WebCL Overview and Roadmap

Tasneem Brutch
Chair WebCL Working Group
Samsung Electronics
WebCL Motivation

- Enable high performance parallel processing on GPU/multicore by web applications.
- Portable and efficient access to heterogeneous multicore devices.
- Platform independent, efficient and standards compliant solution.
  - Leverage device’s multicore/parallel computing resources from web-based apps.
- Integration of OpenCL capabilities in the JavaScript environment.
  - Enhanced performance (comparable to native OpenCL).
- Enable a breadth of interactive web applications with high compute demands, on mobile platforms with multicore resources.
  - E.g. apps: Object Recognition, Speech Recognition, Gaming, Augmented Reality, etc.
WebCL Design Goals

• Enable general purpose parallel programming on heterogeneous processing elements, with following design philosophy:
  - A single coherent standard across desktop and mobile devices.
  - Open, royalty-free standard for general purpose parallel programming.
  - Encourage openness with public specification drafts, mailing lists, forums, etc.
    - Public and Khronos mailing lists and Wiki to facilitate discussion.
      - webcl_public@khronos.org
      - https://www.khronos.org/webcl/wiki/Main_Page
  - Design with security as #1 focus.
WebCL HW and SW Requirements

- WebCL will require a modified browser with OpenCL/WebCL support.
- Hardware, driver and runtime support for OpenCL.
- OpenCL program flow and OpenCL platform model:

Creating an OpenCL Program

- Compile
- Create data and arguments
- Send for execution
WebCL Approach

- **Stay close to the OpenCL standard.**
  - To preserve developer familiarity and to facilitate adoption.
  - Allow developers to translate their OpenCL knowledge to web environment.
  - Easier to keep OpenCL and WebCL in sync, as the two evolve.

- **Intended to be an interface above OpenCL.**
  - Facilitate layering of higher level abstractions on top of WebCL API

- **Design with focus on security.**

- **Portable by design.**
  - Simplification of initialization to promote portability.
WebCL Status and Roadmap

Status:
• Khronos BoD approved the creation of WebCL Working Group-May 2011.
• Nokia and Samsung made their prototype implementations open source.
  - WebCL standard is currently being defined by Khronos.

Roadmap:
• An online specification and IDL.
  - A WebCL Reference Card
  - WebCL logo and rules of use
• Implementation document on guidelines for layering WebCL on OpenCL.
• Definition of conformance process.
  - A conformance test suite for testing WebCL implementations
• Security will remain #1 priority for WebCL.
  - WebCL WG will publish security guidelines.
Design with Focus on Security

- **Security is the highest priority for WebCL.**
  - WebCL will be designed for security.
  - Hardening for WebCL will be promoted and enabled.

- **Ensure that new functionality in the browser does not increase its exposure to attacks.**

- **Near Term:**
  - Provisions to promote robustness.

- **Longer Term:**
  - Work with OpenCL device vendors, driver vendors and browser vendors for an in-depth secure solution, and hardware/firmware supported multi-tasking.
Nokia’s WebCL Prototype

• Nokia open sourced their prototype in May 2011 (LGPL).
• Web-based interactive photo editor utilizing GPU for image processing, through WebGL & Nokia’s OpenCL bindings for JavaScript.
• YouTube Demo: http://www.youtube.com/watch?v=9BF7zzUM1kY
• Add-on for Firefox 4 on Win/Linux (Firefox 5 coming soon)
• Visit http://webcl.nokiaresearch.com for binaries, source code, demos and tutorials.
Samsung WebCL Prototype

• Samsung open sourced their prototype WebCL implementation for WebKit in July 2011 (BSD license).

• Allows JavaScript to run computations on GPU.

• Demos on YouTube: [http://www.youtube.com/user/SamsungSISA](http://www.youtube.com/user/SamsungSISA)

  Demos use WebGL for 3D rendering.
  - Computing N-Body gravitational interactions.
  - Computing surface deformations using a fractal noise function.

  - For comparison, same computations were also done in pure JavaScript.
  - WebCL gave performance increases of up to 100x.
  - Test Platform: MacBook Pro with NVIDIA GPGPU.
Samsung WebCL Prototype (cont.)

- **OpenCL to WebCL Mapping:** Samsung’s WebCL prototype implementation attempts to provide for nearly 1:1 mapping with OpenCL.
  - OpenCL APIs are accessed from JavaScript through the WebCLComputeContext class.
  - Mapping OpenCL to WebCL: `cl = new WebCLComputeContext();`

<table>
<thead>
<tr>
<th>OpenCL API</th>
<th>WebCL API</th>
<th>Calling WebCL APIs from JS</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cl_kernel clCreateKernel</code></td>
<td><code>WebCLKernel createKernel</code></td>
<td><code>cl.createKernel()</code></td>
</tr>
<tr>
<td><code>(cl_program program, const char *kernel_name, cl_int *errcode_ret);</code></td>
<td><code>(WebCLProgram*, const String&amp;);</code></td>
<td></td>
</tr>
<tr>
<td><code>cl_mem clCreateBuffer</code></td>
<td><code>WebCLBuffer createBuffer</code></td>
<td><code>cl.createBuffer()</code></td>
</tr>
<tr>
<td><code>(cl_context context, cl_mem_flags flags, size_t size, void *host_ptr, cl_int *errcode_ret);</code></td>
<td><code>(WebCLContext context, cl_mem_flags flags, size_t size, ArrayBuffer host_ptr);</code></td>
<td></td>
</tr>
</tbody>
</table>
Samsung WebCL Prototype Demo

N-Body Simulation:

- Calculates the positions and velocities of N particles and animates them
- Two simulation modes: JavaScript and WebCL
- Two drawing modes: JavaScript and WebGL with 2D/3D rendering option
- For 1024 particles, WebCL gets 20~40x faster simulation time on Mac
  - http://www.youtube.com/user/SamsungSISA#p/a/u/0/F7YSQx2j7o
Samsung WebCL Prototype Demo

Deformation Demo:

- Calculates and renders transparent and reflective deformed spheres on top of photo background
- Performance comparison on Mac
  - JS: ~1 FPS
  - WebCL: 87-116 FPS
- [http://www.youtube.com/user/SamsungSISA#p/a/u/1/9Ttux1A-Nuc](http://www.youtube.com/user/SamsungSISA#p/a/u/1/9Ttux1A-Nuc)
WebCL Resources

• WebCL public distribution list: webcl_public@khronos.org
• WebCL public Wiki: https://www.khronos.org/webcl/wiki/Main_Page
• Nokia’s WebCL prototype resources:
  - Source code and tutorial (http://webcl.nokiaresearch.com/)
  - Demo on YouTube (http://www.youtube.com/watch?v=9BF7zzUM1kY)
• Samsung’s WebCL prototype resources:
  - Demos on YouTube: (http://www.youtube.com/user/SamsungSISA
  - Code available on Google Docs (http://code.google.com/p/webcl/)
• Call for participation in Khronos WebCL Working Group.

For further information on WebCL, contact:
WebCL Chair: Tasneem Brutch (t.brutch@samsung.com)
WebCL Editor: Tomi Aarnio (tomi.aarnio@nokia.com)
Backup
# WebCL Class Hierarchy

<table>
<thead>
<tr>
<th>WebCLComputeContext</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ m_webclObjects</td>
</tr>
<tr>
<td>+ m_webcl_platform</td>
</tr>
<tr>
<td>+ m_webcl_device</td>
</tr>
<tr>
<td>+ m_webcl_program</td>
</tr>
<tr>
<td>+ m_webcl_memObject</td>
</tr>
<tr>
<td>+ m_webcl_sampler</td>
</tr>
<tr>
<td>+ m_webcl_commandQueue</td>
</tr>
</tbody>
</table>

+ getPlatformIDs()  
+ getDeviceIDs()    
+ createContext()   
+ createCommandQueue() 
+ getKernel()       
+ createProgramWithSource() 
+ buildProgram()    
+ getProgramBuildInfo() 
+ createKernel()    
+ createBuffer()    
+ enqueueWriteBuffer() 
+ enqueueReadBuffer() 
+ setKernel()       
+ getKernelWorkInfo() 
+ enqueueNDRangeKernel() 
+ finish()          
+ releaseMemObject() 
+ releaseProgram()  
+ releaseKernel()   
+ releaseCommandQueue() 
+ releaseContext()  

<table>
<thead>
<tr>
<th>WebCLContext</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ cl_context*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WebCLDevice</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ cl_device_id*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WebCLPlatform</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ cl_platform_id*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WebCLProgram</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ cl_program*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WebCLKernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ cl_kernel*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WebCLMemObject</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ cl_mem*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WebCLCommandQueue</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ cl_command_queue*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WebCLEvent</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ cl_event*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WebCLSampler</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ cl_sampler*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WebCLBuffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ cl_buffer*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WebCLImage</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ cl_image*</td>
</tr>
</tbody>
</table>
<html>
<head>
<title>WebCL Hello World</title>
<script>
function getKernel(id) {
    var kernelScript = document.getElementById(id);
}
</script>
<script id="square" type="x-kernel">
__kernel void square(
    __global float* input,
    __global float* output)
{
    int i = get_global_id(0);
    output[i] = input[i] * input[i];
}
</script>
<script>
var cl;
var platform_ids, device_ids;
var context, queue, program, kernel, input, output;
var err;
var data = new Float32Array(DATA_SIZE);
var results = new Float32Array(DATA_SIZE);
</script>
function ExecuteCL() {
    cl = new WebCLComputeContext();
    platform_ids = cl.getPlatformIDs();
    device_ids = cl.getDeviceIDs(platform_ids[0], cl.DEVICE_TYPE_GPU);

    context = cl.createContext(null, device_ids[0], null, null);
    queue = cl.createCommandQueue(context, device_ids[0], null);

    var kernelSource = getKernel("square");
    program = cl.createProgramWithSource(context, kernelSource);

    err = cl.buildProgram(program, null, null, null);
    var info = cl.getProgramBuildInfo(program, device_ids[0], cl.PROGRAM_BUILD_LOG);
    kernel = cl.createKernel(program, "square");

    input = cl.createBuffer(context, cl.MEM_READ_ONLY, Float32Array.BYTES_PER_ELEMENT * count, null);
    output = cl.createBuffer(context, cl.MEM_WRITE_ONLY, Float32Array.BYTES_PER_ELEMENT * count, null);
cl.enqueueWriteBuffer (queue, input, true, 0, Float32Array.BYTES_PER_ELEMENT * count, data, null);
err = cl.setKernelArgGlobal (kernel, 0, input);
err = cl.setKernelArgGlobal (kernel, 1, output);

var local = cl.getKernelWorkGroupInfo (kernel, device_id, cl.KERNEL_WORK_GROUP_SIZE);
cl.enqueueNDRangeKernel (queue, kernel, 1, 0, count, local, null);
cl.finish (queue, null, function(userData) {
    cl.enqueueReadBuffer (queue, output, true, 0, Float32Array.BYTES_PER_ELEMENT * count, results, null);
    cl.releaseMemObject (input);
    cl.releaseMemObject (output);
    cl.releaseProgram (program);
    cl.releaseKernel (kernel);
    cl.releaseCommandQueue (queue);
    cl.releaseContext (context);
});
}