USING ASM.JS FOR MORE STABLE DSP

By Ethan Geller for Centre National de Création Musicale (GRAME)
July 18, 2013

INTRODUCTION

In almost every practical setting, including standard client-side implementations in browsers, JavaScript is an incredibly dynamic language, continuously being interpreted and/or compiled into cached bytecode using standard JIT compilation practices. However, JIT compilation introduces latencies that are unacceptable for rendering or processing audio. One solution for this is using an ahead of time (AOT) compiler for DSP algorithms. By compiling the intermediate language, in this case javaScript, into machine-optimized code prior to execution rather than continuously caching bytecode during execution, AOT compilers reduce the load of the runtime environment considerably; Compile-time takes a hit in order to reduce runtime.

Mozilla's asm.js is a specification of javaScript that restricts the language to strictly-typed variables, function calls, and pseudo-heap accesses (the heap here being represented as a typed array), and utilizes an AOT compiler. This allows asm.js to operate like lower-level code, and has been used in video game development for web browsers, most notably in the porting of the popular video game engine Unreal Engine 3 to run in a development version on Firefox. asm.js is still early in it's development, but has been estimated to run around half as fast as equivalent native C compiled in clang. Using asm.js can make DSP much more predictable and reduce load during runtime.

RUNNING FAUST OUTPUT THROUGH ASM.JS

The only browser that supports asm.js compilation currently is Mozilla's Firefox. However, producing dynamic audio without plug-ins is currently only possible with the webAudio API, which is not supported in the current build of Firefox. Therefore, the only way to conceivably execute and profile asm.js-optimized DSP scripts is by using Firefox Nightly, the development version of Firefox, which currently has implemented non-webkit standards-based webAudio. Because of this, the Faust js output has also been ported from `webkitAudioContext`-ready code to standards based `audioContext` code. It may be neccesary in future releases of Faust to update the javaScript output to be both webkitAudio and non-webkit audio compatible.

Furthermore, in the current specification of asm.js, an asm.js module has difficulty working with external modules such as the Web Audio API. Therefore, the faust output, which works largely with arithmetic, can be translated to asm.js, but the webAudio implementation is not.

RESULTS

PROFILE `compute`			
Script	*Avg. Call Time (ms)*	*Min. Call Time (ms)*	*Max. Call Time (ms)*
karplusFF.html	**5.107**	**3.307**	**12.729**
karplusFFasmjs.html	**3.672**	**3.469**	**6.15**

javascript profiled using Firebug

In Firefox Nightly, the average call time of the `compute` function in asm.js optimized code was 3.672 ms, compared to an average call time of 5.107ms in the non-asm.js code. Furthermore, asm.js optimized code was significantly less volatile; the call times for the `compute` function experienced a range of 2.681 ms, while the equivalent function's call times in non-asm.js code had a range of 9.422 ms. This is indicative of much more predictable and stable performance in the asm.js, AOT-compiled code compared to the dynamic, JIT compiled non-asm.js code.

CONCLUSION

while Firefox Nightly's implementation of Web Audio and asm.js are both too early in development to be considered stable and predictable, these early results indicate that DSP can be more efficient and stable in runtime using AOT compilation like the one used in asm.js. As Firefox's Web Audio implementation develops and approaches an official release, more reliable tests can be made. In the meantime, I will develop the asm.js architecture for the Faust compiler.

KarplusFF.html

Karplus ported to non-webkit Audio, non-asm.js

```
<html>
<head>
<!-- <script src="jquery-1.7.1.min.js" language="javascript"></script>  -->
<script src="http://code.jquery.com/jquery-1.7.1.min.js" language="javascript"></script>
<script src="file://localhost/usr/local/lib/faust/faustui.js"></script>
<title>
Title
</title>

<!-- Our javascript code -->
<script type="text/javascript">

// init() once the page has finished loading.
window.onload = init;



function karplus() {

	this.fRec0 = new Float32Array(3);
	this.fVec1 = new Float32Array(512);
	this.fRec2 = new Float32Array(2);
	this.fVec0 = new Float32Array(2);
	this.iRec1 = new Int32Array(2);
	this.fhslider0;
	this.fhslider1;
	this.fbutton0;
	this.fhslider2;
	this.IOTA;
	this.fhslider3;
	this.fSamplingFreq;

	this.metadata = function(m) {
		m.declare("author", "Grame");
		m.declare("copyright", "(c)GRAME 2006");
		m.declare("license", "BSD");
		m.declare("math.lib/author", "GRAME");
		m.declare("math.lib/copyright", "GRAME");
		m.declare("math.lib/license", "LGPL with exception");
		m.declare("math.lib/name", "Math Library");
		m.declare("math.lib/version", "1.0");
		m.declare("music.lib/author", "GRAME");
		m.declare("music.lib/copyright", "GRAME");
		m.declare("music.lib/license", "LGPL with exception");
		m.declare("music.lib/name", "Music Library");
		m.declare("music.lib/version", "1.0");
		m.declare("name", "karplus");
		m.declare("version", "1.0");
	}

	this.getNumInputs = function() {
		return 0;
```

```
}
this.getNumOutputs = function() {
	return 1;

}
this.getInputRate = function(channel) {
	var rate;
	switch (channel) {
		default: {
			rate = -1;
			break;
		}

	}
	return rate;

}
this.getOutputRate = function(channel) {
	var rate;
	switch (channel) {
		case 0: {
			rate = 1;
			break;
		}
		default: {
			rate = -1;
			break;
		}

	}
	return rate;

}

this.classInit = function(samplingFreq) {
}

this.instanceInit = function(samplingFreq) {
	this.fSamplingFreq = samplingFreq;
	this.fhslider0 = 0.1;
	this.fhslider1 = 0.5;
	for (var i = 0; (i < 2); i = (i + 1)) {
		this.iRec1[i] = 0;

	}
	this.fbutton0 = 0;
	for (var i = 0; (i < 2); i = (i + 1)) {
		this.fVec0[i] = 0;

	}
	this.fhslider2 = 128;
	for (var i = 0; (i < 2); i = (i + 1)) {
		this.fRec2[i] = 0;

	}
	this.IOTA = 0;
```

```
		for (var i = 0; (i < 512); i = (i + 1)) {
			this.fVec1[i] = 0;

		}
		this.fhslider3 = 128;
		for (var i = 0; (i < 3); i = (i + 1)) {
			this.fRec0[i] = 0;

		}

	}

	this.init = function(samplingFreq) {
		this.classInit(samplingFreq);
		this.instanceInit(samplingFreq);
	}

	this.buildUserInterface = function(ui_interface) {
		ui_interface.openVerticalBox("karplus");
		ui_interface.openVerticalBox("excitator");
		ui_interface.declare("fhslider2", "unit", "f");
		ui_interface.addHorizontalSlider("excitation", function handler(obj) {
function setval(val) { obj.fhslider2 = val; } return setval; }(this), 128, 2, 512, 1);
		ui_interface.addButton("play", function handler(obj) { function setval(val)
{ obj.fbutton0 = val; } return setval; }(this));
		ui_interface.closeBox();
		ui_interface.addHorizontalSlider("level", function handler(obj) { function
setval(val) { obj.fhslider1 = val; } return setval; }(this), 0.5, 0, 1, 0.01);
		ui_interface.openVerticalBox("resonator");
		ui_interface.addHorizontalSlider("attenuation", function handler(obj) {
function setval(val) { obj.fhslider0 = val; } return setval; }(this), 0.1, 0, 1, 0.01);
		ui_interface.declare("fhslider3", "unit", "f");
		ui_interface.addHorizontalSlider("duration", function handler(obj) {
function setval(val) { obj.fhslider3 = val; } return setval; }(this), 128, 2, 512, 1);
		ui_interface.closeBox();
		ui_interface.closeBox();

	}

	this.compute = function(count, inputs, outputs) {
		var output0 = outputs[0];
		var fSlow0 = (0.5 * (1 - this.fhslider0));
		var fSlow1 = (4.65661e-10 * this.fhslider1);
		var fSlow2 = this.fbutton0;
		var fSlow3 = (1 / this.fhslider2);
		var iSlow4 = ((this.fhslider3 - 1.5) & 4095);
		for (var i = 0; (i < count); i = (i + 1)) {
			this.iRec1[0] = (12345 + (1103515245 * this.iRec1[1]));
			this.fVec0[0] = fSlow2;
			this.fRec2[0] = ((this.fRec2[1] + ((fSlow2 - this.fVec0[1]) > 0)) -
(fSlow3 * (this.fRec2[1] > 0)));
			this.fVec1[(this.IOTA & 511)] = ((fSlow0 * (this.fRec0[1] +
this.fRec0[2])) + (fSlow1 * (this.iRec1[0] * (this.fRec2[0] > 0))));
			this.fRec0[0] = this.fVec1[((this.IOTA - iSlow4) & 511)];
			output0[i] = this.fRec0[0];
			this.iRec1[1] = this.iRec1[0];
			this.fVec0[1] = this.fVec0[0];
```

```
                        this.fRec2[1] = this.fRec2[0];
                        this.IOTA = (this.IOTA + 1);
                        this.fRec0[2] = this.fRec0[1];
                        this.fRec0[1] = this.fRec0[0];

                }

        }

}


//webAudio

process_karplus = function(obj)
{
    function process_aux_karplus(event)
    {
        var count;

        /*
        if (event.inputBuffer.numberOfChannels < dsp.getNumInputs()) {
            console.log("Incorrect number of input %d instead of %d",
event.inputBuffer.numberOfChannels, dsp.getNumInputs());
            return;
        }
        */

        if (event.outputBuffer.numberOfChannels < obj.dsp.getNumOutputs()) {
            console.log("Incorrect number of output %d instead of %d",
event.outputBuffer.numberOfChannels, obj.dsp.getNumOutputs());
            return;
        }

        for (var i = 0; i < obj.dsp.getNumInputs(); i++) {
            obj.inputs[i] = event.inputBuffer.getChannelData(i);
            if (obj.inputs[i] != null) {
                count = obj.inputs[i].length;
            }
        }

        for (var i = 0; i < obj.dsp.getNumOutputs(); i++) {
            obj.outputs[i] = event.outputBuffer.getChannelData(i);
            if (obj.outputs[i] != null) {
                count = obj.outputs[i].length;
            }
        }

        obj.dsp.compute(count, obj.inputs, obj.outputs);

    }
    return process_aux_karplus;
}

function create_karplus(audio_context, user_interface, meta_interface, buffer_size)
{
    this.dsp = new karplus();
```

```
    this.dsp.init(audio_context.sampleRate);
    this.dsp.buildUserInterface(user_interface);
    this.dsp.metadata(meta_interface);

    this.inputs = new Array(this.dsp.getNumInputs());
    this.outputs = new Array(this.dsp.getNumOutputs());

    console.log(audio_context.sampleRate);
    console.log(this.dsp.getNumInputs());
    console.log(this.dsp.getNumOutputs());

    this.processor = audio_context.createScriptProcessor(buffer_size,
this.dsp.getNumInputs(), this.dsp.getNumOutputs());
    this.processor.onaudioprocess = process_karplus(this);

    return this.processor;
}

function loadSample(url)
{
    // Load asynchronously

    var request = new XMLHttpRequest();
    request.open("GET", url, true);
    request.responseType = "arraybuffer";

    request.onload = function() {
        source.buffer = context.createBuffer(request.response, false);
        source.loop = true;
        source.noteOn(0);
    }

    request.send();
}

var context;
var ui;
var meta;
var source;
var faustdsp;

function initAudio(buffer_size)
{
    context = new AudioContext();

    meta = new Meta(document.getElementById("FaustMeta"));
    ui = new JUI(document.getElementById("FaustUI"));

    faustdsp = new create_karplus(context, ui, meta, buffer_size);
    faustdsp.connect(context.destination);
}

function init()
{
    initAudio(4096);
}
```

```
function playsound()
{
    var url = $("#sound").val();
    if (source) {
        source.noteOff(0);
        source.disconnect(0);
        source = null;
    }
    source = context.createBufferSource();
    loadSample(url);
    source.connect(faustdsp);
}

function stopsound()
{
    source.noteOff(0);
    source.disconnect(0);
}

</script>
</head>
<body>

<h1><center> Faust process </center></h1>
<center><div id="FaustMeta"></center> </div>
<p>
<center><div id="FaustUI"></center> </div>

<p>
<p>
<center>
<table>
<tr><td class="sound">Sound file:</td> <td> <input type="text" id ="sound" size=20
value="t1.wav"/></td></tr>
</table>
<table>
<tr>
<td><center><button type="button" onclick="playsound()">Play</button></center></td>
<td><center><button type="button" onclick="stopsound()">Stop</button></center></td>
</tr>
</table>
</center>

</body>
</html>
```

KarplusFFasmjs.html

Karplus ported to non-webkit audio and translated for asm.js use

```
<html>
<head>
<!-- <script src="jquery-1.7.1.min.js" language="javascript"></script>  -->
<script src="http://code.jquery.com/jquery-1.7.1.min.js" language="javascript"></script>
<script src="file://localhost/usr/local/lib/faust/faustui.js"></script>
<title>
Title
</title>

<!-- Our javascript code -->
<script type="text/javascript">

// init() once the page has finished loading.
window.onload = init;



function karplus() {
	"use asm";

	this.fRec0 = new Float32Array(3);
	this.fVec1 = new Float32Array(512);
	this.fRec2 = new Float32Array(2);
	this.fVec0 = new Float32Array(2);
	this.iRec1 = new Int32Array(2);
	this.fhslider0;
	this.fhslider1;
	this.fbutton0;
	this.fhslider2;
	this.IOTA;
	this.fhslider3;
	this.fSamplingFreq;

	this.metadata = function(m) {
		m.declare("author", "Grame");
		m.declare("copyright", "(c)GRAME 2006");
		m.declare("license", "BSD");
		m.declare("math.lib/author", "GRAME");
		m.declare("math.lib/copyright", "GRAME");
		m.declare("math.lib/license", "LGPL with exception");
		m.declare("math.lib/name", "Math Library");
		m.declare("math.lib/version", "1.0");
		m.declare("music.lib/author", "GRAME");
		m.declare("music.lib/copyright", "GRAME");
		m.declare("music.lib/license", "LGPL with exception");
		m.declare("music.lib/name", "Music Library");
		m.declare("music.lib/version", "1.0");
		m.declare("name", "karplus");
		m.declare("version", "1.0");
	}

	this.getNumInputs = function() {
```

```
        return 0;

}
this.getNumOutputs = function() {
        return 1;

}
this.getInputRate = function(channel) {
        var rate;
        switch (channel) {
                default: {
                        rate = -1;
                        break;
                }

        }
        return rate;

}
this.getOutputRate = function(channel) {
        var rate;
        switch (channel) {
                case 0: {
                        rate = 1;
                        break;
                }
                default: {
                        rate = -1;
                        break;
                }

        }
        return rate;

}

this.classInit = function(samplingFreq) {
}

this.instanceInit = function(samplingFreq) {
        this.fSamplingFreq = samplingFreq;
        this.fhslider0 = 0.1;
        this.fhslider1 = 0.5;
        for (var i = 0; (i < 2); i = (i + 1)) {
                this.iRec1[i] = 0;

        }
        this.fbutton0 = 0;
        for (var i = 0; (i < 2); i = (i + 1)) {
                this.fVec0[i] = 0;

        }
        this.fhslider2 = 128;
        for (var i = 0; (i < 2); i = (i + 1)) {
                this.fRec2[i] = 0;

        }
```

```
		this.IOTA = 0;
		for (var i = 0; (i < 512); i = (i + 1)) {
			this.fVec1[i] = 0;

		}
		this.fhslider3 = 128;
		for (var i = 0; (i < 3); i = (i + 1)) {
			this.fRec0[i] = 0;

		}

	}

	this.init = function(samplingFreq) {
		this.classInit(samplingFreq);
		this.instanceInit(samplingFreq);
	}

	this.buildUserInterface = function(ui_interface) {
		ui_interface.openVerticalBox("karplus");
		ui_interface.openVerticalBox("excitator");
		ui_interface.declare("fhslider2", "unit", "f");
		ui_interface.addHorizontalSlider("excitation", function handler(obj) {
function setval(val) { obj.fhslider2 = val; } return setval; }(this), 128, 2, 512, 1);
		ui_interface.addButton("play", function handler(obj) { function setval(val)
{ obj.fbutton0 = val; } return setval; }(this));
		ui_interface.closeBox();
		ui_interface.addHorizontalSlider("level", function handler(obj) { function
setval(val) { obj.fhslider1 = val; } return setval; }(this), 0.5, 0, 1, 0.01);
		ui_interface.openVerticalBox("resonator");
		ui_interface.addHorizontalSlider("attenuation", function handler(obj) {
function setval(val) { obj.fhslider0 = val; } return setval; }(this), 0.1, 0, 1, 0.01);
		ui_interface.declare("fhslider3", "unit", "f");
		ui_interface.addHorizontalSlider("duration", function handler(obj) {
function setval(val) { obj.fhslider3 = val; } return setval; }(this), 128, 2, 512, 1);
		ui_interface.closeBox();
		ui_interface.closeBox();

	}

	this.compute = function(count, inputs, outputs) {
		var output0 = outputs[0];
		var fSlow0 = (0.5 * (1 - this.fhslider0));
		var fSlow1 = (4.65661e-10 * this.fhslider1);
		var fSlow2 = this.fbutton0;
		var fSlow3 = (1 / this.fhslider2);
		var iSlow4 = ((this.fhslider3 - 1.5) & 4095);
		for (var i = 0; (i < count); i = (i + 1)) {
			this.iRec1[0] = (12345 + (1103515245 * this.iRec1[1]))|0;
			this.fVec0[0] = fSlow2;
			this.fRec2[0] = ((this.fRec2[1] + ((fSlow2 - this.fVec0[1]) > 0)) -
(fSlow3 * (this.fRec2[1] > 0)));
			this.fVec1[(this.IOTA & 511)] = ((fSlow0 * (this.fRec0[1] +
this.fRec0[2])) + (fSlow1 * (this.iRec1[0] * (this.fRec2[0] > 0))));
			this.fRec0[0] = this.fVec1[((this.IOTA - iSlow4) & 511)|0];
			output0[i] = this.fRec0[0];
			this.iRec1[1] = this.iRec1[0];
```

```
                        this.fVec0[1] = this.fVec0[0];
                        this.fRec2[1] = this.fRec2[0];
                        this.IOTA = (this.IOTA + 1)|0;
                        this.fRec0[2] = this.fRec0[1];
                        this.fRec0[1] = this.fRec0[0];

                }

        }

}
//webAudio

process_karplus = function(obj)
{
    function process_aux_karplus(event)
    {
        var count;

        /*
        if (event.inputBuffer.numberOfChannels < dsp.getNumInputs()) {
            console.log("Incorrect number of input %d instead of %d",
event.inputBuffer.numberOfChannels, dsp.getNumInputs());
            return;
        }
        */

        if (event.outputBuffer.numberOfChannels < obj.dsp.getNumOutputs()) {
            console.log("Incorrect number of output %d instead of %d",
event.outputBuffer.numberOfChannels, obj.dsp.getNumOutputs());
            return;
        }

        for (var i = 0; i < obj.dsp.getNumInputs(); i++) {
            obj.inputs[i] = event.inputBuffer.getChannelData(i);
            if (obj.inputs[i] != null) {
                count = obj.inputs[i].length;
            }
        }

        for (var i = 0; i < obj.dsp.getNumOutputs(); i++) {
            obj.outputs[i] = event.outputBuffer.getChannelData(i);
            if (obj.outputs[i] != null) {
                count = obj.outputs[i].length;
            }
        }

        obj.dsp.compute(count, obj.inputs, obj.outputs);

    }
    return process_aux_karplus;
}

function create_karplus(audio_context, user_interface, meta_interface, buffer_size)
{
    this.dsp = new karplus();
```

```
    this.dsp.init(audio_context.sampleRate);
    this.dsp.buildUserInterface(user_interface);
    this.dsp.metadata(meta_interface);

    this.inputs = new Array(this.dsp.getNumInputs());
    this.outputs = new Array(this.dsp.getNumOutputs());

    console.log(audio_context.sampleRate);
    console.log(this.dsp.getNumInputs());
    console.log(this.dsp.getNumOutputs());

    this.processor = audio_context.createScriptProcessor(buffer_size,
this.dsp.getNumInputs(), this.dsp.getNumOutputs());
    this.processor.onaudioprocess = process_karplus(this);

    return this.processor;
}

function loadSample(url)
{
    // Load asynchronously

    var request = new XMLHttpRequest();
    request.open("GET", url, true);
    request.responseType = "arraybuffer";

    request.onload = function() {
        source.buffer = context.createBuffer(request.response, false);
        source.loop = true;
        source.noteOn(0);
    }

    request.send();
}

var context;
var ui;
var meta;
var source;
var faustdsp;

function initAudio(buffer_size)
{
    context = new AudioContext();

    meta = new Meta(document.getElementById("FaustMeta"));
    ui = new JUI(document.getElementById("FaustUI"));

    faustdsp = new create_karplus(context, ui, meta, buffer_size);
    faustdsp.connect(context.destination);
}

function init()
{
    initAudio(4096);
}
```

```
function playsound()
{
    var url = $("#sound").val();
    if (source) {
        source.noteOff(0);
        source.disconnect(0);
        source = null;
    }
    source = context.createBufferSource();
    loadSample(url);
    source.connect(faustdsp);
}

function stopsound()
{
    source.noteOff(0);
    source.disconnect(0);
}

</script>
</head>
<body>

<h1><center> Faust process </center></h1>
<center><div id="FaustMeta"></center> </div>
<p>
<center><div id="FaustUI"></center> </div>

<p>
<p>
<center>
<table>
<tr><td class="sound">Sound file:</td> <td> <input type="text" id ="sound" size=20
value="t1.wav"/></td></tr>
</table>
<table>
<tr>
<td><center><button type="button" onclick="playsound()">Play</button></center></td>
<td><center><button type="button" onclick="stopsound()">Stop</button></center></td>
</tr>
</table>
</center>

</body>
</html>
```

REFRENCES

"Asm.js." *Asm.js*. Ed. David Herman, Luke Wagner, and Alon Zakai. Mozilla, 17 Mar. 2013. Web. 18 July 2013. <http://asmjs.org/spec/latest/>.

Mikheev, Vitaly. "Improving Swing Performance: JIT vs AOT Compilation | Linux."*Linux*. Linux.sys-con.com, 11 Nov. 2013. Web. 18 July 2013. <http://linux.sys-con.com/node/46901>.

"Porting WebkitAudioContext Code to Standards Based AudioContext." *Mozilla Developer Network*. Mozilla, 8 July 2013. Web. 18 July 2013. <https://developer.mozilla.org/en-US/docs/Web_Audio_API/Porting_webkitAudioContext_code_to_standards_based_AudioContext>.