Introduction

Scalable Vector Graphics

Riccardo Pucella

September 17, 2015

Evaluating <canvas>

- Simple, straightfoward
- Three characteristics:
    - Programmatic: can only draw in JavaScript
    - Unstructured: forgets *how* something was drawn
    - Bitmapped: canvas is a box of pixels

Alternative: <svg>

Scalable Vector Graphics (SVG):

- Expressible within HTML
- Structured

  a graphic is a composition of graphical elements
- Scalable

  a graphic can be resized without losing resolution

Alternative: <svg>

Scalable Vector Graphics (SVG):

- Expressible within HTML

- Structured

  a graphic is a composition of graphical elements

- Scalable

  a graphic can be resized without losing resolution

These are not incompatible with <canvas>
It just requires work

Alternative: <svg>

Scalable Vector Graphics (SVG):

- Expressible within HTML
- Structured
  a graphic is
  elements
- Scalable
  a graphic c
  resolution

More interestingly, SVG graphics can be read and written by other tools, such as *Adobe Illustrator*.

Or edited in a text editor.

The <svg> element

Simple enough - similar to <canvas>:

```
<svg id="demosvg" width="700" height="500">
  ...
</svg>
```

Think of this as a container for SVG graphical elements

The <svg> element

Simple enough - similar to <canvas>:

```
<svg id=″demosvg″ width=″700″ height=″500″>
   ...
</svg>
```

Think of th

elements

Strictly speaking:

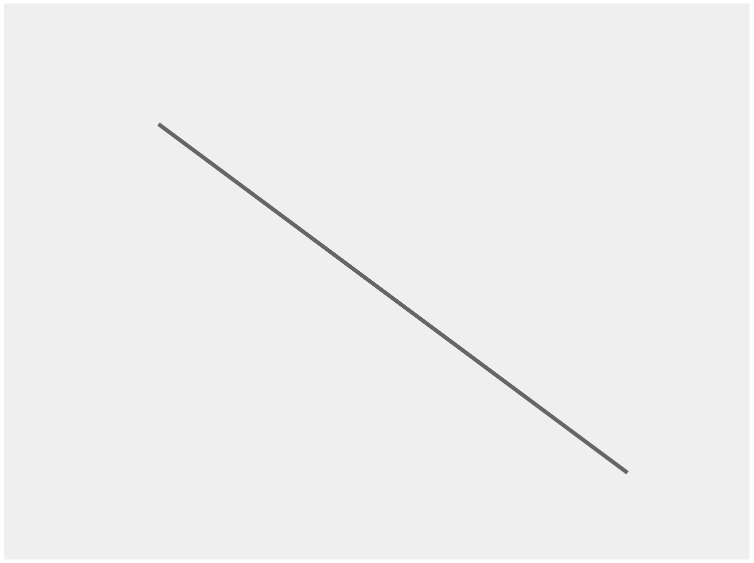
```
<svg id=″demosvg″ width=″700″ height=″500″
     viewport=″0 0 700 500″ version=″1.1″
     xmlns=″http://www.w3.org/2000/svg″>
</svg>
```

Primitive element <line>

Describes a line from point A to point B

```
<line x1=”100” y1=”100” x2=”600” y2=”400”>
</line>
```

Use styling to change the appearance of the line.

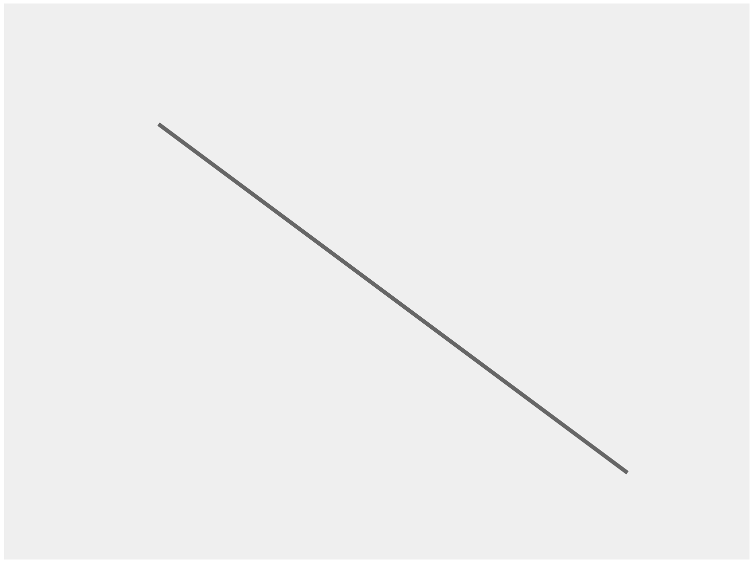


Primitive element <line>

Describes a line from point A to point B

```
<line x1=”100” y1=”100” x2=”600” y2=”400” />
```

Use styling to change the appearance of the line.

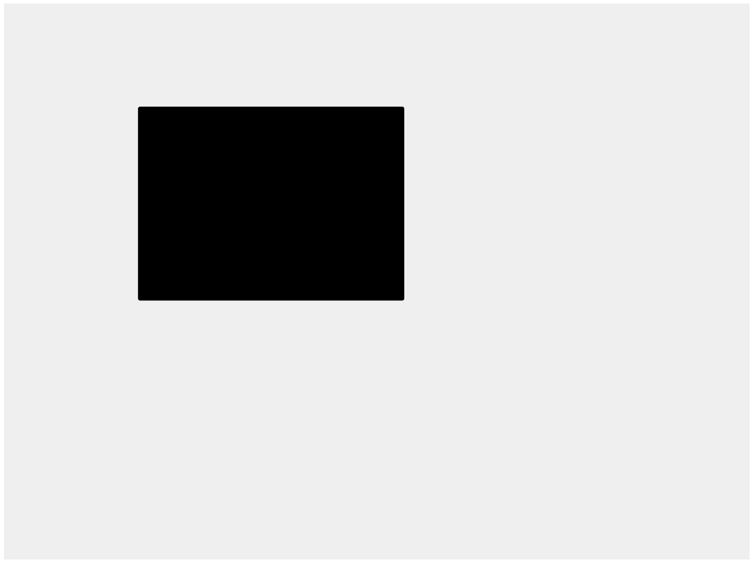


Primitive element <rect>

Describes a rectangle with a corner and a height and width:

```
<rect x=”100” y=”100”
      height=”200” width=”300” />
```

Use styling to change the appearance of the rectangle.

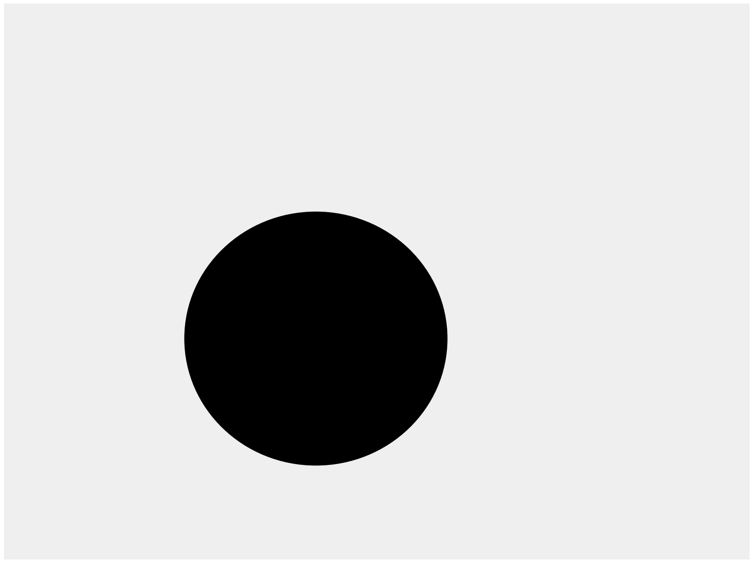


Primitive element <circle>

Describes a circle with a given center and radius:

```
<circle cx=”300” cy=”300” r=”100” />
```

Use styling to change the appearance of the circle.

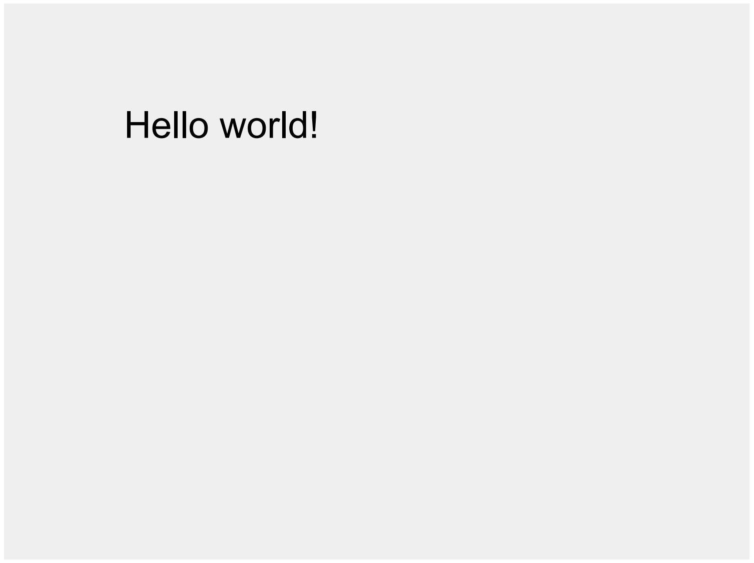


Primitive element <text>

Describes text that gets written in the graphics at a given point:

```
<text x=”100” y=”100”>
  Hello world!
</text>
```

Use styling to change the appearance of the text.



Primitive element <text>

Putting it all together:

```
<svg id=”demosvg” width=”700” height=”500”>
  <rect x=”100” y=”100”
        height=”200” width=”300” />
  <circle cx=”300” cy=”300” r=”100” />
  <text x=”100” y=”100”>Hello world!</text>
</svg>
```

Other primitive elements

- paths
- ellipses
- polygons

All behave pretty much as above

SVG graphics and the DOM

Modifying SVG graphics can be done with the usual tools for modifying the DOM

- Straight JavaScript
- jQuery
- etc...

Tiny bit of trickery might be needed because SVG is an embedded language in HTML - so we have to deal with namespaces

- D3 will fix that

Example: adding a circle to an SVG

```
var svgns = "http://www.w3.org/2000/svg";
var circle = document.createElementNS(svgns,"circle");
circle.setAttributeNS(null,"cx","300");
circle.setAttributeNS(null,"cy","300");
circle.setAttributeNS(null,"r","100");
document.getElementById("svgdemo").appendChild(circle);
```

Example: adding a circle to an SVG

```
var svgns = "http://www.w3.org/2000/svg";
var circle = document.createElementNS(svgns,"circle");
circle.setAttributeNS(null,"cx","300");
circle.setAttributeNS(null,"cy","300");
circle.setAttributeNS(null,"r","100");
document.getElementById("svgdemo").appendChild(circle);
```

Method chaining! The new hot thing. (Not really)

Get used to it -- we'll see more and more of it

Classes and IDs

SVG element can have ID and CLASS attributes.

Meaning that it’s possible to look them up by name and modify them by name.

Also, it means that we can attach event listeners to SVG elements.

Styling

Changing the color of, say, a rectangle, can be done in two ways:

- use a `fill=”green”` or `fill=”#aabbcc”` attribute (called a styling attribute)

- use CSS styling
  - `style=”fill: #aabbcc;”`
  - or through a CSS declaration and an ID or class

Watch out: style may override attribute

Available styles

Too many to list.

Common ones:

- fill	color of a filled shape
- fill-opacity	how opaque to fill a shape
- stroke	color of a line or edge
- stroke-width	width of line or edge
- font-family	for text
- font-size	for text
- font-weight	for text
- text-anchor	for text
- ...	

Transformations

Every SVG element has a **`transform`** attribute:

`transform=”rotate(`*`angle`*`,`*`x`*`,`*`y`*`)”`

`transform=”translate(`*`x`*`,`*`y`*`)”`

`transform=”scale(`*`ratio`*`,`*`x`*`,`*`y`*`)”`

Transformations can be composed

Groups

Creating a group lets you treat a set of graphic elements as a single graphic element:

```
<g fill=”blue”>
  <rect ...>
  <circle …>
</g>
```

Blue applied as a fill to all elements in group (unless overridden in an element)

Groups

Creating a group lets you treat a set of graphic elements as a single graphic element:

```
<g transform=”rotate(45,100,100)”>
   <rect ...>
   <circle …>
</g>
```

Rotate all elements in the group by 45 degrees around (100,100)

Groups

Creating a group lets you treat a set of graphic elements as a single graphic element:

```
<g id=”mygroup” class=”blues”>
   <rect ...>
   <circle …>
</g>
```

Give a name to the group

- can attach event listener
- can apply styles

Like a <div> in HTML

Summary

Both <canvas> and <svg> let you describe graphics

- unstructured versus structured
- bitmapped versus scalable
- <canvas> is operational
  <svg> is denotational

Generally:

- <canvas> used for art
- <svg> for program-generated graphics