WEBCL FOR HARDWARE-ACCELERATED WEB APPLICATIONS

Won Jeon, Tasneem Brutch, and Simon Gibbs
{won.jeon,t.brutch, s.gibbs}@samsung.com
Samsung Information Systems America

Contents

Motivations

- New mobile apps (e.g., Mobile AR, speech processing, and computational photography) have high compute demands.

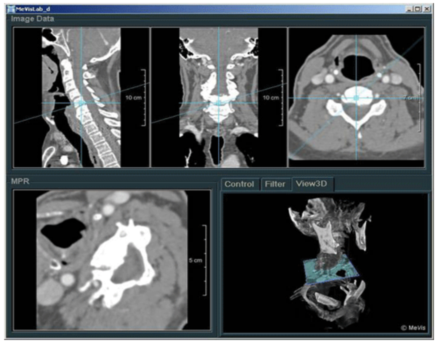
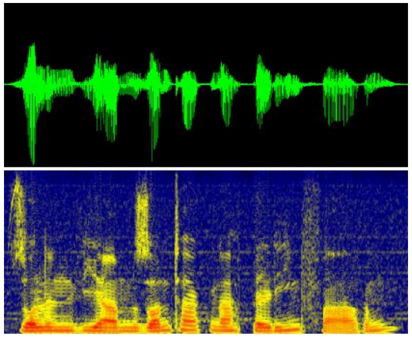


Image Processing



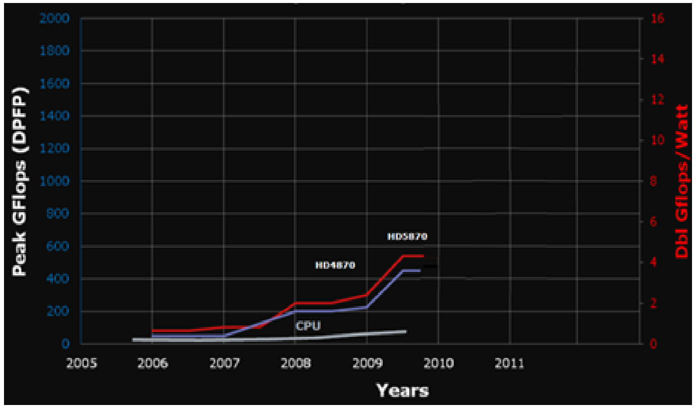
Audio processing

Gaming

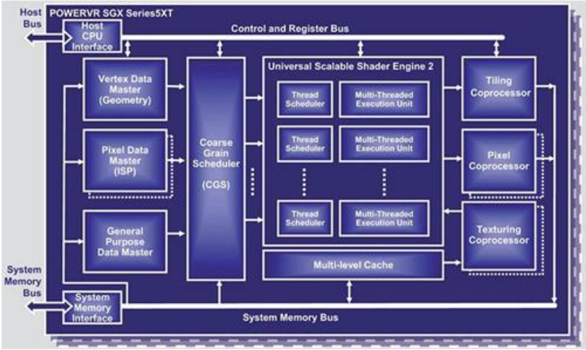
Augmented reality

Motivations

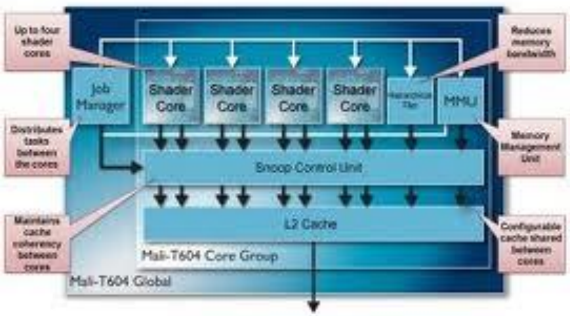
- GPU offers a vastly superior FLOPS and FLOPS/watt as compared to CPU.
- Mobile device roadmaps include multicore processors and general purpose GPUs.



Performance/power for GPU vs. CPU (from AMD)



Imagination Technologies PowerVR 5xx (2010)



ARM Mali T604 (2012/2013)

NVIDIA Tegra Wayne (~2013)

Motivations

- Web-centric platform

Goals

- Make compute intensive web applications possible on the device.
- Promote a platform independent, efficient and standards-compliant solution, for ease of application development and application portability.

Overview

- Enables high performance parallel processing on multicore device by web apps
  - Portable and efficient access to heterogeneous multicore devices
  - Enables a breadth of interactive web apps with high compute demands
  - Provides a single coherent standard across desktop and mobile devices
- JavaScript binding for OpenCL
- Hardware and software requirements
  - Modified browser with OpenCL/WebCL support
  - Hardware, driver, and runtime support for OpenCL
- Stay close to OpenCL standards
  - To preserve developer familiarity and to facilitate adoption
  - Allows 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
  - Facilitates layering of higher level abstractions on top of WebCL API

Current Status

- Prototype implementation
  - Uses WebKit, open sourced in June 2011 (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
  - Synchronization (clFinish)
  - Error handling

OpenCL: Introduction

- OpenCL execution model
  - Define N-dimensional computation domain
  - Execute a kernel at each point in 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

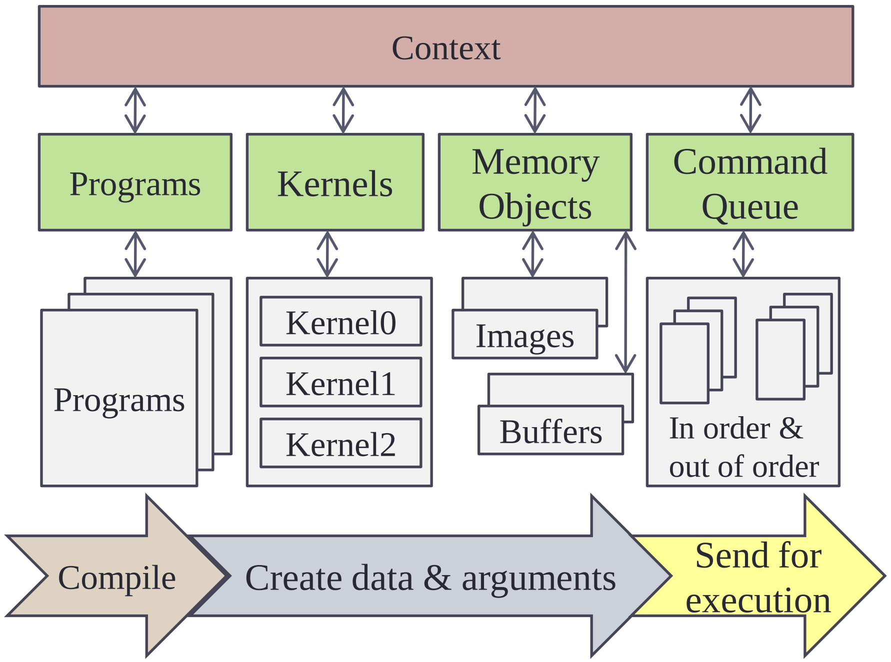
```
kernel void vectorMult(
     global const float* a,
     global const float* b,
     global float* c)
{
     int id = get_global_id(0);

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

OpenCL: Overview

- Language specification
  - C-based cross-platform programming interface
  - Subset of ISO C99 with language extensions
  - Defined numerical accuracy
  - Online or offline compilation and build of compute kernel executables
  - Rich set of built-in functions
- API
  - Platform API
    - A hardware abstraction layer over diverse computational resources
    - Query, select, and initialize compute devices
    - Create compute contexts and work-queues
  - Runtime API
    - Execute compute kernels
    - Manage scheduling, compute, and memory resources

OpenCL: Execution of OpenCL Program

1. Query host for OpenCL device.
2. Create a context to associate OpenCL devices.
3. Create a program for execution on one of more associated devices.
4. From the programs, select kernels to execute.
5. Create memory objects on the host or on 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.



OpenCL: Sample

```
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);

3   input = clCreateBuffer (context,  CL_MEM_READ_ONLY,  sizeof(data)    , NULL, NULL);
    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

```
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);

   clFinish (queue);
4

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

   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.

OpenCL: Sample

```
1 __kernel square(
2     __global float* input,
      __global float* output,   4
3     const unsigned int count)
  {
5     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.

Programming WebCL

- Initialization
- Create kernel
- Run kernel and cleanup
- WebCL memory object creation
- WebGL vertex animation
- WebGL texture animation

Initialization

```
<script>
    var cl = new WebCL ();

    var platforms = cl.getPlatforms();

    var gpu = true;
    var devices = platforms[0].getDeviceIDs(platforms[0], gpu ? cl.DEVICE_TYPE_GPU : cl.DEVICE_TYPE_CPU);

    var context = cl.createContext(null, devices[0], null, null);
    var queue = context.createCommandQueue(devices[0], null);
</script>
```

Create Kernel

```
<script id="square" 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>
function getKernel ( id ) {
    var kernelScript = document.getElementById( id );
    if(kernelScript === null || kernelScript.type !== "x-kernel") return null;
    return kernelScript.firstChild.textContent;
}
</script>

<script>
    var kernelSource = getKernel("square");
    var program = context.createProgram(kernelSource);
    program.buildProgram(null, null, null);
    var kernel = program.createKernel("square");
</script>
```

Run Kernel & Cleanup

```
<script>
    …
    var input  = context.createBuffer(cl.MEM_READ_ONLY,  Float32Array.BYTES_PER_ELEMENT * count, null);
    var output = context.createBuffer(cl.MEM_WRITE_ONLY, Float32Array.BYTES_PER_ELEMENT * count, null);

    queue.enqueueWriteBuffer(input, true, 0, Float32Array.BYTES_PER_ELEMENT * count, data, null);

    kernel.setKernelArg(0, input);
    kernel.setKernelArg(1, output);
    kernel.setKernelArg(2, count, cl.KERNEL_ARG_INT);

    var workGroupSize = kernel.getWorkGroupInfo(device[0], cl.KERNEL_WORK_GROUP_SIZE);
    globalWorkSize[0] = count;
    localWorkSize[0] = workGroupSize;
    queue.enqueueNDRangeKernel(kernel, null, globalWorkSize, localWorkSize);

    queue.finish(GetResults, cl);
}

Function GetResults(cl)
{
       queue.enqueueReadBuffer(output, true, 0, Float32Array.BYTES_PER_ELEMENT*count, results, null);
}
</script>
```

Memory Object Creation

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

```
<script>
    var canvas = document.getElementById("aCanvas");
    var data = canvas.getContext("2d").getImageData();
    var clImage = context.createImageData2D(cl.MEM_READ_ONLY, data);// format, size from element
</script>
```

- **From JavaScript Image()**

```
<script>
    var clImage = null;
    var image = new Image();
    image.src = "lena.jpg";
    image.onload = function() {
       clImage = context.createImageData2D(cl.MEM_READ_ONLY, image);// format, size from element
    };
</script>
```

- **From Uint8Array()**

```
<script>
    var bpp = 4;   // bytes per pixel
    var pixels = new Uint8Array(width * height * bpp);
    var pitch = width * bpp;
    var clImage = context.createImageData2D(cl.MEM_READ_ONLY, cl.RGBA, width, height, pitch, pixels);
</script>
```

Memory Object Creation

- **From WebGL 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(cl.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(cl.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(cl.MEM_READ_ONLY, renderBuffer);
</script>
```

Vertex Animation

- **Vertex Buffer Initialization**

```
<script>
    var points = new Float32Array(NPOINTS * 2);
    var glVertexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, glVertexBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, points, gl.DYNAMIC_DRAW);
    var clVertexBuffer = context.createFromGLBuffer(cl.MEM_READ_WRITE, glVertexBuffer);
    kernel.setKernelArg(0, NPOINTS, cl.KERNEL_ARG_INT);
    kernel.setKernelArg(1, clVertexBuffer);
</script>
```

- **Vertex Buffer Update and Draw**

```
<script>
function DrawLoopStart() {
    queue.enqueueAcquireGLObjects(1, clVertexBuffer, null);
    queue.enqueueNDRangeKernel(kernel, 1, 0, NPOINTS, local, null);
    queue.enqueueReleaseGLObjects(1, clVertexBuffer, null);
    queue.finish(DrawLoopEnd);
}
function DrawLoopEnd(userData) {
    gl.bindBuffer(gl.ARRAY_BUFFER, glVertexBuffer);
    gl.clear(gl.COLOR_BUFFER_BIT);
    gl.drawArrays(gl.POINTS, 0, NPOINTS);
    gl.flush();
}
</script>
```

Texture Animation

- **Texture Initialization**

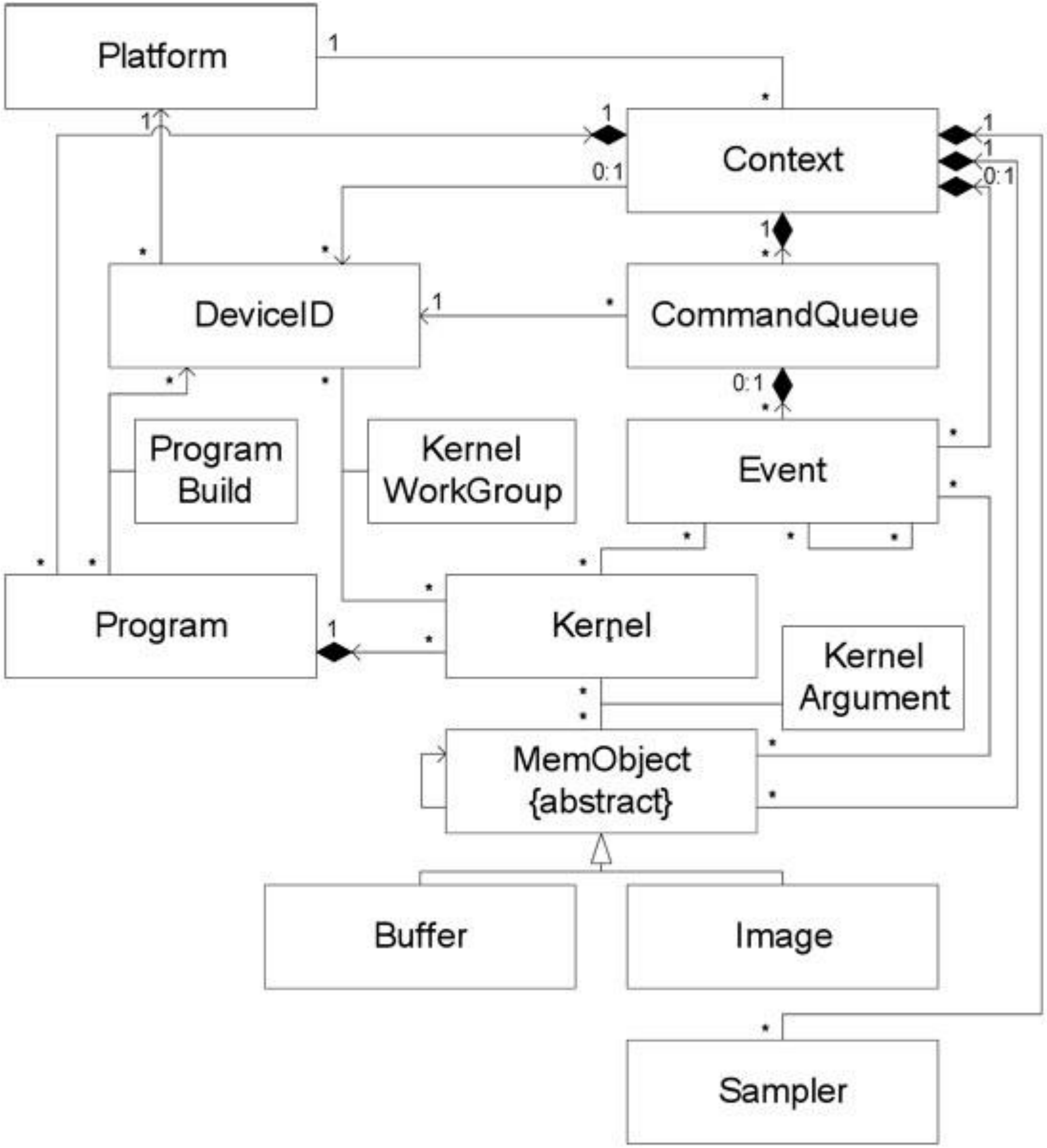
```
<script>
    var glTexture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, glTexture);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
    var clTexture = context.createFromGLTexture2D(cl.MEM_READ_WRITE, gl.TEXTURE_2D, 0, glTexture);
    kernel.setKernelArg(0,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

- **Texture Update and Draw**

```
<script>
function DrawLoopStart() {
    queue.enqueueAcquireGLObjects(1, clTexture, null);
    queue.enqueueNDRangeKernel(kernel, 1, 0, NWIDTH*NHEIGHT, local, null);
    queue.enqueueReleaseGLObjects(1, clTexture, null);
    queue.finish(DrawLoopEnd);
}
function DrawLoopEnd(userData) {
    gl.clear(gl.COLOR_BUFFER_BIT);
    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, glTexture);
    // draw geometry
    gl.flush();
}
</script>
```

Implementation

OpenCL UML



Class Relationships

- inheritance
- aggregation
- association
- direction of navigability

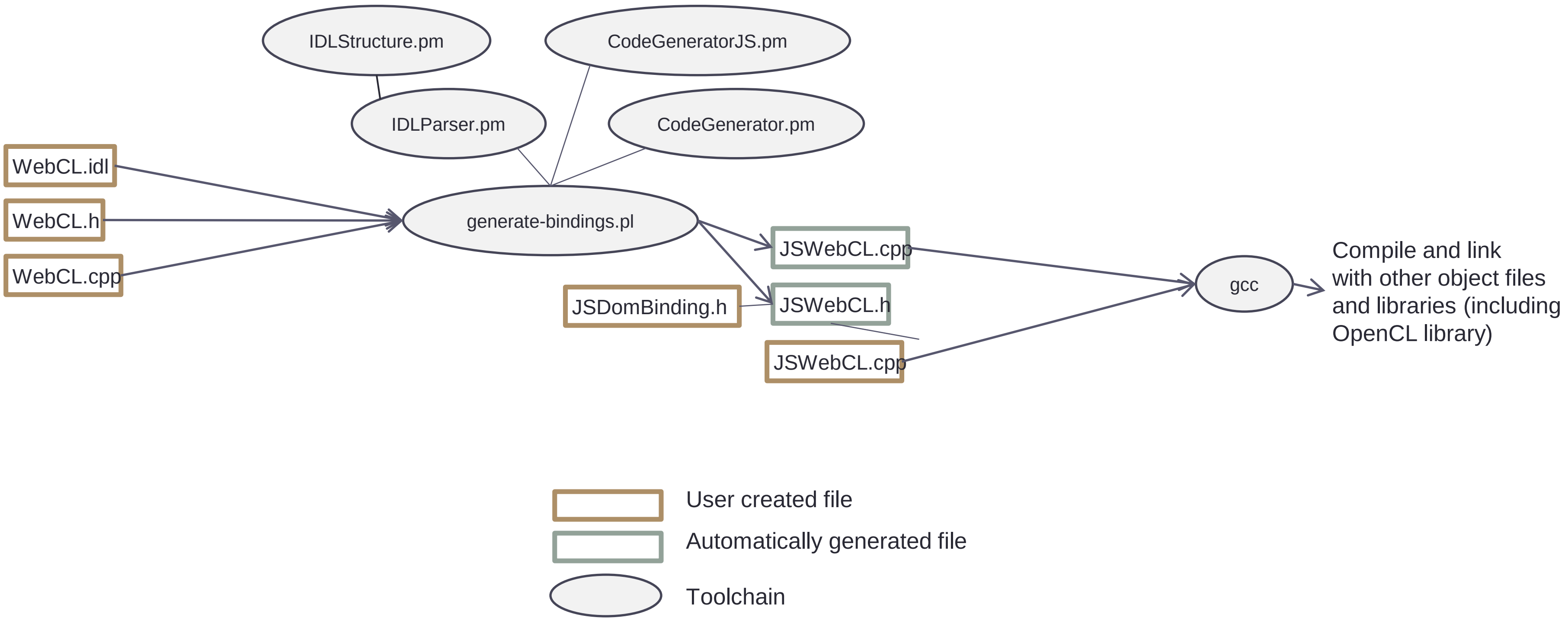
Relationship Cardinality

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

WebCL IDL Files

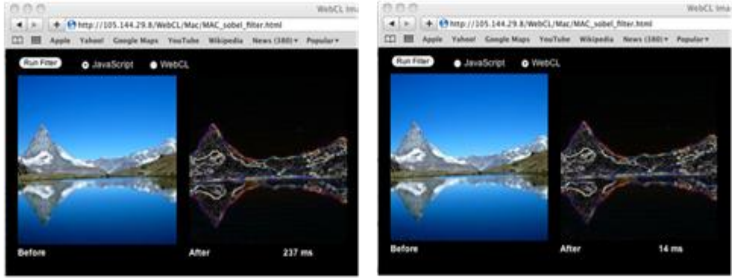
- Modified file
  - Source/WebCore/page/DOMWindow.idl
- List of added files (under Source/WebCore/webcl)
  - WebCL.idl
  - WebCLBuffer.idl
  - WebCLCommandQueue.idl
  - WebCLContext.idl
  - WebCLDevice.idl
  - WebCLDeviceList.idl
  - WebCLEvent.idl
  - WebCLEventList.idl
  - WebCLImage.idl
  - WebCLKernel.idl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

Compilation Steps

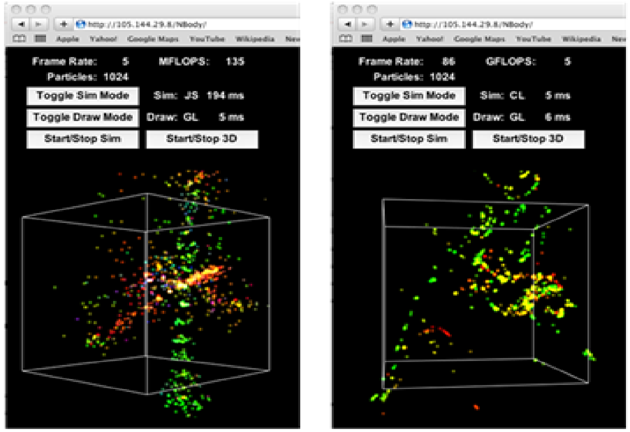


Implementation

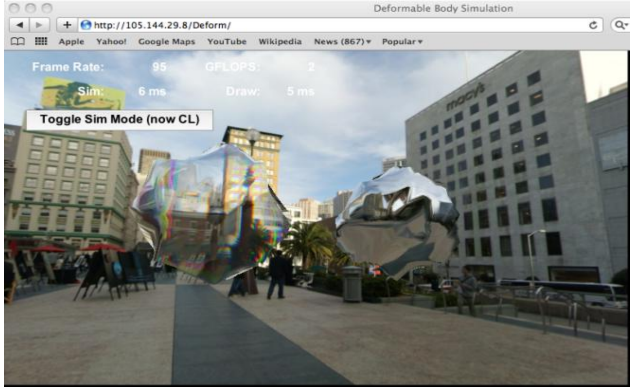
- Integration with WebKit
  - r78407 (released on Feb 10, 2011)
  - r101696 (released on Dec 2, 2011)
  - Available at http://code.google.com/p/webcl
- Performance evaluation
  - Sobel filter
  - N-body simulation
  - Deformable body simulation



Sobel filter (left: JavaScript, right: WebCL)



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



Deformation (WebCL)

Demo

Performance Results

- Tested platform
  - Hardware: MacBook Pro (Intel Core i7@2.66GHz CPU, 8GB memory, Nvidia GeForce GT330M 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

Summary

- 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 open source project is available at http://code.google.com/p/webcl.

Thank you!

- Any questions or comments?
- Contact:
  - Won Jeon, won.jeon@samsung.com
  - Tasneem Brutch, t.brutch@samsung.com
  - Simon Gibbs, s.gibbs@samsung.com