Parallel Web Programming

Tobias Groß, Björn Meier

Hardware/Software Co-Design, University of Erlangen-Nuremberg

May 23, 2013
Outline

WebGL
  OpenGL Rendering Pipeline
  Shader

WebCL
  Motivation
  Development
  Design

A portable javascript thread library for Ajax applications
  Ajax
  Multithread library

Web Workers
  Motivation
  Programing with Web Workers
  Load distribution by using Web Workers for a real-time web application
WebGL

- cross-browser and cross-platform API to create 2D/3D graphics
- based on OpenGL ES 2.0
- uses OpenGL shading language GLSL
- runs in HTML5 canvas elements
- low-level API can be simplified by
  - WebGL libraries (three.js, GLGE, ...)
  - Matrix libraries
ES2.0 Programmable Pipeline

Figure: OpenGL ES 2.0 rendering pipeline[1]
Rendering Pipeline

1. Vertex Shader
2. Triangle assembly
3. Rasterization
4. Fragment Shader

Figure: Example for rendering an object.
Shader

Vertex Shader

• receives a single vertex composed of a series of Vertex Attributes
• produces an output vertex
• 1:1 mapping from input vertices to output vertices

```
<script id="2d-vertex-shader" type="x-shader/x-vertex">
    attribute vec2 a_position;
    void main() {
        gl_Position = vec4(a_position, 0, 1);
    }
</script>
```

Figure: vertex shader

Fragment Shader

processes a Fragment from the rasterization process into a set of colors and a single depth value

```
<script id="2d-fragment-shader" type="x-shader/x-fragment">
    void main() {
        gl_FragColor = vec4(0,1,0,1); // green
    }
</script>
```

Figure: fragment shader
WebGL is gather system

- **fast** for a special purpose
- **lack** of flexibility

**gather read**

The program will write to a fixed position (like the target fragment position of a fragment shader), but has fast access to arbitrary data sources (textures, uniforms, etc.).

**scatter write**

The program receives a stream of input data which it cannot arbitrarily address, but can do fast writes to arbitrary memory locations.
Outline

WebGL
- OpenGL Rendering Pipeline
- Shader

WebCL
- Motivation
- Development
- Design

A portable javascript thread library for Ajax applications
- Ajax
- Multithread library

Web Workers
- Motivation
- Programming with Web Workers
- Load distribution by using Web Workers for a real-time web application
Motivation for WebCL

- increasing number of compute intensive browser applications
  - complex image and audio processing
  - scientific simulations
  - games
  - augmented reality
  - ...
- JavaScript is insufficient
- use of heterogeneous resources

OpenCL kernel vs GLSL shader

- OpenCL has functionality not available in GLSL shaders
- Scattered writes
- Local memory
- Thread synchronization
- Atomic memory operations
Development

Specification
of an JavaScript binding to the Khronos OpenCL standard is still in progress by the Khronos Group.

Implementation prototypes

- Nokia (Firefox extension)
- Samsung (WebKit)
- Motorola (NodeJS)
Design for Samsung implementation[2] (Nokia and Motorola are similar)

- interface above OpenCL (drivers with OpenCL support are required)
- single WebCL object
- Error handling: JavaScript-like exception handling mechanism
- Interoperability with HTML Canvas, Image and Video Elements
- kernel source code must be part of web page
- WebGL interoperability
```javascript
__kernel void 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];
}
```

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

```javascript
var kernelSource = getKernel("square");

var program = context.createProgram(kernelSource);
// Build the program executable
program.build(device);
// Create the compute kernel in the program we wish to run
kernel = program.createKernel("square");

// Set the arguments to our compute kernel
kernel.setArg(0, input);

queue.enqueueNDRangeKernel(kernel, null, globalWorkSize, localWorkSize);
```

Outline

WebGL
- OpenGL Rendering Pipeline
- Shader

WebCL
- Motivation
- Development
- Design

A portable javascript thread library for Ajax applications
- Ajax
- Multithread library

Web Workers
- Motivation
- Programming with Web Workers
- Load distribution by using Web Workers for a real-time web application
Ajax (Asynchronous JavaScript and XML)

- asynchronous web applications
- dynamically changing web pages

**Problem:**
event driven asynchronous programs on a single thread
Multithread library [4][3]

Features:

- JavaScript library
- concurrent programming with threads
- preemptive scheduler
- object-oriented API
- portability
API

Public methods:
- create
- stop
- sleep
- yield
- kill
- Http.get
- Http.post
- ...

```javascript
function load (url) {
  var th = Thread.create(nowLoading);
  try {
    var res = Thread.Http.get(url);
    document.write(res.responseText);
  } catch (e) {
    document.write("ERROR: " + e);
  }
  th.kill();
}

function nowLoading () {
  var bar = ["|", "/", "-", "]
  var i = 0;
  while (true) {
    window.status = "Now loading..." + bar[(i++)%4];
    Thread.sleep(125);
  }
}

Thread.create(load, "http://...");
```

Figure: example for the multithread library[3]
Design

- timesharing the UI-thread for pseudo multithreading
- program break-up into basic blocks
- continuation-based multithreading

```javascript
function f () {
    var c = getc();
    document.write(c, "\n");
}
```

Figure: Original program[7]

```javascript
function f () {
    var c = getc();
    function rest () {
        document.write(c, "\n");
    }
    rest();
}
```

Figure: Rewriting result[7]

- trampolined style

```javascript
do {
    try {
        context = context.continuation.procedure.call(
            context.continuation.this_val,
            context.ret_val
        );
    } catch (e) {
        exception handling
    }
} while (context.timeout == undefined && time slice remains);
```

Figure: trampoline[7]
Outline

WebGL
   OpenGL Rendering Pipeline
   Shader

WebCL
   Motivation
   Development
   Design

A portable javascript thread library for Ajax applications
   Ajax
   Multithread library

Web Workers
   Motivation
   Programing with Web Workers
   Load distribution by using Web Workers for a real-time web application
Motivation

What are Web Workers?

- extension for JavaScript
- available in all modern browsers
- allows to run scripts in the background
- are relatively heavy-weight
  - high start-up performance cost
  - high per-instance memory cost

Why using Web Workers?

- they allow to write client-side code, that utilize threads, which run in background
- this threads don’t block the frontend (keeps the webpage responsive)
- you can relocate computing power from server-side to client-side
Example using Web Workers

```javascript
<script>
    var worker = new Worker('worker.js');
    worker.onmessage = function (event) {
        document.getElementById('result').textContent = event.data;
    }
</script>

Figure: compute prime numbers 1 [6]

```javascript
var n = 1;
search: while (true) {
    n += 1;
    for (var i = 2; i <= Math.sqrt(n); i += 1)
        if (n % i == 0)
            continue search;
    // found a prime!
    postMessage(n);
}

Figure: compute prime numbers 2 [6]
Web Workers API (extract)

Constructors
- `Worker('script.js')`

Communication
- Main thread uses `postMessage` method to communicate with the worker
- Worker handle messages with a `onmessage` function, which must be implemented
- Communication in other direction uses function `postMessage` and a messagehandler as well
The web application

Initial situation

- it’s a web-based multiplayer online role-playing game
- every user operates an avatar
- the avatars interact with other avatars or NPCs
- to interact, avatars have to move around
- the movement is computed depending on forces which affect the avatar

Problem

- Server has to compute forces of all avatars and NPCs
- the latency has to be smaller than 500 ms
- this limits the number of avatars a server can handle at once

Solution

- Load distribution from server to client using Web Workers
Program flow with Web Worker

Figure: System with Web Worker[5]
Evaluation

Figure: Web Worker performance based on browser implementation[5]
Evaluation

Figure: CPU Usage of server[5]
Evaluation

Figure: Data transfer rate[5]
References I

    The standard for embedded accelerated 3d graphics, 2013.
    [Online; accessed 23-Mai-2013].

    Webcl for hardware-accelerated web applications.
    2012.

    Javascript multithread framework for asynchronous processing.
    *IPSJ Transactions on Programming*, 48:1–18.

    A portable javascript thread library for ajax applications.
    In *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*, OOPSLA ’07, pages 817–818, New York, NY, USA, 2007. ACM.
