

- Original game developed by Raven Software and id Software
- Multi Platform: Xbox 360, Windows, Linux, Mac OS X
- Just a demo to show that we can deliver similar visual quality to native applications

Quake 4 Demo Challenges

- **1694 assets to load**
  - Compression: 7-Zip better than gzip
  - Caching to reduce HTTP request: assets with unique names to be cached for 10 years
  - Archives to reduce HTTP requests overhead: group textures into tar files
- **353 lights, 346 particle systems**
  - Culling: portals, bounding box trees
- **55 shading techniques, 451 materials**
  - Minimize state changes: sort by technique and material
- **Mostly CPU limited because of scene complexity**
  - Some rooms have dozens of lights casting shadows

What we need for the future

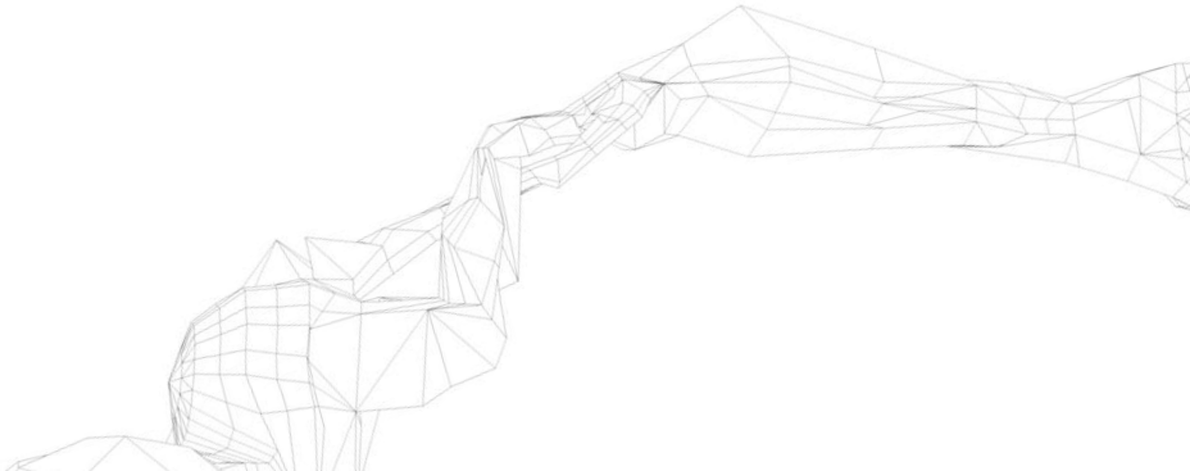
To make JavaScript and the browser the ultimate platform for game development, what else is needed?

Many things, but I am not going to talk today about monetization, mobile browsers, copyright protection, anti-cheating, etc.

I am just going to focus on JavaScript performance.

Where is the performance gap with native applications?

- Memory usage
- Parallelism
- Improved floating point performance



Memory Usage

turbulenz

Memory overhead still too high for standard JS objects.

More memory means more cache misses and potentially running out of memory and / or paging to disk.

Manually storing complex objects on typed arrays is hard to maintain.

Typed objects could be the solution

http://wiki.ecmascript.org/doku.php?id=harmony:typed_objects

We want to write code like this:

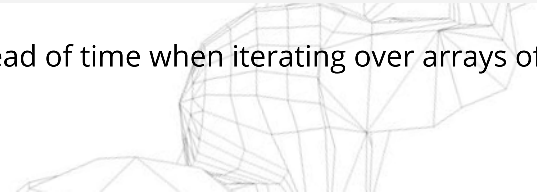
```
const Point2D = new StructType({ x: uint16, y: uint16 });
const Vector2D = new StructType({ x: int32, y: int32 });
const Color = new StructType({ r: uint8, g: uint8, b: uint8, a: uint8 });
const Pixel = new StructType({ point: Point2D, color: Color });

let a = Pixel({ point: { x:  0, y: 0 },
               color: { r: 255, g: 255, b: 255, a: 255 } });
let b = Pixel({ point: { x:  1, y: 1 },
               color: { r: 0, g: 255, b: 0, a: 255 } });
let a2b = Vector2D({ x: (b.point.x - a.point.x), y: (b.point.y - a.point.y) });
```

Memory Usage...

turbulenz

Instead of code like this:

```
// PhysicsContact
// [0,  3) : localA (vector3)
// [3,  6) : localB (vector3)
...
// [22,25) : Jacobian : normal x relA * iInertiaA (vector3)
// [25,28) : Jacobian : normal x relB * iInertiaB (vector3)
...
// [50,51) : bounce
// [51,52) : new
…
var contact = new Float32Array(52);
…
var ra0 = contact[6];
var ra1 = contact[7];
var ra2 = contact[8];
….
contact[22] = ((A0 * ca0) + (A3 * ca1) + (A6 * ca2));
contact[23] = ((A1 * ca0) + (A4 * ca1) + (A7 * ca2));
contact[24] = ((A2 * ca0) + (A5 * ca1) + (A8 * ca2));
```



We also need a way to prefetch memory ahead of time when iterating over arrays of typed objects, this is still fundamental in order to reduce cache misses.

Parallelism

We need to take advantage of multiple cores, even my phone has 4 of them! Web Workers are not perfect:

- **Data can be transferred but not shared**
  - Requires data duplication in many cases
  - Being able to share read-only data could help
- **Threads communicate by messages**
  - Hard to visualize and maintain complex trees of task dependencies
- **Hacky-ish way of embedding Web Worker code**
  - Not all browsers supporting Web Workers allow you to create Blob URLs
- **Number of CPU cores is not exposed**
  - Easy to oversubscribe or undersubscribe

Parallelism...

Parallel Array (River Trail) is useful but limited

http://wiki.ecmascript.org/doku.php?id=strawman:data_parallelism

- Requiring elemental functions may limit the kind of problems you can solve with them
- Every parallel operation generates a new array
  - It would be better to be able to provide a destination array
- Some browsers will not like the fact that parallel operations are blocking
- Would you be able to debug it?

WebCL is more powerful than Parallel Array

http://www.khronos.org/webcl/

- Allows non-blocking operations
- Total control over source and destination buffers
- But OpenCL kernels can only be debugged outside the browser
- Still limited in the kind of code it can execute in parallel
- And we have no idea if / when it would be available without requiring a plugin...

Parallelism...

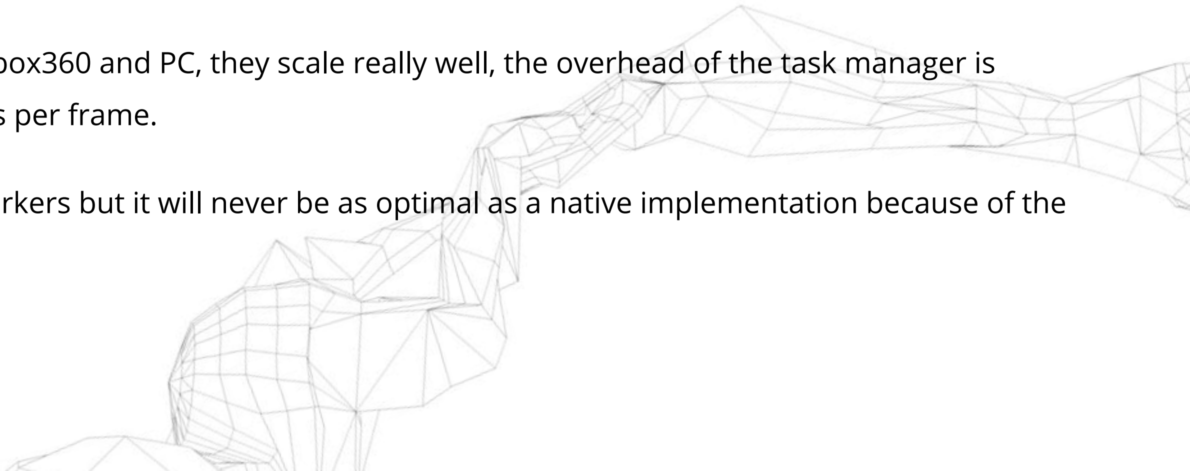
Ideal API would provide a task management system based on data access and ownership

- Tasks would be declared to operate on specific buffers with specific read / write requirements
- Task manager will build dependency tree based on task order and data access requirements
- Task manager will distribute individual tasks among all available cores

This is similar to how some declarative build systems work, you declare all the tasks and their explicit dependencies and the build system takes care of building everything in parallel in the right order.

We implemented systems like that for PS3, Xbox360 and PC, they scale really well, the overhead of the task manager is minimal unless you create thousands of tasks per frame.

Could be partially implemented with Web Workers but it will never be as optimal as a native implementation because of the limitations previously described.



Improved floating point performance

Games spend most of their time doing floating point operations but we are not taking advantage of the full potential of modern CPUs.

32-bit floating point operations

- Most games do not need 64-bit floating point operations but JS standard requires it
- Early experiments using Math.fround show 10-20% improvement
- But it is expensive to modify thousands of existing lines of code to add Math.fround everywhere
    - `contact[22] = (Math.fround(Math.fround(A0 * ca0) + Math.fround(A3 * ca1)) + Math.fround(A6 * ca2));`
- Would be simpler for game developers if language spec could be relaxed to allow 32-bit operations when the final result of a sequence of operations is stored to a 32-bit location

SIMD

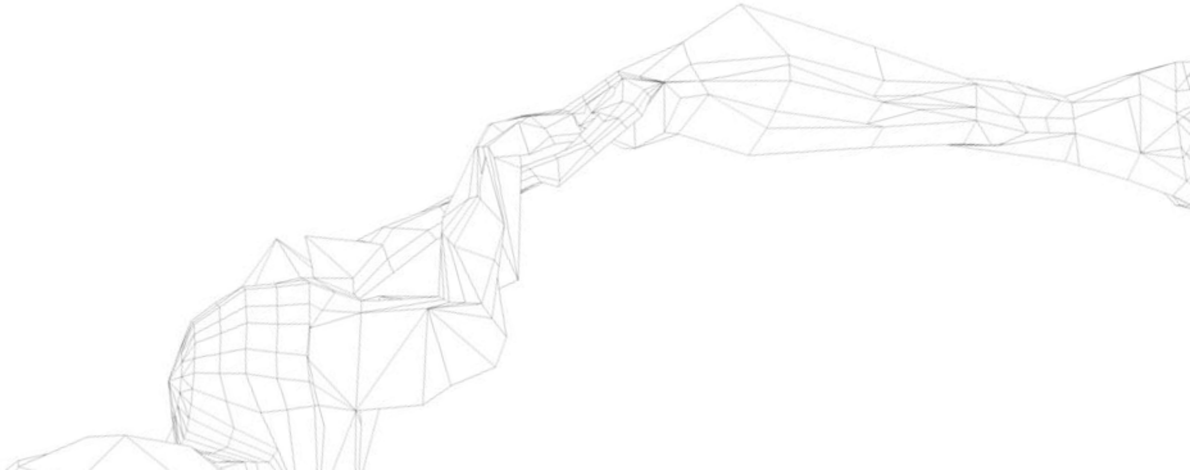
- Many vector and matrix operations could be optimized with SIMD instructions
- Would need good integration with Typed Arrays and/or Typed Objects
- Exiting proposal has limited types, Mono.Simd is more thorough
    - https://github.com/johnmccutchan/ecmascript_simd
    - http://docs.go-mono.com/?link=N%3aMono.Simd
- Also expensive to modify existing vector math libraries to use it
    - But gains should be bigger and probably worth it

Improved floating point performance

Crazy idea: runtime settings?

- “use single-precision”;
- “use fast-math”;

Native applications have access to compiler settings that can be used to tune code generation for speed. For example, GCC has “-ffast-math” and Visual Studio has “/fp:fast” to relax the rules on floating point operations, allowing optimizations that may not generate precise results but good enough in many cases.



Questions & Answers

Game site:

https://ga.me

Twitter:

@turbulenz

@wonderstruck_uk

Developer site:

http://biz.turbulenz.com

David Galeano:

@davidgaleano

davidgaleano@turbulenz.com

Open Source Turbulenz Engine:

https://github.com/turbulenz/turbulenz_engine

Online Turbulenz Engine documentation:

http://docs.turbulenz.com