High Performance Computational Graphics in the Browser

The Graphical Web 2015

Nikos Andronikos
CiSRA
SVG Gradients are boring one-dimensional

http://jsfiddle.net/dodgeyhack/hgpd16am/

http://jsfiddle.net/dodgeyhack/hvmj3bf0/
Mesh Based Gradients in SVG 2

- An array of Coon’s patches
- With smoothing!

Patches inherit top and left sides where possible
Diffusion Curves
Demo!

Me: I’d like to release a demo of our DC renderer
My boss: Maybe you could release a JS version?
Me: hahhahahahahaha

... hmmmmmmmm maybe I’ll try this emscripten thing?
What is Emscripten?

An LLVM to JS compiler

• This means you can compile c and c++ programs as JavaScript to run in the browser

Such as:

• Unreal Engine
• Unity Engine
• Quake3
• Doom
What is Emscripten?

An LLVM to JS compiler

- LLVM bytecode in, asm.js out
Emscripten Performance

1.5 x

What makes emscripten possible?

The level of optimisation in modern JavaScript engines

And..

ArrayBuffer Objects

• Enable representation of a heap
• Strongly typed so enable many optimisations
What makes emscripten possible?

ArrayBuffer Objects

```javascript
var buf = new ArrayBuffer(8);
var int32view = new Uint32Array(buf);

int32view[0] = 0xA5DA724;
int32view[1] = 0xDEFFFFFF;

var int8view = new Uint8Array(buf);

// outputs:
// 24, A7m 5D, 1A, FF, FF, FF, DE on little-endian platforms
// 1A, 5D, A7, 24, DE, FF, FF, FF on big-endian platforms
for (var i = 0; i < int8view.length; i++) {
    console.log(int8view[i]);
}
```

**ArrayBuffer(8)**

<table>
<thead>
<tr>
<th>24</th>
<th>A7</th>
<th>5D</th>
<th>1A</th>
<th>FF</th>
<th>FF</th>
<th>FF</th>
<th>DE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1A5DA724</td>
<td>DEFFFFFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Uint8Array**

**Uint32Array**
Asm.js

A restrictive subset of JavaScript with a defined validation schema that enables extreme optimisation

Comprised of:

• Various numeric types
• Minimal set of operators
• A static heap
• First class functions and function pointers
What makes Asm.js fast?

• runtime type checks not required
• Available operations and types are limited
• Use of ArrayBuffer objects for memory representation
• No objects created that require garbage collection

Ahead of Time compilation can offer more predictable performance
Asm.js
Type is inferred with annotations and coercion.

---

```javascript
function gen_image(width, height) {
    width = width|0;
    height = height|0;

    var y = 0, x = 0, i = 0, c = 0;

    for (y = 0; (y|0) < (height|0); y = ((y|0) + 1)|0) {
        for (x = 0; (x|0) < (width|0); x = ((x|0) + 1)|0) {
            c = ((x|0) % 255) ^ ((y|0) % 255);
            data[i] = c;
            i = ((i|0) + 1)|0;
            data[i] = c;
            i = ((i|0) + 1)|0;
            data[i] = c;
            i = ((i|0) + 1)|0;
            data[i] = 255;
            i = ((i|0) + 1)|0;
        }
    }
}
```
JavaScript Engine Goals

JavaScript engines must efficiently run many varied types of script.

To offer best performance, a JS engine must offer:

- Low latency for scripts executed on load
- Extreme performance for computationally intensive code invoked regularly
Modes of operation

Interpretation

• A virtual machine performs the instructions requested of it

Just in time (JIT) compilation

• After reaching a threshold of use, a portion of code is compiled into a unit of machine code

Ahead of time (AOT) compilation

• Before it is needed (i.e. on load), a portion of code is compiled into a unit of machine code
Basic JS Engine Architecture

- Each level offers progressively better optimisation
- Optimisation is performed at the method level
- Baseline JIT profiles during execution and stores type info
- From observed types, speculative optimisation is performed
- If type checks fail, execution falls back to the previous level
SpiderMonkey (Mozilla’s JS Engine)

- Very similar to the theoretical engine, with the addition of an AOT compiler for asm.js
- asm.js is converted to IONMonkey IR and then optimised and compiled in IONMonkey
- AOT compilation aims to offer predictable performance
V8 (Google’s JS Engine)

- No interpreter!
- Turbofan is currently only partially enabled
  - It is used for asm.js
JavaScriptCore (WebKit’s JS Engine)

- JavaScriptCore has two levels of optimising JIT
- FTL JIT is built upon LLVM
The big question...

Are they fast yet?
What’s to come?

SIMD.js (Single Instruction Multiple Data)

- Proposed for ES7 [https://github.com/tc39/ecmaScript_simd]
- Implemented in FF Nightly

No SIMD 😴

```
1  +  1  =  2
```

With SIMD 😎

```
1  +  1  =  2
3  +  4  =  7
2  +  7  =  9
0  +  3  =  3
```

http://jsfiddle.net/dodgeyhack/q9y1at1p/
What’s to come?

Web Assembly (wasm)

• An evolution of asm.js

• wasm will address the gaps in JS that are difficult to fill without adding a lot of complexity in the language

• Initially a simple translation to asm.js – wasm can be polyfilled

• Uses a binary Abstract Syntax Tree (AST) representation, but ‘view source’ will work

https://www.w3.org/community/webassembly/
What are diffusion curves?

- A type of vector drawing primitive
- The curve itself doesn’t render. It sets the positions of colour constraints for a gradient
- Colour diffuses outwards, and mixes with the colour from other diffusion curves to produce a complex gradient image
SVG Gradients are boring one-dimensional

http://jsfiddle.net/dodgeyhack/hgpd16am/

http://jsfiddle.net/dodgeyhack/hvmj3bf0/
Advanced gradients are multi-dimensional!
Advanced gradients are multi-dimensional!
Advanced gradients are multi-dimensional!
Image rendered from diffusion curve data (© Alexandrina Orzan and Adrien Bousseau) provided by INRIA.
Image rendered from diffusion curve data (© Alexandrina Orzan and Adrien Bousseau) provided by INRIA.
Examples...
SVG Advanced Gradients

Unofficial Proposal Draft, 6 August 2015

Table of Contents

1. Introduction

Advanced gradients are two dimensional images comprised of smooth transitions and sharp discontinuities. There are a number of methods for defining advanced gradients, such as:

- Mesh based gradients
- Diffusion curves

Each method has its own advantages. Mesh based gradients are typically very fast to render, but the markup to describe them is verbose and makes animation difficult. Diffusion curves on the other hand are a compact representation based on the traditional artist’s workflow of sketching outlines and filling with colour.

1. Introduction

ADVANCED GRADIENTS

Example 1

A diffusion curve (e.g., a star) with the curves required to create it (left).
Diffusion Curves in the context of SVG

A real example

```xml
<sxv>
  <defs>
    <dcPatch boundary="inherit"
      boundaryColor="#fcb5a #cfa50 #bb8942 #eca958">
      <!-- long spokes -->
      <dcPath d="M0.5,0.5 L0.9,0.5" colors="#8e5a32 / #fbd765" />
      <dcPath d="M0.5,0.5 L0.6236,0.8804" colors="#c79c4e / #8e5a32" />
      <dcPath d="M0.5,0.5 L0.1763,0.735" color-profile="#fed86d / #643c23,
               #f6c442a / #ac6e3f,
               #4c402b / #ff8719" />
      <!-- short spokes -->
      <dcPath d="M0.5,0.5 0.6236,0.5991" colors="#764d31 / #fed966" />
      <dcPath d="M0.5,0.5 0.4524,0.645" colors="#6b462b / #5e442c" />
      <dcPath d="M0.5,0.5 0.347,0.4997" colors="#bd8a49 / #543822, #bb8444
               #d97544 / #5c3a21" />
      <dcPath d="M0.5,0.5 0.4529,0.3544" color-profile="#c79c4e / #a36a35,
               #723d2c / #653e31" />
      <dcPath d="M0.5,0.5 0.624,0.410" colors="#ff5f5f / #b7e3b, #58a41
               #9b8740 / #637c2d" />
    </dcPatch>
  </defs>
  <!-- filled star -->
  <polygon fill="url(#dc1)" points="0.9,0.5 0.6236,0.5991 0.6236,0.8804
                                   0.4524,0.645 0.1763,0.735 0.347,0.4997 0.1763,0.2548 0.4529,0.354
                                   0.6236,0.1195 0.624,0.410" />
</svg>
```
Diffusion Curves in the context of SVG

A real example

```xml
<svg>
  <defs>
    <dcPatch boundary="inherit"
      boundaryColor="#f0b5a #cfa050 #bb942 #eca958">
      <!-- long spokes -->
      <dcPath d="M0.5,0.5 L0.9,0.5" colors="#8e5a32 / #fbd765" />
      <dcPath d="M0.5,0.5 L0.6236,0.8804" colors="#c79c4e / #8e5a32" />
      <dcPath d="M0.5,0.5 L0.1763,0.735" color-profile="#fed86d / #64c323, #5c442a / #ac6e3f, #4b2c1b / #c99a63" />
      <dcPath d="M0.5,0.5 L0.1763,0.2648" color-profile="#5c442a / #ac6e3f, #4b2c1b / #c99a63" />
      <dcPath d="M0.5,0.5 L0.6236,0.1195" colors="#5c442a / #fd7a3b" />
      <!-- short spokes -->
      <dcPath d="M0.5,0.5 0.6236,0.5901" colors="#76a431 / #fed966" />
      <dcPath d="M0.5,0.5 0.4524,0.645" colors="#5b462b / #6e442c" />
      <dcPath d="M0.5,0.5 0.347,0.4997" colors="#bd8a49 / #543822, #bb8444" />
      <dcPath d="M0.5,0.5 0.4529,0.3544" color-profile="#c79c4e / #a36a35, #c99a63" />
      <dcPath d="M0.5,0.5 0.624,0.410" colors="#ff4c2f / #b77e3b, #c68a41" />
    </dcPatch>
  </defs>
  <!-- filled star -->
  <polygon fill="url(#dc1)" points="0.9,0.5 0.6236,0.5901 0.6236,0.8804 0.4524,0.645 0.1763,0.735 0.347,0.4997 0.1763,0.2648 0.4529,0.3544 0.6236,0.1195 0.624,0.410" />
</svg>
```
Diffusion Curves in the context of SVG

A real example
Diffusion Curves in the context of SVG

A real example

```xml
<svg>
  <defs>
    <dcPatch boundary="inherit"
      boundaryColor="#fddb5a #cfa050 #bb8942 #eca958">
      <!-- long spokes -->
      <dcPath d="M0.5,0.5 L0.9,0.5" colors="#8e5a32 / #fbd755" />
      <dcPath d="M0.5,0.5 L0.6236,0.8804" colors="#c79c4e / #8e5a32" />
      <dcPath d="M0.5,0.5 L0.1763,0.735" color-profile="#fed8ed / #643c23,
                #6c442a / #ac6e3f,
                #5c402b / #ff159, #c79c4e" />
      <!-- short spokes -->
      <dcPath d="M0.5,0.5 0.6236,0.5901" colors="#7e4d31 / #fed966" />
      <dcPath d="M0.5,0.5 0.4524,0.645" colors="#6b462b / #6e442c" />
      <dcPath d="M0.5,0.5 0.347,0.4997" colors="#bb8a44, #543822, #bb8444" />
      <dcPath d="M0.5,0.5 0.4529,0.3544" colors="#c79c4e / #a36a35, #"
                <dcPath d="M0.5,0.5 0.624,0.410" colors="#ff25f / #b77e3b, #c68a41" />
    </dcPatch>
  </defs>
  <!-- filled star -->
  <polygon fill="url(#dc1)"
    points="0.9,0.5 0.6236,0.5901 0.6236,0.8804
    0.4524,0.645 0.1763,0.735 0.347,0.4997 0.1763,0.2548 0.4529,0.3544
    0.6236,0.1195 0.624,0.410" />
</svg>
```
Diffusion Curves in the context of SVG

A real example
A fast and accurate DC renderer

- To make DC viable in SVG we have implemented a multi-grid based diffusion curves renderer
  - Multi-grid is an approach which has been used previously, but never perfected

- Generally, a multi-grid renderer is:
  - Accurate at the constraints. Approximate (but close) in between.

- Our renderer:
  - Doesn’t exhibit the issues typically seen in a mg renderer (leakage, etc)
    - Constraints retain relationship with curve data, allowing the constraints to be determined from the curve data, not the multi-grid image pixel values
  - Is fast
    - 10x faster than comparable fastest prior art method
A fast and accurate DC renderer

Start

Retrieve diffusion curve image description

Generate constraint pixel image

Downsample with constraint thickening

Upsample with Laplacian smoothing

End
A fast and accurate DC renderer

Start

Retrieve diffusion curve image description

Generate constraint pixel image

Downsample with constraint thickening

Upsample with Laplacian smoothing

End

512 x 512
A fast and accurate DC renderer

Leakage
A fast and accurate DC renderer

Start

Retrieve diffusion curve image description

Generate constraint pixel image

Downsample with constraint thickening

Upsample with Laplacian smoothing

End

256 x 256
A fast and accurate DC renderer

128 x 128
A fast and accurate DC renderer

Start

Retrieve diffusion curve image description

Generate constraint pixel image

Downsample with constraint thickening

Upsample with Laplacian smoothing

End

64 x 64
A fast and accurate DC renderer

Start

Retrieve diffusion curve image description

Generate constraint pixel image

Downsample with constraint thickening

Upsample with Laplacian smoothing

End

32 x 32
A fast and accurate DC renderer
A fast and accurate DC renderer
A fast and accurate DC renderer

1. Start
2. Retrieve diffusion curve image description
3. Generate constraint pixel image
4. Downsample with constraint thickening
5. Upsample with Laplacian smoothing
6. End
A fast and accurate DC renderer
A fast and accurate DC renderer

Start

Retrieve diffusion curve image description

Generate constraint pixel image

Downsample with constraint thickening

Upsample with Laplacian smoothing

End
A fast and accurate DC renderer

Start

Retrieve diffusion curve image description

Generate constraint pixel image

Downsample with constraint thickening

Upsample with Laplacian smoothing

End
A fast and accurate DC renderer

1. Start
2. Retrieve diffusion curve image description
3. Generate constraint pixel image
4. Downsample with constraint thickening
5. Upsample with Laplacian smoothing
6. End
A fast and accurate DC renderer

Start

Retrieve diffusion curve image description

Generate constraint pixel image

Downsample with constraint thickening

Upsample with Laplacian smoothing

End
A fast and accurate DC renderer

Start

Retrieve diffusion curve image description

Generate constraint pixel image

Downsample with constraint thickening

Upsample with Laplacian smoothing

End
Finishing up

- Questions?

- Thankyou for having me!

Contact:
- nikos.andronikos@cisra.canon.com.au
- @nandronikos