## Web Physics Summary

| Web Physics | • Transparent access to native 2D physics engine through JS APIs.  
• OpenCL accelerated 2D Physics Engine (Box2D-OpenCL). |
|--------------|-----------------------------------------------------------------------------------------------------|
| Motivation   | • High performance, interactive compute & graphics apps on mobile  
• Transparent & efficient access to acceleration from web apps. |
| Goals        | • Cross-platform, portable, accelerated, robust & flexible |
| Usages       | • Real time mobile gaming; Compute intensive simulation;  
• Computer Aided modeling; Physical effects & realism;  
• Augmented Reality; Real-time big data visualization; |
| Approach     | Hybrid Web Physics approach:  
• Accelerate using native multicore hardware & open parallel APIs.  
• Expose parallelism through JS APIs to web applications. |
| Accomplishments | • Web Physics JavaScript API Framework (Web Physics JS APIs)  
• OpenCL accelerated 2D Physics Engine (Box2D-OpenCL)  
• JavaScript Physics Engine (Box2DWeb 2.2.1 APIs) |

### Web Physics Summary Diagram

- Games
- Media Apps
- Web Apps
- JavaScript Libraries
- Web API
- Web Physics JS Bindings
- Native Apps
- OpenCL Accelerated Box2D--OpenCL
- Native Physics Engine
- OpenCL Device Driver
- Multicore CPU
- Manycore GPGPU

---

TIZEN DEVELOPER CONFERENCE 2015 SAN FRANCISCO
Web Physics Approach
## Web Physics Overview

<table>
<thead>
<tr>
<th>Architectural Decision</th>
<th>Reasoning</th>
</tr>
</thead>
</table>
| Accelerated Box2D      | • JS performance a concern for high compute use cases.  
                         • OpenCL parallelization API for heterogeneous multicore devices (CPU, GPU) can provide needed performance. |
| JavaScript bindings for physics engine APIs | • Portable Web APIs for compute intensive web apps.  
                                             • Cross platform application developer support  
                                             • Transparent access to accelerated library, without requiring parallelization of apps. |
| Modular architecture   | • Incremental parallelization of physics engine pipeline  
                         • Ease of testing. |
| Preserve Box2D open source APIs | • Leverage existing ecosystem: To ensure that existing apps using Box2D physics engine can use Box2D-OpenCL |

**Diagram:**
- **Box2D-OpenCL**
  - OpenCL Accelerated 2D Physics Engine Rigid Body Pipeline
  - Accelerated native applications
- **OpenCL API**
- **Web API**
- **Web Physics JS APIs**
  - JavaScript Bindings
- **Accelerated Web Apps**

**Web Physics components**
- Multicore CPU
- GPGPU
Web Physics Requirements

1. **Transparent and efficient access to acceleration on client** through JS APIs, from Web applications:
   - JS Bindings implemented in WebKit-based browser engine
2. **Near-native performance** for high compute web simulation:
   - Hybrid approach: Access to accelerated native physics engine through JS bindings
   - $\leq 1.5X$ performance impact from JS binding, relative to native physics engine
3. **Full feature support, and no API change:**
   - Complete Box2D features and API support in accelerated native physics engine
   - No API change: Box2D-OpenCL APIs should be identical to those of Box2D 2.2.1
4. **Complete JavaScript API support** in JS physics engine for benchmarking
   - Box2DWeb 2.2.1 JS physics engine supporting all Box2D 2.2.1 features.
5. **Acceleration using open parallel API for heterogeneous platforms:**
   - OpenCL accelerated physics engine for multicore CPU and GPGPU (Box2D-OpenCL)
JavaScript Bindings
Web Physics Architectural Approaches

<table>
<thead>
<tr>
<th>Plugin-Approach</th>
<th>WebCore Approach</th>
<th>Inject Bundle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pros</td>
<td>Pros</td>
<td>Pros</td>
</tr>
<tr>
<td>Browser portability.</td>
<td>Best comparative performance.</td>
<td>Usable by native and web apps</td>
</tr>
<tr>
<td>Maintainability: Less impact from physics engine changes</td>
<td>No IPC overhead</td>
<td>Portability</td>
</tr>
<tr>
<td>Code reusability</td>
<td></td>
<td>Maintainability</td>
</tr>
<tr>
<td></td>
<td>Difficult to Maintain</td>
<td>No IPC overhead</td>
</tr>
<tr>
<td></td>
<td>More complex</td>
<td></td>
</tr>
<tr>
<td>Cons</td>
<td>Cons</td>
<td>Cons</td>
</tr>
<tr>
<td>IPC communication expensive between plugin &amp; web process</td>
<td>Difficult to Maintain</td>
<td>Complexity</td>
</tr>
<tr>
<td></td>
<td>More complex</td>
<td>Not Reusable across browsers</td>
</tr>
</tbody>
</table>
Box2DWeb 2.2.1 JavaScript Physics Engine

- Physics Engine written entirely in JavaScript.
- Existing open source project (box2dweb) had implemented Box2D 2.1.2 APIs.
- Box2DWeb 2.2.1 JS physics engine implements JS APIs corresponding to all Box2D 2.2.1 APIs.
- Used for benchmarking & comparative analysis.
- Currently in the process of being open sourced.
Web Physics Binding Implementation

✧ **IDL code generator features:**
   + Constructor overloading
   + Support for constructor-type static read-only attribute in IDL
     - Patch up-streamed to WebKit.org
     - JSC specific binding classes, functions generated from Box2D IDLs. Code generator can support JSC & V8.

✧ **Wrapper Layer features:**
   + Interfaces with native Box2D physics engine. A glue layer from Box2D native engine to auto-generated JS binding classes.
   + Most code is JS engine independent
   + Complete support for Box2D classes
   + 1:1 mapping to Box2D native objects
   + Preserves Box2D tree structure
   + Callback function support using Listeners
   + Supports DebugDraw

✧ **Performance overhead <= 1.5x relative to Box2D**
   - Includes parsing overhead of JavaScript code, and binding code overhead

✧ **Implementation restrictions:**
   - Strict type checking, relative to Box2DWeb
   - Native classes & functions written into IDL files, & their implementation added manually
OpenCL Acceleration
Box2D-OpenCL Parallelization

Collision Detection:
- *Broad Phase* and *Narrow Phase*.
- Enforces natural physical constraints on simulated physical objects.
- Detects collisions.

Constraint Solver:
- Computes velocity and position constraints.
- Updates velocities and positions of bodies.

Benchmarking:
- *Collision Detection & Constraints Solver* prioritized for parallelization.
- \(~64\%\) of total time spent in *Collision Detection* stage.
  - \(~21\%\) of time spent in *Broad Phase*.
  - \(~43\%\) in *Narrow Phase*.
  - \(~35\%\) of total time spent in *Solver*. 
Collision Detection – Broad Phase

**Goal:** Quickly find pairs of objects that *might* collide each other, and cull out *all* pairs that *cannot* collide each other.

**Algorithm:** Axis Aligned Bounding Box (AABB) used to approximate objects for fast testing

<table>
<thead>
<tr>
<th>Sequential Broad Phase:</th>
<th>Parallel Broad Phase:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• BVH (Bounding Volume Hierarchy)</td>
<td>• SaP (Sweep and Prune) [1]</td>
</tr>
<tr>
<td>• Traverse from top to bottom</td>
<td>• Compute an interval ([x_i, X_i]) for each AABB (O_i)</td>
</tr>
<tr>
<td>• Good for sequential programs but not efficient for parallel programs due to low parallelism in higher levels of BVH</td>
<td>• Sort all AABBs by (x_{\text{min}})</td>
</tr>
<tr>
<td></td>
<td>• For all AABBs (O_i), execute the followings in parallel:</td>
</tr>
<tr>
<td></td>
<td>• Sweep from (x_i) to (X_i) to find all (O_j) with (x_j \in [x_i, X_i]) and (i &lt; j)</td>
</tr>
<tr>
<td></td>
<td>• For each (O_j), if (O_i \cap O_j \neq \emptyset), output a pair ((O_i, O_j))</td>
</tr>
</tbody>
</table>

Collision Detection – Narrow Phase

**Goal:** For each pair generated by BP, test whether the two objects collide or not. If collide, generate a manifold for the pair.

**Algorithm:** Separating Axis Theorem (SAT) used to test intersection of two objects.

### Sequential Narrow Phase:
- Loop over all pairs
- Check each pair using different functions for different types (e.g. polygon-polygon, polygon-edge, circle-edge, etc.)

### Parallel NP (Solution 1):
- Execute a single kernel for all pairs, using different branches for different types inside the kernel:
  - if polygon-polygon pair
    - Compute for p-p type
  - else if circle-edge pair
    - Compute for c-e type
  - ...

### Parallel NP (Solution 2):
- Execute a multiple kernels for different type of pairs, each kernel deal with one type of pair only.

➢ **Solution 2** has better performance
Collision Detection – Narrow Phase

- At the end of NP stage, we compact all “valid” pairs (i.e. whose two objects collide with each other), to the head of the list for the next stage.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
<th>N</th>
<th>O</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

input pairs (from BP)
validity masks (1-valid, 0-invalid)
exclusive scan results ($b_i=a_0+a_1+…a_{i-1}$)
compacted valid pairs

- $b_i=a_0+a_1+…a_{i-1}$
Goal: For each collided contact pair generated by collision detection, solve the physics constraints and update two bodies’ velocities and positions

1. Solve velocity constraints iteratively
   - Compute relative velocities between bodies
   - Compute impulse from the relative velocities
   - Update velocities using the impulse

2. Solve position constraints iteratively
   - Compute penetration and convert it to impulse
   - Update positions using the impulse

Parallel Solver:
Accelerated pipeline for parallel computation of contact constraints
Box2D-OpenCL Parallel Solver Architecture

Goal: Optimize every component in solver pipeline to get the best parallel performance

1. **Integrate velocities**: Compute all bodies in parallel
   - Apply external forces to update body velocity
2. **Cluster collided contact pairs into different contact groups**
   - All pairs in each group do not share the same body
   - Allows parallel computation in each group
3. **Impulse Initial Guess (Warm Start)**: In parallel for each contact group
   - Last frame’s impulse (if any) is used as an initial guess for current impulse
   - This initial guess accelerates velocity constraint solver
4. **Solve Velocity Constraints** parallel for each contact group
   - Compute impulse to solve contact constraints and update body velocities
5. **Integrate Positions** for all bodies in parallel
   - Use the new computed velocity to update the position for each body
6. **Solve Position Constraints** in parallel for each contact group
   - Compute impulse to update body positions to avoid penetration
Parallel Solver Features

1. **Significant performance improvement** for high speed simulation:
   - 5X performance improvement relative sequential algorithms

2. **Full feature support** includes special joints types:
   - Parallel implementation of different joint types: Distance, Joint/Revolute, Joint/Prismatic, Joint/Pulley, Joint/Gear, joint/Rope, Joint, etc.

3. **Transparency**: Parallelization is transparent to JS app developer
   - App developers do not need to know anything about physics engine parallelization

4. **Easily extensible**:
   - Fluid, Smoke, Fracture, Sand, etc.

5. **CPU/GPU parallelization**:
   - Portable across different platforms
Experimental Results
Performance Results

Core i7 (8 cores), NVIDIA GT 650M (384 shader cores):
Max speedup for Box2DWeb vs. Box2D+Binding & Box2DWeb vs. Box2D-OpenCL+Binding:
• Box2DWeb vs. Binding+Box2D: 8.39X
• Box2DWeb vs. Binding+Box2D-OpenCL: 23.58X

Tizen 2.2.1:
Max speedup for Box2DWeb vs. Binding+Box2D & Box2DWeb vs. Binding+Box2D-OpenCL:
• Box2DWeb vs. Binding+Box2D: 4.75X
• Box2DWeb vs. Binding+Box2D-OpenCL: 14.88X
Box2D-OpenCL Acceleration

- Performance data for rigid body pipeline of OpenCL parallelized Box2D-OpenCL (without binding)
  - Tested on system with Core i7-3770K CPU, Radeon HD 7770 (640 unified shaders) GPGPU

- Sequential performance numbers are for Box2D.
- GPGPU numbers are for Box2D-OpenCL on GPGPU.
- CPU numbers are for Box2D-OpenCL on multicore CPU.
Conclusion
Comprehensive Web Physics Testing

Test Plan

<table>
<thead>
<tr>
<th>#</th>
<th>Test Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Unit Tests for WP JavaScript APIs</td>
<td>Tested for functionality and correctness</td>
</tr>
<tr>
<td>2</td>
<td>Stress Testing</td>
<td>Repetitive construction &amp; destruction of classes. Continuous extended execution</td>
</tr>
<tr>
<td>3</td>
<td>Code Path Coverage Testing</td>
<td>Native and JS applications to test Box2D, Box2D-OpenCL &amp; Web Physics bindings for code coverage</td>
</tr>
<tr>
<td>4</td>
<td>Benchmarking &amp; Performance Analysis</td>
<td>Port of Web Physics bindings to Tizen. Benchmarking &amp; performance analysis</td>
</tr>
<tr>
<td>5</td>
<td>Demo apps for testing</td>
<td>Demo applications for Web Physics (Magic Sands, glBrownian, Touch&amp;Play)</td>
</tr>
<tr>
<td>6</td>
<td>Memory Leak testing</td>
<td>Static &amp; dynamic testing (using Valgrind &amp; developer tool). Box2D &amp; Box2D-OpenCL tested with native &amp; JS demos</td>
</tr>
<tr>
<td>7</td>
<td>Robustness testing</td>
<td>Negative test cases. Testing with invalid and insufficient input.</td>
</tr>
</tbody>
</table>

Comprehensive Testing

- Testing categories:
  - Unit tests
  - Stress tests
  - Code path coverage
  - Benchmarking
  - Demo apps
  - Memory Leak tests
  - Robustness testing
- Tested on multiple platforms
Conclusion

- **Web Physics and Box2D-OpenCL results:**
  - OpenCL accelerated physics engine, with web-based JS interface
  - Box2D-OpenCL: OpenCL accelerated rigid body pipeline. Exposes same API as Box2D
  - JS Physics Engine (Box2DWeb 2.2.1): Soon to be open sourced
  - Web Physics JS bindings & Box2D-OpenCL optimized for Tizen.
  - Box2D-OpenCL open sourced (contributions to Box2D-OpenCL are invited)
    - [https://github.com/Samsung/Box2D-OpenCL](https://github.com/Samsung/Box2D-OpenCL)

- The presenters would like to acknowledge and thank,
  - Simon Gibbs for project guidance
  - Braja Biswal, Sumit Maheshwari, Gajendra N. for contributions to testing & bindings & development of demo applications
  - Linhai Qiu and Chonhyon Park for contributions to parallelization

Thank you!