The future of gradients in SVG?

The Graphical Web 2014

Nikos Andronikos
CiSRA
The future of gradients in SVG?

This presentation is about my vision for the future of gradients in SVG.

Complex Gradients can give a whole new look to images and the possibility of re-creating photo realistic images in vector graphics.

- **Mesh gradients**
  - Fast to render
  - Can require many control points

- **Diffusion curves**
  - The most flexible way of defining a gradient that I know of
  - Easy for designers to create and to edit
  - Easy to animate
  - Scale independent
What are diffusion curves?

Introduction:

- Last year I gave a more general talk on diffusion curves

- This year I’ll talk about:
  - A model for including diffusion curves in SVG
  - A fast and accurate renderer for diffusion curves
  - How diffusion curves and mesh gradients can both fit into SVG
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
Examples...
More examples...

- During my talk, I showed many more renderings of diffusion curve images.

- I would like to thank INRIA for allowing me to use their XML diffusion curve data which is © Alexandrina Orzan and Adrien Bousseau.

- The diffusion curve data is available from: http://maverick.inria.fr/Publications/2008/OBWBTS08/index.php
More examples...

What can you do with diffusion curves that is difficult to do without?
More examples...

Lighting
More examples...
What do I want out of this talk?

- Ideas
- Inspiration
- How will you use DC?
  - Gather requirements for the model

There’s two problems to solve here:

- **Technical.** That’s up to the experts to provide a rendering solution that can be implemented in browsers.

- **Authoring control.** How do we make diffusion curves most useful for the authors? That’s where I’d especially welcome input from the attendees of this conference.
Collecting test images

- Sharing DC test images
  - We hope to set up a pool of diffusion curve test images
    - Use existing W3C license terms
      - 3-clause BSD license and/or W3C test suite license
      - Royalty free for test and research purposes
      - no commercial use in product
  - Canon have a lot of material that could be contributed
  - We will invite other members of the community to also contribute

- The aim is to:
  - Illustrate potential of the technology
    - Visually
    - Compactness of the representation
  - Benchmark renderers
  - Compliance testing
  - Accelerate progress in the field
A proposal for Diffusion Curves in SVG ...
Diffusion Curves in the context of SVG

Not approved!
The following is not approved by the SVG working group yet.

It is merely my work on a proposal
This is something I’ve been working on to get discussion started in the community

In progress:
  - SVG element syntax

To do:
  - Document more clearly, the model behind the syntax

I plan to publish this work in progress proposal after the conference on the SVG WG wiki
Diffusion Curves in the context of SVG

The basics of the model I wish to propose

- Diffusion curve gradient patches define a fill
  - Used just like any other fill
  - This separates the gradient from the geometry (unlike an SVG mesh based gradient)

- A diffusion curve gradient patch is:
  - A boundary curve
    - Pixels outside this curve are not touched by the diffusion process
    - A boundary is useful as it helps achieve consistent results across different rendering models. Therefore, it is a requirement of the specification that a boundary be present, not a requirement of the model.
  - Zero or more inner diffusion curves
Diffusion Curves in the context of SVG

Defining a patch in SVG

```xml
<svg viewBox="0 0 100 100">
  <defs>
    <dcPatch id="dc1" boundary-colors="white">
      <rect x="0" y="0" width="100" height="100" fill="url(#dc1)" />
    </dcPatch>
  </defs>
</svg>
```

- The default user co-ordinate space for the patch is ObjectBoundingBox
- The default boundary for the dcPatch is a rectangle \{x=0; y=0; width=1; height=1\}
Diffusion Curves in the context of SVG

Defining the boundary

- The boundary is a diffusion curve itself and has colour constraints

```xml
<svg viewBox="0 0 100 100">
  <defs>
    <dcPatch id="dc1" boundary-colors="white red blue pink"
             boundary="0 0 1 1"
             patchUnits="objectBoundingBox">
    </dcPatch>
  </defs>
  <rect x="0" y="0" width="100" height="100" fill="url(#dc1)" />
</svg>
```

- This is the colour on the inside edge of the boundary. The colour on the outside is independent of the colour of the DC patch.
Diffusion Curves in the context of SVG

Settings colours (in general)

- Colours are specified with ‘stops’

- Multiple ‘stops’ can be specified on a path
  - Each stop specifies a colour constraint (L/R or both) at a particular position on the path
  - In between the stops, colours are interpolated smoothly
  - If a stop is “doubled-up” – two colour (pairs) provided – there is a sudden transition along the path at that point
Diffusion Curves in the context of SVG

Defining the diffusion curves

- The dcPath element defines a diffusion curve within a patch

```xml
<svg viewBox="0 0 100 100">
  <defs>
    <dcPatch id="dc1" boundary-colors="black">
      <dcPath d="M0,0 L1,1" color-profile="cyan" />
    </dcPatch>
  </defs>
  <rect x="0" y="0" width="100" height="100" fill="url(#dc1)" />
</svg>
```

- The ‘d’ attribute follows standard SVG path syntax

- color-profile uses a command type syntax to:
  - Add constraint stops at positions along the path
  - Set colour on left, right, or both sides of the path at each constraint
  - Similar to the proposed variable width stroke syntax
Diffusion Curves in the context of SVG

Demo!
Diffusion Curves in the context of SVG

Defining the diffusion curves

- color-profile grammar

\[
\text{color-profile} = [\langle \text{path-pos}\rangle? \langle \text{dc-constraint-set}\rangle]\#
\]

\[
\text{dc-constraint-set} = [\ [L \langle \text{color}\rangle \langle \text{color}\rangle?] \ | \ [R \langle \text{color}\rangle \langle \text{color}\rangle?]]
\]

\[
\text{path-pos} = [\text{sub} \langle \text{number}\rangle]? [\text{seg} \langle \text{number}\rangle]? \langle \text{offset}\rangle?
\]

\[
\text{offset} = \langle \text{number}\rangle \ | \ \langle \text{percentage}\rangle
\]
Diffusion Curves in the context of SVG

Defining the diffusion curves

- The dcStop element could co-exist with the color-profile attribute

```
<defs>
  <dcPatch id="dc1" boundary-colors="#fdcb5a">
    <dcPath d="M0.5,0.5 L0.9,0.5"
      color-profile="#8e5a32, seg 1 #fbd765, seg 2 #bb8942" />
  </dcPatch>

  <dcPatch>
    <dcPath d="M0.5,0.5 L0.9,0.5">
      <dcStop segment="0" colors="#8e5a32" />
      <dcStop segment="1" colors="#fbd765" />
      <dcStop segment="2" colors="#bb8942" />
    </dcPath>
  </dcPatch>
</defs>
```
Diffusion Curves in the context of SVG

‘inherit’ boundary type

```
<svg viewBox="0 0 100 100">
  <defs>
    <dcPatch id="dc1" boundary-colors="white red blue pink"
              boundary="inherit">
    </dcPatch>
  </defs>
  <rect x="0" y="0" width="100" height="100" fill="url(#dc1)" />
</svg>
```

- The shape being filled is translated into the patch coordinate system. The shape being filled then becomes the boundary.
- Any defined boundary constraints are mapped onto the new boundary.
- Also results in a rectangular bitmap being produced, but no pixels outside of the filled shape are touched.
Diffusion Curves in the context of SVG

Defining the boundary

- An example of the ‘inherit’ value for boundary

```xml
<svg>
  <defs>
    <dcPath boundary="inherit"
      boundaryColor="#fcb5a #cfa050 #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 / #f9e5a32" />
      <dcPath d="M0.5,0.5 L0.1763,0.735" color-profile="#fed86d / #64c233, #4cc4ad" />
      <dcPath d="M0.5,0.5 L0.1763,0.2648" color-profile="#6c442a / #ac6e3f, #4cc4ad" />
      <dcPath d="M0.5,0.5 L0.6236,0.1195" colors="#5c402b / #ffd159, #c79c4e" />
      <!-- short spokes -->
      <dcPath d="M0.5,0.5 0.6236,0.5901" colors="#764d31 / #fed966" />
      <dcPath d="M0.5,0.5 0.4524,0.645" colors="#eb4f62 / #5e442c" />
      <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="#c794e / a36a35, #764d31" />
      <dcPath d="M0.5,0.5 0.624,0.410" colors="#f2c5f / #b7e3b, #4c84a1" />
    </dcPath>
  </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

Defining the boundary – inherit
Diffusion Curves in the context of SVG

Defining the boundary – inherit

- Without ‘inherit’, a rectangular boundary is used and the following patch is generated

Note: Boundary colours have been set to red, green, blue, purple to highlight the effect
Diffusion Curves in the context of SVG

Defining the boundary – inherit

- When boundary='inherit' the colour constraints are mapped onto the boundary of the shape being filled.

Note: Boundary colours have been set to red, green, blue, purple to highlight the effect
Diffusion Curves in the context of SVG

Meshes

```xml
<svg>
  <defs>
    <dcPatch id="p1" boundary="inherit"
      boundaryColor="rgb(255 0 250) rgb(248 251 4) rgb(17 8 247)" />
    <dcPatch id="p2" boundary="inherit"
      boundaryColor="rgb(248 251 4) rgb(2 250 6) rgb(17 8 247)" />
    <dcPatch id="p3" boundary="inherit"
      boundaryColor="rgb(2 250 6) rgb(240 2 13) rgb(17 8 247)" />
    <dcPatch id="p4" boundary="inherit"
      boundaryColor="rgb(240 2 13) rgb(255 0 250) rgb(17 8 247)" />
  </defs>

  <g id="mesh">
    <path d="M15,20 L85,15 L50,40 z" fill="url(#p1)" />
    <path d="M85,15 L80,65 L50,40 z" fill="url(#p2)" />
    <path d="M80,65 L30,67 L50,40 z" fill="url(#p3)" />
    <path d="M30,67 L15,20 L50,20 z" fill="url(#p4)" />
  </g>
</svg>
```
How to render Diffusion Curves quickly...
A fast and accurate DC renderer

- In 2013 CiSRA published a paper at SIGGRAPH Asia.

- This paper describes a boundary element renderer

- Extremely high quality
  - Highly accurate result
  - No leakage
  - Consistent rendering under all transformations

- Slow to render

- But extremely useful for our understanding and for testing

---

In 2013 CiSRA published a paper at SIGGRAPH Asia. This paper describes a boundary element renderer. Extremely high quality:
- Highly accurate result
- No leakage
- Consistent rendering under all transformations

Slow to render

But extremely useful for our understanding and for testing.
Diffusion Curves in the context of SVG

The bi-harmonic model (visually)

- Author can control, independently on each side of the curve:
  - Colour
  - Gradient

- An equal gradient value on either side of the curve gives a smooth transition of colour across the curve
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

Leakage
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

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

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
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

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

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
A fast and accurate DC renderer

Performance figures

Tested on a single processor Intel i5 2.67 GHz
A fast and accurate DC renderer

Visual comparison

CiSRA multi-grid renderer

CiSRA BEM renderer
A fast and accurate DC renderer

Code complexity?

- Multigrid renderer is not highly complex
  - It's a very efficient approximation of a very complex solution

- Less than 3000 lines to prepare multigrid input
- 500 lines for multigrid solver
Finishing up

- We feel confident that the diffusion curves technology will be able to meet the requirements of the web
- We have demonstrated a renderer with suitable performance
- SVG and diffusion curves are an excellent fit
  - Diffusion curves place authors first
  - SVG is a graphics format that is read and written by humans
    - Curve data is intrinsically scalable
- The use cases are strong
- We plan to be active in proposing a module for SVG and would love your input in future
Finishing up

Questions?

Thankyou for having me!

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