TIZEN™ DEVELOPER CONFERENCE
MAY 7–9, 2012

WebCL for Hardware-Accelerated Web Applications

Won Jeon, Tasneem Brutch, and Simon Gibbs

What is WebCL?

- WebCL is a JavaScript binding to OpenCL.
- WebCL enables significant acceleration of compute-intensive web applications.
- WebCL currently being standardized by Khronos.

Contents

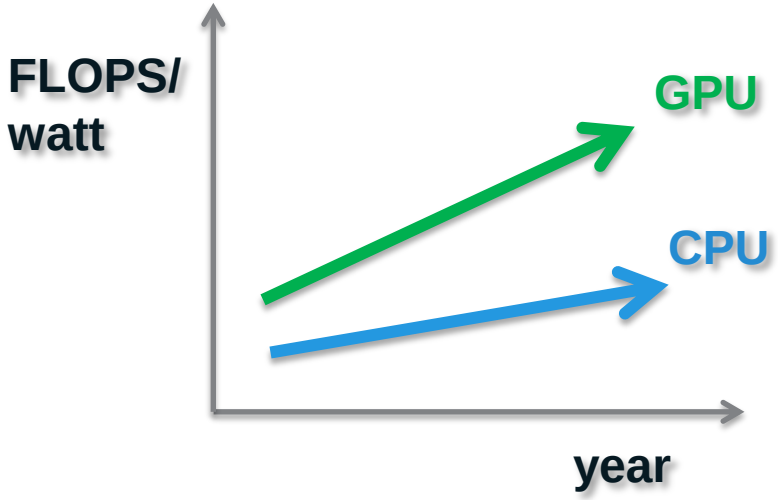
- **Introduction**
  - Motivation and goals
  - OpenCL
- **WebCL**
  - Demos
  - Programming WebCL
  - Samsung's Prototype: implementation & performance
- **Standardization**
  - Khronos WebCL Working Group
  - WebCL robustness and security
  - Standardization status
- **Conclusion**
  - Summary and Resources

Motivation

- New mobile apps with high compute demands:
  - augmented reality
  - video processing
  - computational photography
  - 3D games

Motivation…

- GPU offers improved FLOPS and FLOPS/watt as compared to CPU.



Motivation…

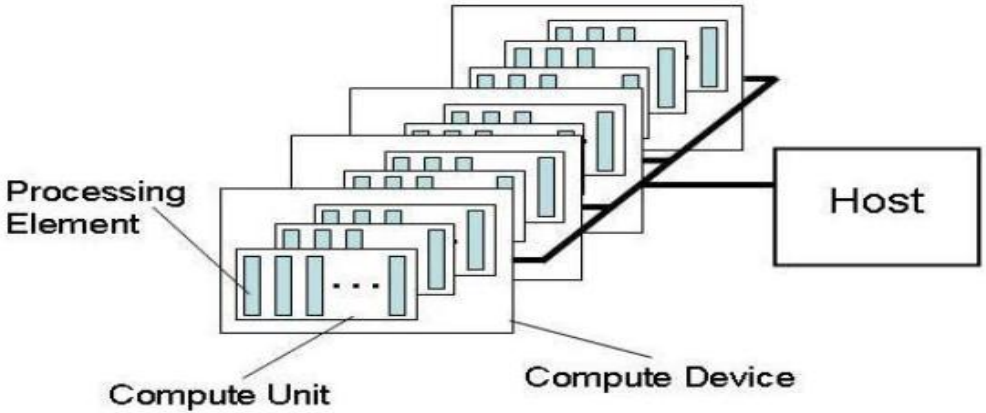
- Web-centric platforms

OpenCL: Overview

- Features
  - C-based cross-platform programming interface
  - Kernels: subset of ISO C99 with language extensions
  - Run-time or build-time compilation of kernels
  - Well-defined numerical accuracy
    (IEEE 754 rounding with specified maximum error)
  - Rich set of built-in functions: cross, dot, sin, cos, pow, log …

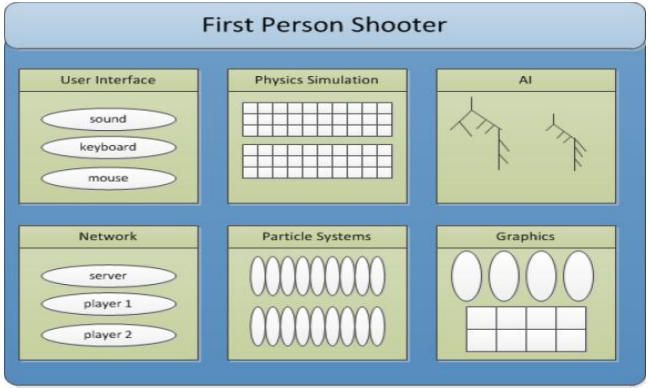
OpenCL: Platform Model

- A host is connected to one or more OpenCL devices
- OpenCL device
  - A collection of one or more ***compute units*** (~ cores)
  - A compute unit is composed of one or more ***processing elements*** (~ threads)
  - Processing elements execute code as SIMD or SPMD



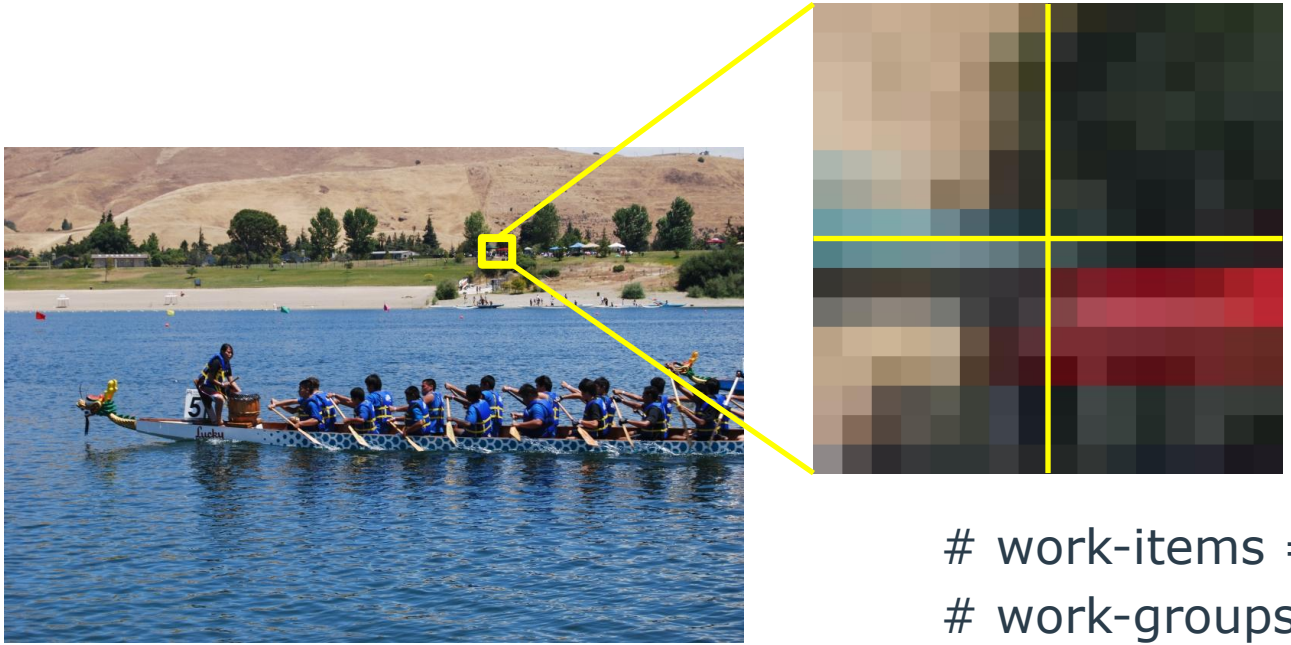
OpenCL: Execution Model

- Kernel
  - Basic unit of executable code, similar to a C function
  - Data-parallel or task-parallel
- Program
  - Collection of kernels and functions called by kernels
  - Analogous to a dynamic library (run-time linking)
- Command Queue
  - Applications queue kernels and data transfers
  - Performed in-order or out-of-order
- Work-item
  - An execution of a kernel by a processing element (~ thread)
- Work-group
  - A collection of related work-items that execute on a single compute unit (~ core)



Different Types of Parallelism

Work-group Example



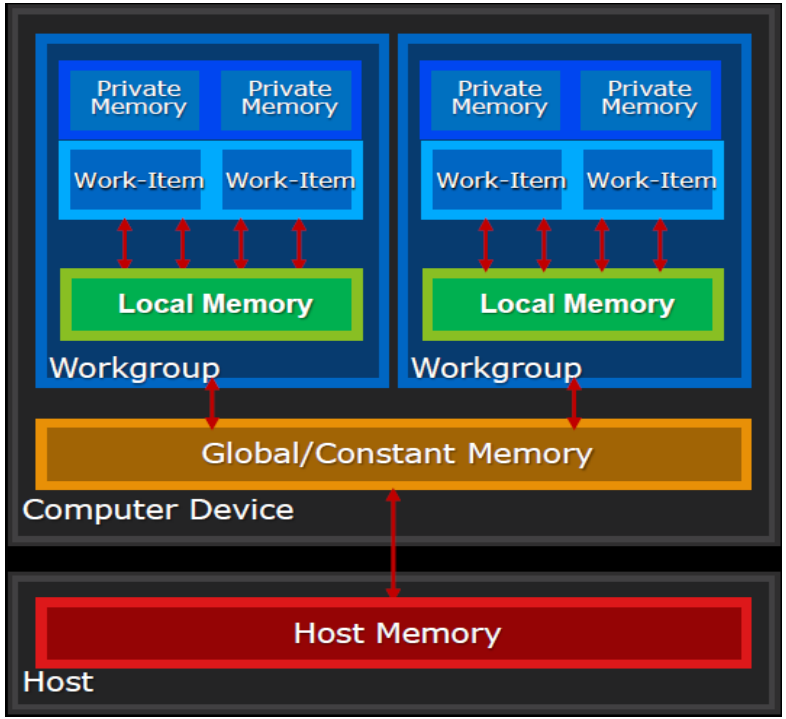
work-items = # pixels

work-groups = # tiles

work-group size = tileW * tileH

OpenCL: Memory Model

- Memory types
  - Private memory (blue)
    - Per work-item
  - Local memory (green)
    - At least 32KB split into blocks, each available to any work-item in a given work-group
  - Global/Constant memory (orange)
    - Not synchronized
  - Host memory (red)
    - On the CPU
- Memory management is explicit
  - Application must move data from host → global → local and back



OpenCL: Kernels

- OpenCL kernel
  - Defined on an N-dimensional computation domain
  - Execute a kernel at each point of the computation domain

Traditional Loop

```
void vectorMult(
     const float* a,
     const float* b,
     float* c,
     const unsigned int count)
{
     for(int i=0; i<count; i++)
         c[i] = a[i] * b[i];
}
```

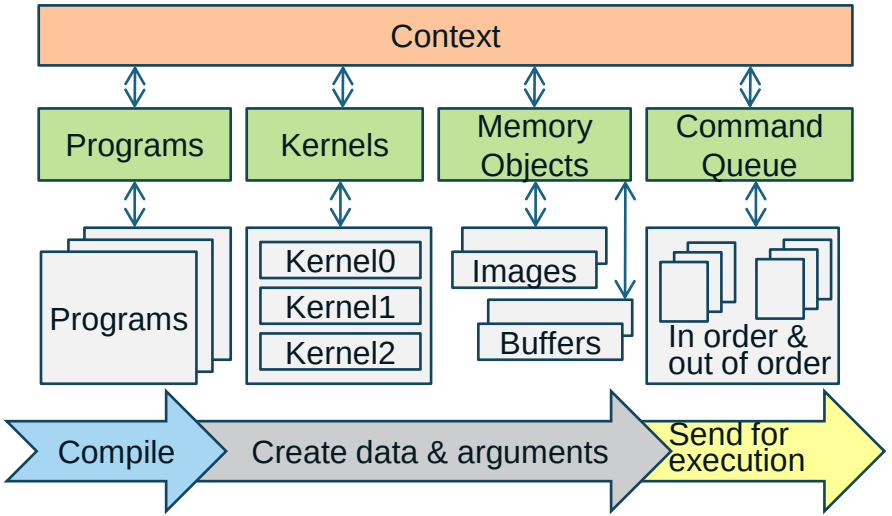
Data Parallel OpenCL

```
kernel void vectorMult(
     global const float* a,
     global const float* b,
     global float* c)
{
     int id = get_global_id(0);

     c[id] = a[id] * b[id];
}
```

OpenCL: Execution of OpenCL Program

1. Query host for OpenCL devices.
2. Create a context to associate OpenCL devices.
3. Create programs for execution on one or more associated devices.
4. From the programs, select kernels to execute.
5. Create memory objects accessible from the host and/or the device.
6. Copy memory data to the device as needed.
7. Provide kernels to the command queue for execution.
8. Copy results from the device to the host.



WebCL Demos

Programming WebCL

- Initialization
- Create kernel
- Run kernel
- WebCL image object creation
  - From Uint8Array
  - From <img>, <canvas>, or <video>
  - From WebGL vertex buffer
  - From WebGL texture
- WebGL vertex animation
- WebGL texture animation

WebCL: Initialization (draft)

```
<script>
    var platforms = WebCL.getPlatforms();

    var devices = platforms[0].getDevices(WebCL.DEVICE_TYPE_GPU);

    var context = WebCL.createContext({ WebCLDevice: devices[0] } );
    var queue = context.createCommandQueue();
</script>
```

WebCL: Create Kernel (draft)

```
<script id="squareProgram" type="x-kernel">
    __kernel square(
        __global float* input,
        __global float* output,
        const unsigned int count)
    {
        int i = get_global_id(0);
        if(i < count)
            output[i] = input[i] * input[i];
    }
</script>

<script>
    var programSource = getProgramSource("squareProgram");   // JavaScript function using DOM APIs
    var program = context.createProgram(programSource);
    program.build();
    var kernel = program.createKernel("square");
</script>
```

OpenCL C

(based on C99)

WebCL: Run Kernel (draft)

```
<script>
    …
    var inputBuf  = context.createBuffer(WebCL.MEM_READ_ONLY,  Float32Array.BYTES_PER_ELEMENT * count);
    var outputBuf = context.createBuffer(WebCL.MEM_WRITE_ONLY, Float32Array.BYTES_PER_ELEMENT * count);

    var data = new Float32Array(count);
    // populate data …
    queue.enqueueWriteBuffer(inputBuf, data, true);          // last arg indicates API is blocking

    kernel.setKernelArg(0, inputBuf);
    kernel.setKernelArg(1, outputBuf);
    kernel.setKernelArg(2, count, WebCL.KERNEL_ARG_INT);

    var workGroupSize = kernel.getWorkGroupInfo(devices[0], WebCL.KERNEL_WORK_GROUP_SIZE);
    queue.enqueueNDRangeKernel(kernel, [count], [workGroupSize]);

    queue.finish();                                          // this API blocks

    queue.enqueueReadBuffer(outputBuf, data, true);          // last arg indicates API is blocking
</script>
```

WebCL: Image Object Creation (draft)

- **From Uint8Array()**

```
<script>
    var bpp = 4;   // bytes per pixel
    var pixels = new Uint8Array(width * height * bpp);
    var pitch = width * bpp;
    var clImage = context.createImage(WebCL.MEM_READ_ONLY,
      {channelOrder:WebCL.RGBA, channelType:WebCL.UNORM_INT8, size:[width, height], pitch:pitch } );
</script>
```

- **From <img> or <canvas> or <video>**

```
<script>
    var canvas = document.getElementById("aCanvas");
    var clImage = context.createImage(WebCL.MEM_READ_ONLY, canvas);     // format, size from element
</script>
```

WebCL: Vertex Animation (draft)

- **Vertex Buffer Initialization**

```
<script>
```

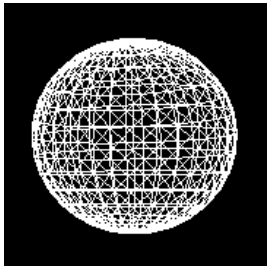
Web GL

```
var points = new Float32Array(NPOINTS * 3);
var glVertexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, glVertexBuffer);
gl.bufferData(gl.ARRAY_BUFFER, points, gl.DYNAMIC_DRAW);
```

Web CL

```
var clVertexBuffer = context.createFromGLBuffer(WebCL.MEM_READ_WRITE, glVertexBuffer);
kernel.setKernelArg(0, NPOINTS, WebCL.KERNEL_ARG_INT);
kernel.setKernelArg(1, clVertexBuffer);
```

```
</script>
```



- **Vertex Buffer Update and Draw**

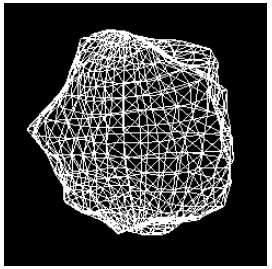
```
<script>
function DrawLoop() {
```

Web CL

```
queue.enqueueAcquireGLObjects([clVertexBuffer]);
queue.enqueueNDRangeKernel(kernel, [NPOINTS], [workGroupSize]);
queue.enqueueReleaseGLObjects([clVertexBuffer]);
```

Web GL

```
gl.bindBuffer(gl.ARRAY_BUFFER, glVertexBuffer);
gl.clear(gl.COLOR_BUFFER_BIT);
gl.drawArrays(gl.POINTS, 0, NPOINTS);
gl.flush();
}
</script>
```



WebCL: Texture Animation (draft)

- **Texture Initialization**

```
<script>
    var glTexture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, glTexture);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
    var clTexture = context.createFromGLTexture2D(WebCL.MEM_READ_WRITE, gl.TEXTURE_2D, glTexture);
    kernel.setKernelArg(0, NWIDTH, WebCL.KERNEL_ARG_INT);
    kernel.setKernelArg(1, NHEIGHT, WebCL.KERNEL_ARG_INT);
    kernel.setKernelArg(2, clTexture);
</script>
```

- **Texture Update and Draw**

```
<script>
function DrawLoop() {
    queue.enqueueAcquireGLObjects([clTexture]);
    queue.enqueueNDRangeKernel(kernel, [NWIDTH, NHEIGHT], [tileWidth, tileHeight]);
    queue.enqueueReleaseGLObjects([clTexture]);
    gl.clear(gl.COLOR_BUFFER_BIT);
    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, glTexture);
    gl.flush();
}
</script>
```

WebCL: Prototype Status

- Prototype implementations
  - Nokia's prototype: uses Firefox, open sourced May 2011 (LGPL)
  - Samsung's prototype: uses WebKit, open sourced June 2011 (BSD)
  - Motorola's prototype: uses node.js, open sourced April 2012 (BSD)
- Functionality
  - WebCL contexts
  - Platform queries (clGetPlatformIDs, clGetDeviceIDs, etc.)
  - Creation of OpenCL Context, Queue, and Buffer objects
  - Kernel building
  - Querying workgroup size (clGetKernelWorkGroupInfo)
  - Reading, writing, and copying OpenCL buffers
  - Setting kernel arguments
  - Scheduling a kernel for execution
  - GL interoperability
    - clCreateFromGLBuffer
    - clEnqueueAcquireGLObjects
    - clEnqueueReleaseGLObjects
  - Synchronization
  - Error handling

WebCL: Implementation

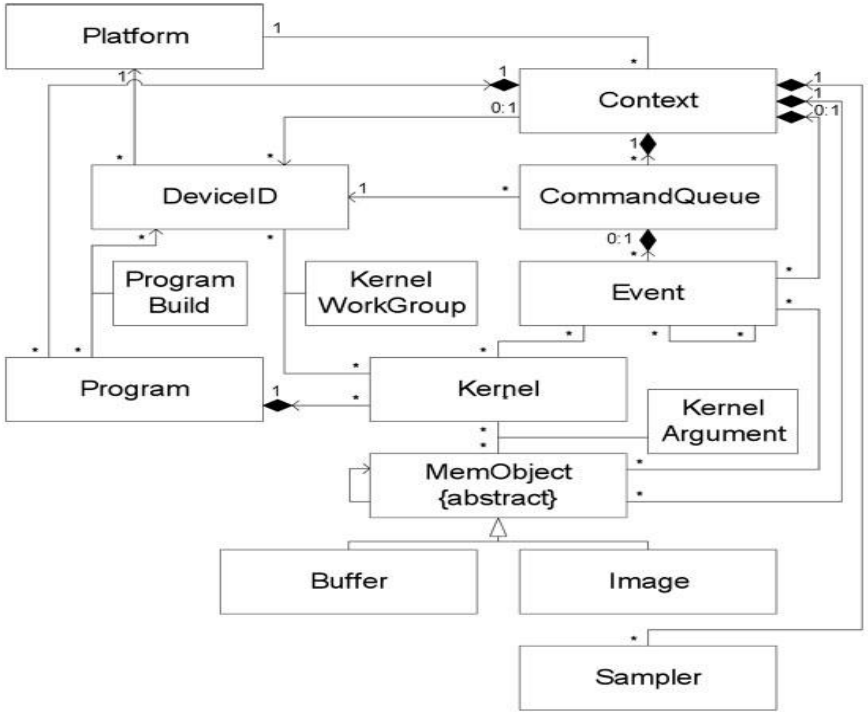
OpenCL UML

Class Relationships

- inheritance
- aggregation
- association
- direction of navigability

Relationship Cardinality

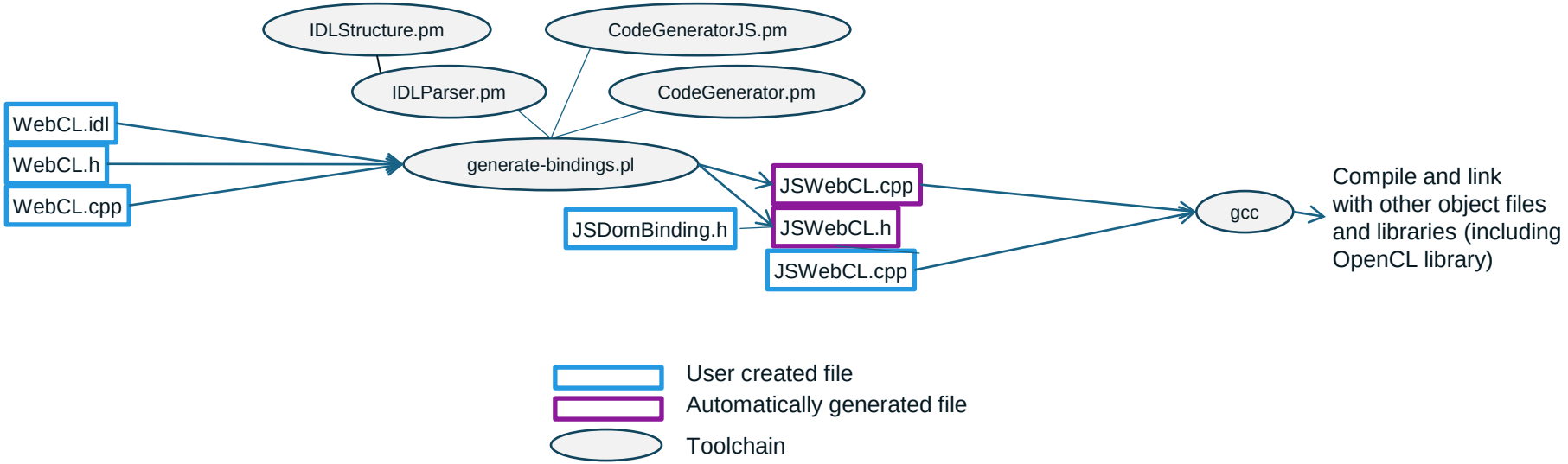
*	many
1	one and only one
0:1	optionally one



Samsung's WebCL Prototype: Implementation on WebKit

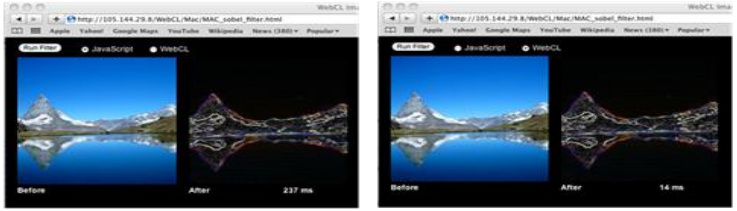
- Modified file
  - Source/WebCore/page/DOMWindow.idl
- List of added files (under Source/WebCore/webcl)
  - WebCL.idl
  - WebCLBuffer.idl
  - WebCLCommandQueue.idl
  - WebCLContext.idl
  - WebCLDevice.idl
  - WebCLDeviceList.idl
  - WebCLEvent.idl
  - WebCLEventList.idl
  - WebCLImage.idl
  - WebCLKernel.idl
  - WebCLKernelList.idl
  - WebCLMem.idl
  - WebCLPlatform.idl
  - WebCLPlatformList.idl
  - WebCLProgram.idl
  - WebCLSampler.idl

Samsung's WebCL Prototype: Building

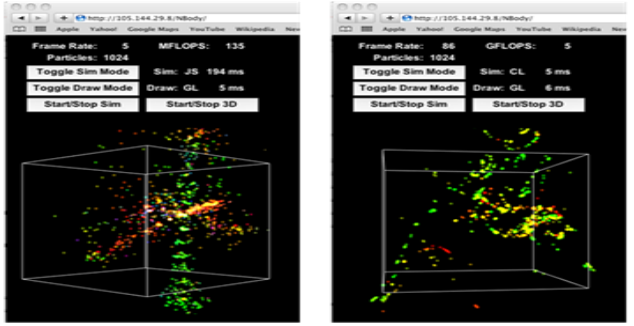


Samsung's WebCL Prototype: Status

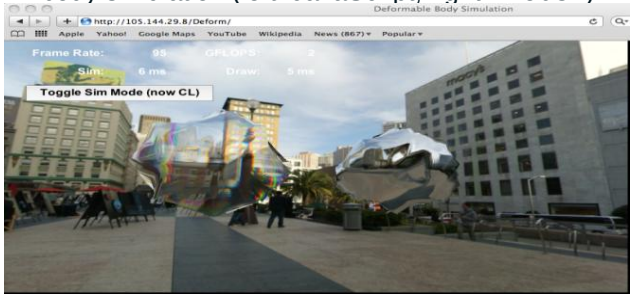
- Integration with WebKit
  - r78407 (released on February 2011)
  - r92365 (released on August 2011)
  - r101696 (released on December 2011)
  - Available at http://code.google.com/p/webcl
- Performance evaluation
  - Sobel filter
  - N-body simulation
  - Deformable body simulation



Sobel filter (left: JavaScript, right: WebCL)



N-body simulation (left: JavaScript, right: WebCL)



Deformation (WebCL)

Samsung's WebCL Prototype: Performance Results

- Tested platform
  - Hardware: MacBook Pro (Intel Core i7@2.66GHz CPU, 8GB memory, Nvidia GeForce GT 330M GPU)
  - Software: Mac OSX 10.6.7, WebKit r78407
  - Video clips available at http://www.youtube.com/user/SamsungSISA

Demo Name	JavaScript	WebCL	Speed-up
Sobel filter (with 256x256 image)	~200 ms	~15ms	**13x**
N-body simulation (1024 particles)	5-6 fps	75-115 fps	**12-23x**
Deformation (2880 vertices)	~ 1 fps	87-116 fps	**87-116x**

Performance comparison of JavaScript vs. WebCL

WebCL Standardization

Tasneem Brutch
Khronos WebCL Working Group Chair

TIZEN™ DEVELOPER CONFERENCE
MAY 7–9, 2012

WebCL Standardization

- Standardize portable and efficient access to heterogeneous multicore devices from web content.
- Define the ECMAScript API for interacting with OpenCL or equivalent computing capabilities.
- Designed for security and robustness.
- Use of OpenCL Embedded Profile.
- Interoperability between WebCL and WebGL.
- WebCL Portability:
  - Proposal by Samsung to simplify initialization and to promote portability.

WebCL Robustness and Security

- Security threats
  - Information leakage.
  - Denial of service from long running kernels.
- WebCL security requirements
  - Prevent cross-origin information leakage
    - Prevent CORS (Cross Origin Resource Sharing) to restrict media use by non-originating web site.
  - Prevent out of range memory access
    - Prevent out of bounds memory writes.
    - Out of bounds memory reads should not return any uninitialized value.
  - Memory initialization
    - All memory objects shall be initialized at allocation time to prevent information leakage.
  - Prevent denial of service
    - Ability to detect offending kernels.
    - Ability to terminate contexts associated with malicious or erroneous kernels.
  - Undefined OpenCL behavior to be defined by WebCL
    - Prevent security vulnerabilities resulting from undefined behavior.
    - Ensure strong conformance testing.
- OpenCL robustness and security
  - Discussion of WebCL security and robustness proposals, currently in progress with OpenCL WG.

WebCL Standardization Status

- WebCL API definition in progress:
  - First working draft released to public through public_webcl@khronos.org - April, 2012.
  - Definition of conformance process and conformance test suite for testing WebCL implementation – 2H, 2012.
  - Release of WebCL specification draft – 2H, 2012.
- Strong interest in WebCL by Khronos members.
- Non-Khronos members engaged through public mailing list.

Summary and Resources

- WebCL is a JavaScript binding to OpenCL, allowing web applications to leverage heterogeneous computing resources.
- WebCL enables significant acceleration of compute-intensive web applications.

WebCL Resources

- Samsung's open source WebCL prototype implementation available at: http://code.google.com/p/webcl
- Khronos member WebCL distribution list: webcl@khronos.org
- Public WebCL distribution list: public_webcl@khronos.org
- WebCL public Wiki: https://www.khronos.org/webcl/wiki/Main_Page
- WebCL public working draft: https://cvs.khronos.org/svn/repos/registry/trunk/public/webcl/spec/latest/index.html

Thank You!

Appendix

WebCL: Differentiation and Impact

- Related technologies
  - **Google's Renderscript**: providing 3D rendering and compute on Android
  - **Intel's River Trail**: JavaScript extension for ParallelArray objects
- Comparison to WebCL
  - WebCL is closer to the HW:
    - Fine-grained control over memory use
    - Fine-grained control over thread scheduling
  - "Interoperability" with WebGL
  - Specialized features: image samplers, synchronization events

OpenCL: Sample (1 of 3)

Ref: http://developer.apple.com/library/mac/#samplecode/OpenCL_Hello_World_Example/Listings/hello_c.html

```
__kernel square(
    __global float* input,
    __global float* output,
    const unsigned int count)
{
    int i = get_global_id(0);
    if(i < count)
            output[i] = input[i] * input[i];
}
```

Notes

1. A kernel is a special function written using a C-like language.
2. Objects in global memory can be accessed by all work items.
3. Objects in const memory are also accessible by all work items.
4. Value of input, output and count are set via clSetKernelArg.
5. Each work item has a unique id.

OpenCL: Sample (2 of 3)

```
int main(int argc, char** argv)
{
    int count = COUNT;              // number of data values
    float data[ COUNT ];            // used by input buffer
    float results[ COUNT ];         // used by output buffer

1   clGetDeviceIDs (NULL, CL_DEVICE_TYPE_GPU, 1, &device_id, NULL);
    context = clCreateContext (0, 1, &device_id, NULL, NULL, &err);

    queue = clCreateCommandQueue (context, device_id, 0, &err);
    program = clCreateProgramWithSource (context, 1, (const char **) & KernelSource, NULL, &err);

2   clBuildProgram (program, 0, NULL, NULL, NULL, NULL);
    kernel = clCreateKernel (program, "square", &err);

    input = clCreateBuffer (context,  CL_MEM_READ_ONLY,  sizeof(data)   , NULL, NULL);
3   output = clCreateBuffer (context, CL_MEM_WRITE_ONLY, sizeof(results), NULL, NULL);
```

Notes

1. Application may request use of CPU (multicore) or GPU.
2. Builds (compiles and links) the kernel source code associated with the program.
3. Creates buffers that are read and written by kernel code.

OpenCL: Sample (3 of 3)

```
1  clEnqueueWriteBuffer (queue, input, CL_TRUE, 0, sizeof(data), data, 0, NULL, NULL);

   clSetKernelArg (kernel, 0, sizeof(cl_mem), &input);
2  clSetKernelArg (kernel, 1, sizeof(cl_mem), &output);
   clSetKernelArg (kernel, 2, sizeof(unsigned int), &count);

   clGetKernelWorkGroupInfo (kernel, device_id, CL_KERNEL_WORK_GROUP_SIZE, sizeof(local), &local, NULL);

   global = count;
3  clEnqueueNDRangeKernel (queue, kernel, 1, NULL, &global, &local, 0, NULL, NULL);

4  clFinish (queue);

5  clEnqueueReadBuffer (queue, output, CL_TRUE, 0, sizeof(results), results, 0, NULL, NULL );

   clReleaseMemObject (input);
   clReleaseMemObject (output);
   clReleaseProgram (program);
   clReleaseKernel (kernel);
   clReleaseCommandQueue (queue);
   clReleaseContext (context);
}
```

Notes

1. Requests transfer of data from host memory to GPU memory.
2. Set arguments used by the kernel function.
3. Requests execution of work items.
4. Blocks until all commands in queue have completed.
5. Requests transfer of data from GPU memory to host memory.

WebCL: Overview

- JavaScript binding for OpenCL
- Provides high performance parallel processing on multicore CPU & GPGPU
  - Portable and efficient access to heterogeneous multicore devices
  - Provides a single coherent standard across desktop and mobile devices
- WebCL HW and SW requirements
  - Need a modified browser with WebCL support
  - Hardware, driver, and runtime support for OpenCL
- WebCL stays close to the OpenCL standard
  - Preserves developer familiarity and facilitates adoption
  - Allows developers to translate their OpenCL knowledge to the web environment
  - Easier to keep OpenCL and WebCL in sync, as the two evolve
- Intended to be an interface above OpenCL
  - Facilitates layering of higher level abstractions on top of WebCL API

Goal of WebCL

- Compute intensive **web applications**
  - High performance
  - Platform independent
  - Standards-compliant
  - Ease of application development

WebCL: Memory Object Creation (draft)

- **From WebGL vertex buffer (ref: OpenGL 9.8.2)**

```
<script>
    var meshVertices = new Float32Array(NVERTICES * 3);
    var meshBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, meshBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, meshVertices, gl.DYNAMIC_DRAW);
    var clBuffer = context.createFromGLBuffer(WebCL.MEM_READ_WRITE, meshBuffer);
</script>
```

- **From WebGL texture (ref: OpenGL 9.8.3)**

```
<script>
    var texture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
    var clImage = context.createFromGLTexture2D(WebCL.MEM_READ_ONLY, gl.TEXTURE_2D, 0, texture);
</script>
```

- **From WebGL render buffer (ref: OpenGL 9.8.4)**

```
<script>
    var renderBuffer = gl.createRenderbuffer();
    var clImage = context.createFromGLRenderBuffer(WebCL.MEM_READ_ONLY, renderBuffer);
</script>
```