Game Development In

JavaScript

First Impressions
Javascript

- Designed for web application
- Had a rough childhood
- Very misunderstood
- Performance difference across browsers

Benchmark

- Pros/Cons
### JavaScript: WebGL & Canvas

<table>
<thead>
<tr>
<th>Canvas API</th>
<th>WebGL equivalent</th>
<th>Three.js framework equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>scale</td>
<td>teximage2D + modify vertices</td>
<td>Camera</td>
</tr>
<tr>
<td>rotate</td>
<td>teximage2D + modify vertices</td>
<td>Camera</td>
</tr>
<tr>
<td>translate</td>
<td>teximage2D + modify vertices</td>
<td>Camera</td>
</tr>
<tr>
<td>transform</td>
<td>teximage2D + modify vertices</td>
<td>Camera</td>
</tr>
<tr>
<td>clearRect</td>
<td>clear+viewport, or teximage2d</td>
<td>Textures</td>
</tr>
<tr>
<td>fillRect</td>
<td>clear+viewport, or teximage2d</td>
<td>Textures</td>
</tr>
<tr>
<td>strokeRect</td>
<td>teximage2d</td>
<td>Textures</td>
</tr>
<tr>
<td>path</td>
<td>teximage2d+stencil + vertex paths</td>
<td>Camera + objects</td>
</tr>
<tr>
<td>fillText</td>
<td>teximage2d+stencil + vertex paths</td>
<td>Camera + objects + textures</td>
</tr>
<tr>
<td>strokeText</td>
<td>teximage2d+stencil + vertex paths</td>
<td>Camera + objects + textures</td>
</tr>
<tr>
<td>drawImage</td>
<td>teximage2d</td>
<td>textures</td>
</tr>
<tr>
<td>createImageData</td>
<td>fill buffer of teximage2d in CPU application</td>
<td>CPU application code</td>
</tr>
<tr>
<td>getImageData</td>
<td>readpixels</td>
<td>readpixels</td>
</tr>
<tr>
<td>putImageData</td>
<td>fill buffer of teximage2d in CPU application</td>
<td>fill buffer of teximage2d in CPU application</td>
</tr>
</tbody>
</table>

- programmable shaders (vertex, fragment)
- offscreen buffers
- efficient 3D representation of depth
- efficient POINTS rendering

+ Convenience methods - Materials (sprite ..)
+ Convenience methods - Lights
+ Convenience methods - Scenes
 Javascript Graphics libraries

- THREE.js
  - 3D only
  - Performance test

- PIXI.js
  - 2D only
  - Performance test
Google is your friend

- StackOverflow
- Shadertoy
- CGLearn ;)
- Gamedev.net
- TigSource
Lets Talk Game cycles

VS

Interpret input
Update State
Apply Input
Collision Detection
Resource Management
Rendering

Interpret input
Rendering

Update State
Apply Input
Collision Detection
Resource Management
Worker

- Great if you want to utilize more than 1 core on your CPU
- Completely separated from main context!
- Send data as messages.
- Benchmark
Memory management in Javascript

● Scopes and vars
  a = “such Waste”
  var a = “such Efficient”
  // Much wow!
● Delete Keyword
● Require JS + Object Oriented approach = ❤️
Environments

- Linear Environments (Half Life 2, Duke Nukem)
- Zoned Environments (Guild Wars 2, Age of Conan)
- Streamed Environments (Skyrim, World of Warcraft, Gothic 3, GTA 5, Minecraft)
Example: Minecraft World

- World is defined by Chunks.
- Seed is used to generate Chunks.
- Chunks are loaded and unloaded from the “scene“ as player travels around.
- Unloaded Chunks are not deleted
Example: Minecraft World - Farlands

Normal Minecraft

Edge Far Lands

The Corner Far Lands
Dynamics

Games are more interesting if objects within the game can interact.

- Interactions
- Physics

All such activities require us to know something about proximity
Physics Engines/Libraries

- Cannon
- Ammo
- Box2D
- PhysiJS
Collision Detection

Separated into two phases:

- Broad phase
- Narrow phase
Collision Detection: Broad Phase

Find potential collision candidates in space.

Problem?

Methods to partition space

- None
- Grid
- Tree structures (example)
Collision Detection: Narrow Phase

Using Mesh for collision testing is expensive, let's use simplification instead.
Collision Detection: Narrow Phase

Object to Object collisions testing

- Separated Axis Theorem
- Line segment intersection
Collision Detection: False Negatives/Positives

Collision?  

Collision!