SUPINFO International University

Sign of Success

Graphics API

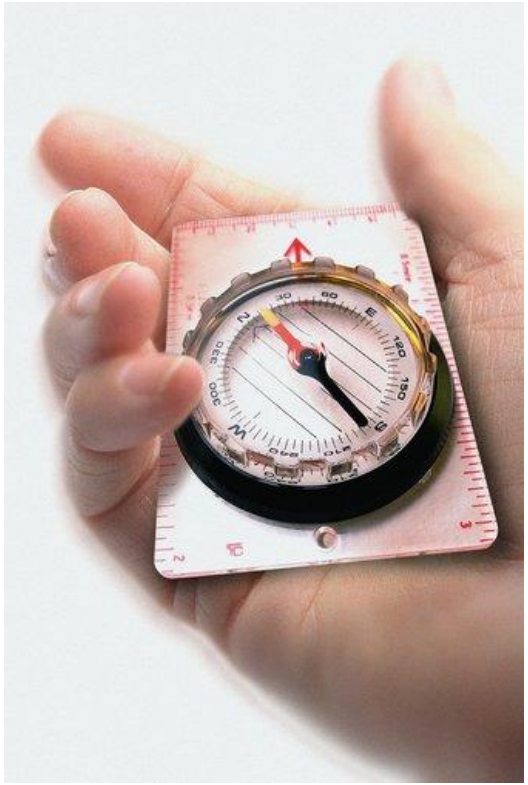
HTML drawing

Course objectives

By completing this course you will be able to:

- Understand graphics API concept
- Draw shapes with SVG
- Draw shapes with Canvas
- Choose one or the other depending on what you need to do

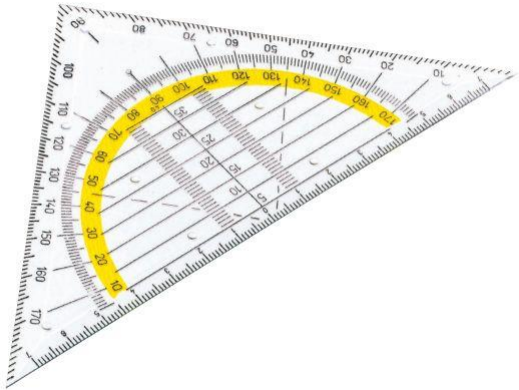
Course plan



- SVG
- Canvas
- SVG or Canvas ?

Graphics API

SVG



Presentation

- Graphics in HTML5
  - Done by markups
- Simple and quick rendering
  - Suitable for lightweight graphics
  - Preferred for static design
- Already recommended by W3C

History

- In the late nineties, there were two different markup languages for graphics:
  - PGML (Precision Graphics Markup Language)
    - Created by Adobe
    - Supported by IBM, Netscape and Sun
    - Based on PDF and PostScript formats
  - VML (Vector Markup Language)
    - Created by Microsoft, Macromedia, Autodesk, Hewlett-Packard and Visio
    - Implemented in Internet Explorer, Microsoft Office and Silverlight

History

- The war between these two formats was bad for developers usage

- W3C spent a long time to define a normalization and take advantages from both of them

- In 2000, SVG 1.0 specification was born ! ☺
  - Now SVG 1.1 is natively supported by all major browsers

Syntax

- Tag <svg>
  - Acts as a element
  - Can contain several shapes

- One tag by shape: <circle>, <line>, …
  - Enhanced by attributes
  - Specify origins, and dimensions

The SVG tag

- The <svg> tag

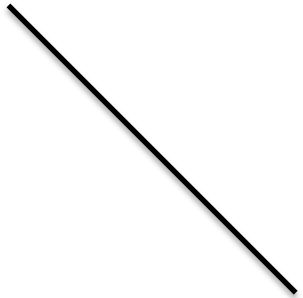
```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
    <!-- Your shapes here -->
 </svg>
```

- Root element for your shapes defined inside
  - Their coordinates will be calculated from the SVG top left corner element

Line

- Used to draw a line:
  - x1 and y1 for departure coordinates
  - x2 and y2 for arrival coordinates

```
<line x1="0" y1="0" x2="600" y2="600" fill="black" />
```



Polyline

- Special line including many points:

```
<polyline
   points="0,10 20,10 30,20 50,0 70,20
      90,0 110,20,120,10 150,10"
   fill="red" stroke="black" stroke-width="3" />
```

- Without fill attribute, just a multi-pointed line:

```
<polyline points="20,20 40,25 60,40 80,
                  120 120,140 200,180"
   style="fill:none;stroke:black;
      stroke-width:3" />
```

Rect

- Used to draw a rectangle:
  - x and y for coordinates
  - width and height for dimensions

```
<rect x="550" y="0" width="300" height="400" fill="black" />
```

Circle

- Used to draw a circle:
  - cx and cy for coordinates
  - r for radius

```
<circle cx="60" cy="60" r="30" fill="rgba(30, 150, 255, 1)" />
```

Ellipse

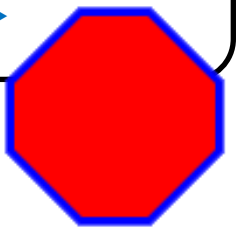
- Almost same than circle:
  - cx and cy for coordinates
  - rx and ry for radius

```
<ellipse cx="60" cy="60" rx="30" ry="60" fill="blue" />
```

Polygon

- Designs shapes with at least three points:
  - Unlike polyline, all the lines connect up (ex: the first and the end point is connected)
  - Same attributes than polyline

```
<polygon points="40,10 70,10 100,40 100,70
                 70,100, 40,100 10,70, 10,40"
   style="fill: red; stroke: blue; stroke-width: 4;" />
```



Text

- Outputs a simple text:
  - Mostly like a element, but in SVG!
  - Can be reshaped thanks to the « transform » attribute

```
<text x="15" y="15" fill="brown"
    transform="rotate(45)"
    style="font-size: 24px;" >
        Take me to the top!
</text>
```

Text

- Outputs a simple text:
  - If you want to jump lines, enclose each line by the <tspan> tag and specify their coordinates!
    - x and y is related to the svg container's coordinates
    - dx and dy is added to the previous element's coordinates

```
<text y="15" fill="brown" transform="rotate(45)"
    style="font-size: 24px;" >
    <tspan x="75">SVG</tspan>
    <tspan dy="20" x="75">is soooo</tspan>
    <tspan dy="20" x="75">cool</tspan>
</text>
```

Filters

- SVG also provides filters!
  - Simple tags to apply over SVG components
    - Change shapes, colors, opacity, …
  - Useful for animations and picture retouching!

- Can be defined two ways:
  - By the « filter » CSS property
  - Directly in the shape tag (circle, rect, …)

Filters

- Filters are encapsulated inside the <defs> tag
  - Containing all definitions for filters and gradients

- Usage:

```
<svg xmlns=... >
    <defs>
        <!-- Filters and/or gradients -->
    </defs>
    <rect ... />
</svg>
```

Filters

- In order to use your filters:
  - Define an id for each of them:

```
<filter id="filter1" ... >
   <!-- My filter -->
</filter>
```

  - Link the filter to your shape thanks to the filter attribute:

```
<rect width="90" height="90" fill="yellow"
      filter="url(#filter1)" />
```

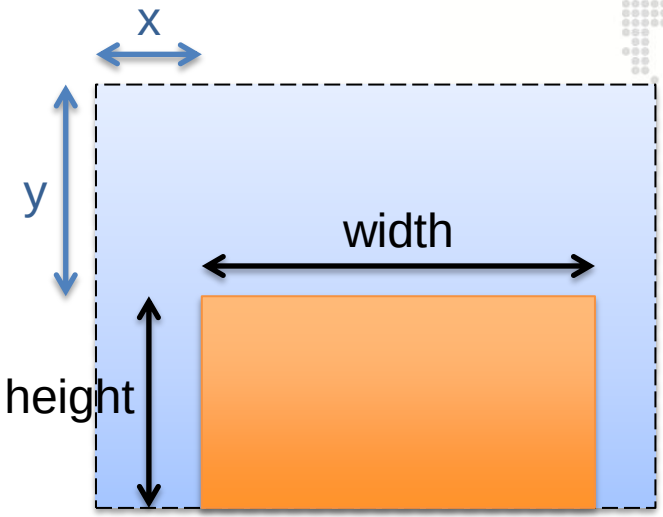
Filter common attributes

- Four optionnal attributes to remember:
  - You must specify their values in percentages
  - These attributes can truncate your SVG shape

Property	Default	Use
x	**0%**	Moves the filter at right
y	**0%**	Moves the filter at bottom
width	**100%**	Specify the x-length of the filter
height	**100%**	Specify the y-length of the filter

Filter common attributes

Look at the following schema, and the example:

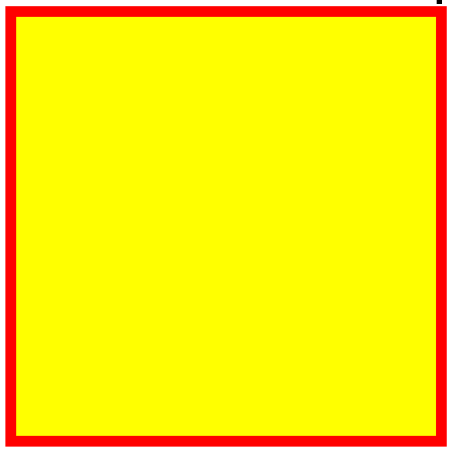



```
<filter id="filter1" x="20%" y="50%" width="70%" height="50%">
    <!-- Some effects -->
</filter>
```

Filters

- For the next examples, we'll consider a single shape in our SVG element, described as follows:

```
<svg>
    <defs>
        <filter id="f1">
        <!-- We'll define our effects here -->
        </filter>
    </defs>
    <rect width="400" height="400" x="10" y="10"
        stroke-width="10" stroke="red"
        fill="yellow" filter="url(#f1)" />
</svg>
```



Effects

- Many effects are available in SVG:
  - Flood
  - GaussianBlur
  - ColorMatrix
  - …

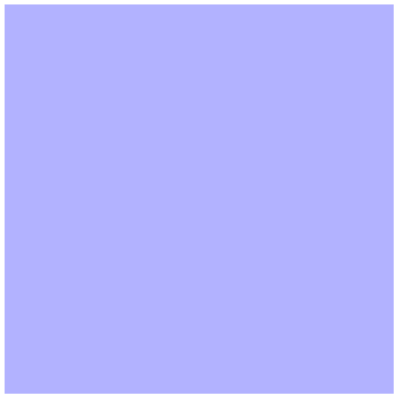
- We'll concentrate on these three!
  - Here is the complete list:

    http://www.w3.org/TR/SVG/filters.html

FeFlood

- Covers the shape with the specified color:
  - Be careful: this will override fill and fill-stroke property

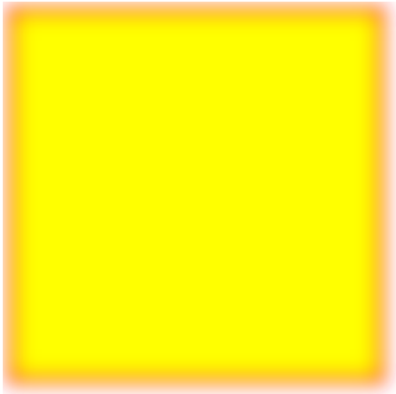
```
<feFlood flood-color="blue" flood-opacity=".3" />
```



FeGaussianBlur

- Performs on a gaussian blur:
  - Depending on the stdDevitation attribute

```
<feGaussianBlur stdDeviation="10" />
```



FeColorMatrix

- Change your colors by:
  - HUE
  - Saturation
  - Luminance
  - A complete matrix

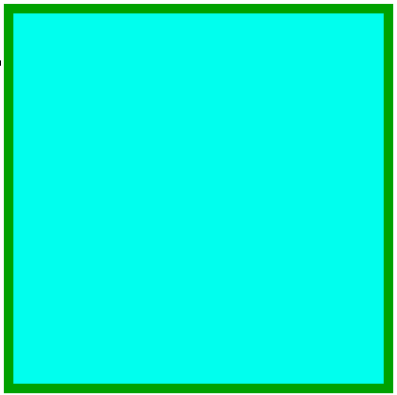
- Here is a tool for better understanding:

  http://www.colorsontheweb.com/colorwizard.asp

FeColorMatrix

- HUE:
  - Takes one int argument as values
  - « Rotate » the base color

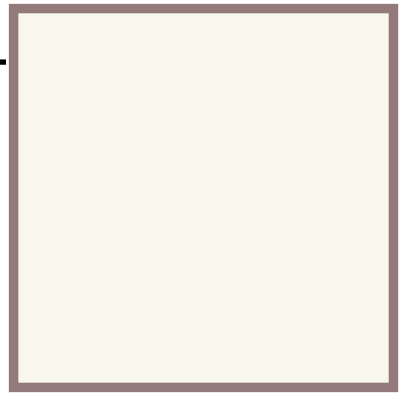
```
<feColorMatrix type="hueRotate" values="90" />
```



FeColorMatrix

- Saturation:
  - Takes one real argument as values (from 0 to 1)
  - Saturates the base color

```
<feColorMatrix type="saturate" values="0.1" />
```



FeColorMatrix

- Luminance:
  - Invert colors and associates them to a grey tone, depending on their luminance

```
<feColorMatrix type="luminanceToAlpha" />
```

FeColorMatrix

- Matrix:
  - Takes a complete 5 x 4 matrix:
    - Transforms base color by matrix multiplication
  - Example with a random base color: rgba(255, 0, 127, 0.7)
    - Transformed into matrix:

R	G	B	A	1
1	0	0,5	0,7	1
1	0	0,5	0,7	1
1	0	0,5	0,7	1
1	0	0,5	0,7	1

FeColorMatrix

- Matrix reminders:
  - Let’s specify this random matrix:

R	G	B	A	1
1	0	0	0	0
0	0,3	0,5	0	0
0	0	0,6	0	0
0	0	1	0	0

  - Multiplication result:

R	G	B	A	1
1	0	0,5	0,7	1
1	0	0,5	0,7	1
1	0	0,5	0,7	1
1	0	0,5	0,7	1

R	G	B	A	1
1	0	0	0	0
0	0,3	0,5	0	0
0	0	0,6	0	0
0	0	1	0	0

R	G	B	A	1
1	0	0	0	0
0	0	0,25	0	0
0	0	0,3	0	0
0	0	0,5	0	0

FeColorMatrix

- Matrix reminders:
  - Now that we have the following result:

	R	G	B	A	1
R'	1	0	0	0	0
G'	0	0	0,25	0	0
B'	0	0	0,3	0	0
A'	0	0	0,5	0	0

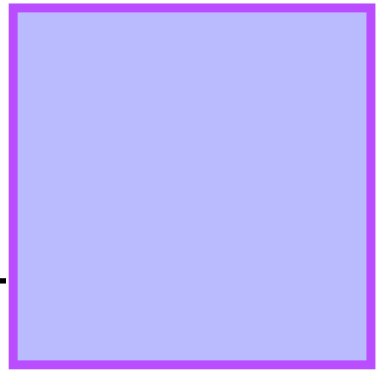
  - The resulting color will be:
    - Red = 255 x (1 + 0 + 0 + 0 + 0) = 255
    - Blue = 255 x (0 + 0 + 0.25 + 0 + 0) = 63
    - Green = 255 x (0 + 0 + 0.3 + 0 + 0) = 76
    - Alpha = 1 x (0 + 0 + 0.5 + 0 + 0) = 0.5

rgba(255,63,76,0.5)

FeColorMatrix

- Matrix:
  - SVG example:

```
<feColorMatrix type="matrix"
    values="0 0 0 .33 0
            0 .33 0 0 0
            1 0 0 0 0
            0 0 0 .7 0" />
```



Gradients

- As CSS3 gradient property:
  - Radial
  - Linear
- Also defined in <defs> tag
- Used with fill attribute

Linear Gradient

- LinearGradient:
  - Don’t forget to change fill attribute on shape:

fill="url(#grad1)"

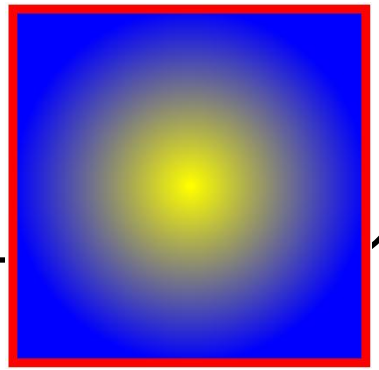
```
<linearGradient id="grad1">
    <stop offset="0%" style="stop-color:rgb(255,255,0);
        stop-opacity:1" />
    <stop offset="100%" style="stop-opacity:1;
        stop-color:rgb(255,0,0);" />
</linearGradient>
```

Radial Gradient

- RadialGradient:
  - Don’t forget to change fill attribute on shape:

fill="url(#grad1)"

```
<radialGradient id="grad1">
    <stop offset="0%" style="stop-color:rgb(255,255,0);
        stop-opacity:1" />
    <stop offset="100%" style="stop-opacity:1;
        stop-color:rgb(0,0,255);" />
</radialGradient>
```



CSS with SVG

- Of course you can take advantage of CSS with SVG
  - Even use CSS3 namespaces!

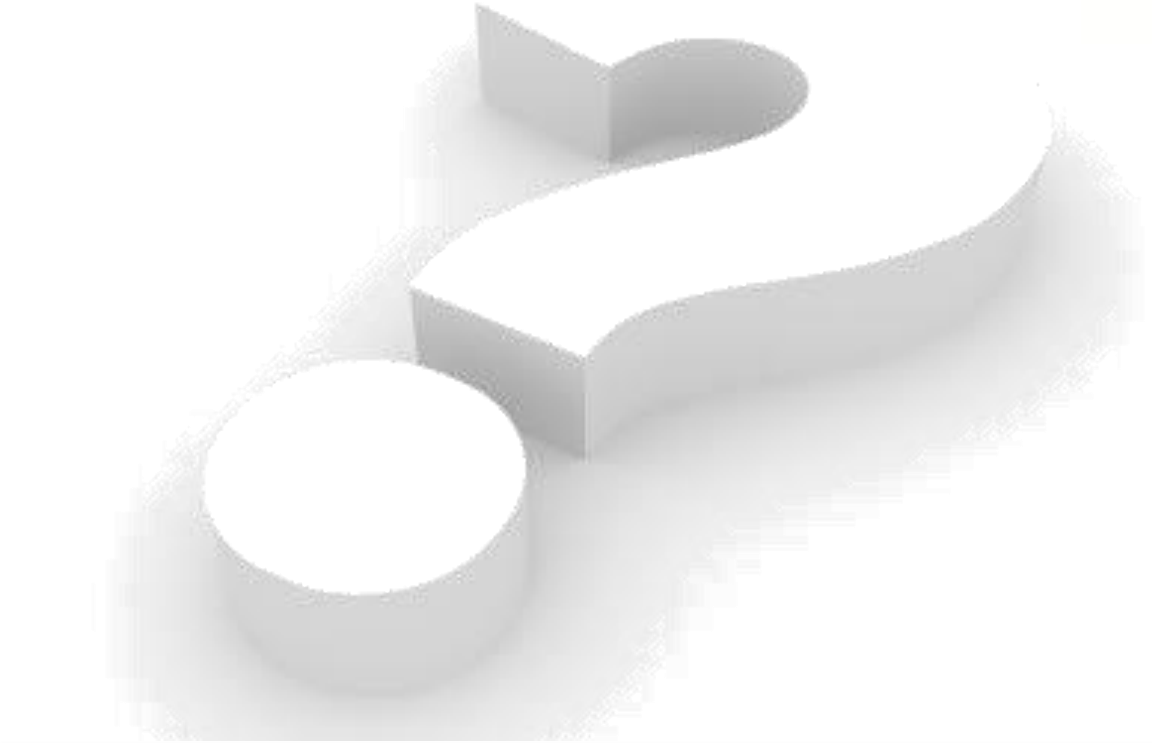
- Like all other tags, SVG is part of the DOM
  - Easily accessible with CSS…
  - …and JavaScript!

CSS3 namespaces and SVG

```
<html>
<head>
    <title>SVG example</title>
    <style>
        /* Define namespaces before anything else */
        @namespace "http://www.w3.org/1999/xhtml";
        @namespace s "http://www.w3.org/2000/svg";

        figure { width: 180px; text-align: center; }
        s|svg { margin: 10px 30px; height: 120px; }
        s|svg polygon { opacity: .5; }
    </style>
</head>
<body>
<h1>Look at my SVG!</h1>

<!-- ... -->
```

```
<!-- ... -->

  <figure>
    <svg>
      <defs>
        <filter id="f1">
          <feColorMatrix type="hueRotate"
              values="90"/>
          <feGaussianBlur stdDeviation="5" />
        </filter>
      </defs>
      <polygon … filter="url(#f1)" />
    </svg>
    <figcaption>SVG rocks</figcaption>
  </figure>
</body>
</html>
```

Questions?



Exercise

- Reproduce this image:

Graphics API

CANVAS API

Presentation

- JavaScript API
  - Only one DOM node: <canvas>
  - All your shapes are created programmatically
- Bitmap rendering
- Can draw in two and in three dimensions
  - Called respectively 2D and WebGL

Canvas context

- The canvas tag:
  - Defines a drawable area
  - Has two contexts (one for 2D, the other for 3D)
    - Mandatory to create shapes

```
<canvas id="canvas1"></canvas>

<script type="text/javascript">
    var context =
        document.getElementById("canvas1").getContext("2d");
</script>
```

Canvas basics

- Shapes' origin are usually defined by a kind of cursor
  - You can move this cursor!
  - Just like hovering your pencil over your sheet

```
var context =
    document.getElementById("canvas1").getContext("2d");
context.moveTo(100, 50); // 100px from left, 50px from top
```

- Call « stroke() » at the end of your drawing:

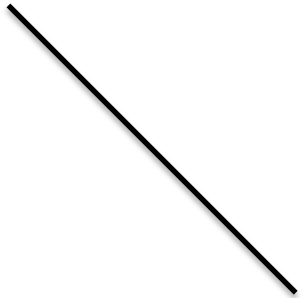
```
context.stroke(); // Outputs the shape
```

Line

- Used to draw a line:
  - Departure coordinates are defined by your cursor

context.lineTo(x, y);

```
context.moveTo(0, 0); // Move cursor at top left corner
context.lineTo(50, 50); // From 0,0 to 50,50
```



Rect

- Used to draw a rectangle:

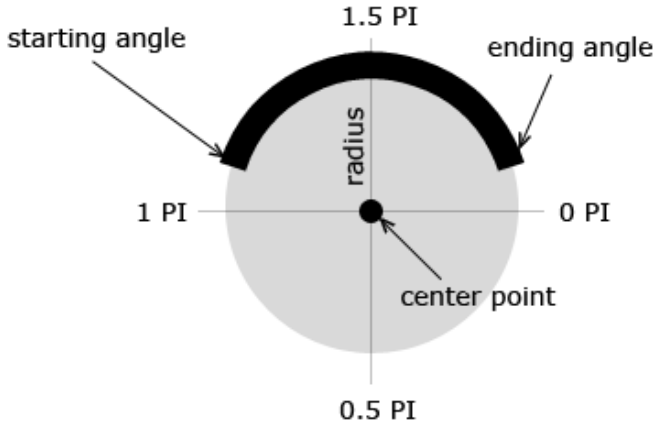
context.rect(x, y, width, height);

```
context.rect(550, 0, 300, 400);
```

Arc

- Used to draw an arc:

- Before viewing the syntax, take a look at the following schema:



Source: http://www.html5canvastutorials.com

Arc

- Used to draw an arc:

context.arc(x, y, radius, start, end, counterClockWise);

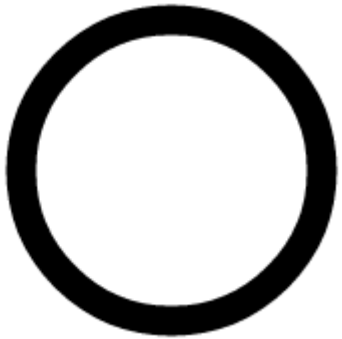
```
context.arc(250, 200, 75, Math.PI, 0, true);
```

Arc

- Also possible to draw circles:

context.arc(x, y, radius, start, end, counterClockWise);

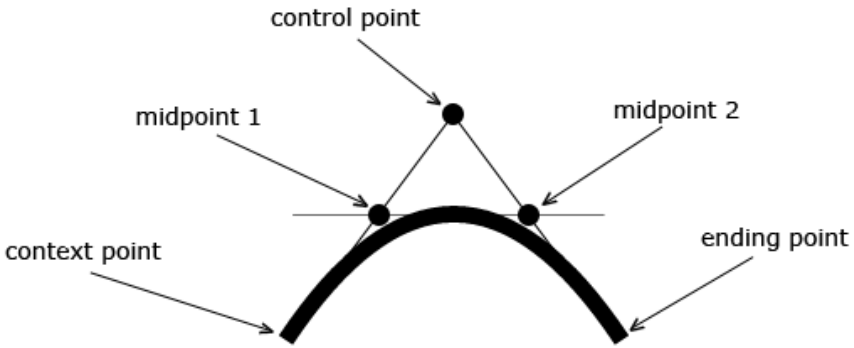
```
context.arc(250, 200, 75, 2 * Math.PI, 0, true);
```



Curves

- Two types of curves: quadratic and Bezier

- Before viewing the syntax, take a look at the following quadratic curve schema:



Source: http://www.html5canvastutorials.com

Quadratic Curve

- Curve with one control point
  - Origin defined by your current cursor position

context.quadraticCurveTo(cX, cY, endX, endY);

```
context.moveTo(200, 0);
context.quadraticCurveTo(50, 300, 150, 150);
```

Bezier Curve

- Curve with two control points
  - Origin defined by your current cursor position

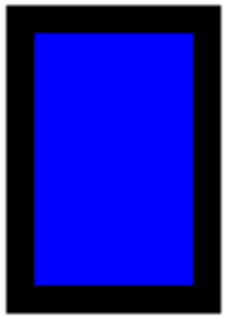
context.bezierCurveTo(cX1, cY1, cX2, cY2, endX, endY);

```
context.moveTo(200, 0);
context.bezierCurveTo(300, 250, 150, 0, 300, 100);
```

Context attributes

- Change drawing style:
  - Be sure to set it up before « fill() » and/or « stroke() »!

```
context.rect(100, 150, 100, 150); // Draw a rectangle
context.lineWidth = 15; // Width in pixels
context.strokeStyle = "rgba(0, 0, 0, 1)"; // Black
context.fillStyle = "blue"; // Change inner color
context.fill(); // Fill rectangle
Context.stroke(); // Draw borders
```



Canvas basics

- Canvas allows to « auto-close » your shapes
  - Last point will be linked to first one by a straight line

```
context.beginPath(); // Start drawing the shape
context.moveTo(100, 300);
context.lineTo(200, 200);
context.lineTo(300, 300);
context.strokeStyle = "red";
context.lineWidth = 15;
context.closePath(); // End drawing the shape
context.stroke();
```

Text

- Outputs a simple text
  - No need to use « stroke() » function

context.fillText(text, x, y);

```
context.fillText("Canvas rocks!", 100, 100);
```

Canvas rocks!

Text

- Change font defaults:

context.font = attributes;

```
context.font = "italic 40px Calibri";
```

Canvas rocks!

Image

- Drawing an image:

```
var oImage = new Image();
oImage.onload = function() {
    context.drawImage(oImage, 700, 50);
};
oImage.src = "/img/success.png";
```

LinearGradient

- Create gradient by relying on an imaginary line:

context.createLinearGradient(x1, y1, x2, y2);

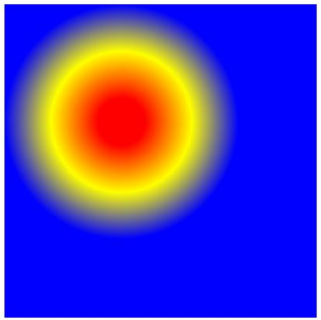
```
var grd = context.createLinearGradient(100, 100, 500, 100);
grd.addColorStop(0,"red");
grd.addColorStop(.5, "yellow");
grd.addColorStop(1, "blue");
context.rect(100, 100, 500, 500);
context.fillStyle = grd;
context.fill();
```

RadialGradient

- Create gradient by relying on two imaginary circles:

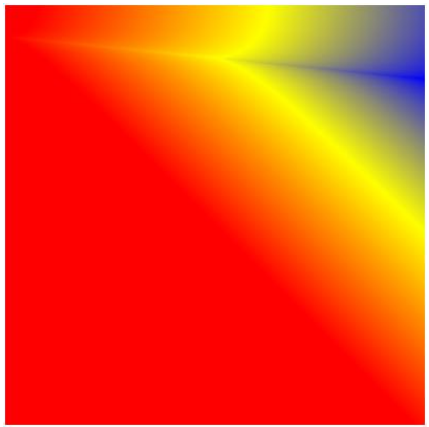
context.createRadialGradient(x1, y1, r1, x2, y2, r2);

```
var grd = context.createRadialGradient(250, 250, 30, 250,
    250, 150);
grd.addColorStop(0,"red");
grd.addColorStop(.5, "yellow");
grd.addColorStop(1, "blue");
context.rect(100, 100, 500, 500);
context.fillStyle = grd;
context.fill();
```

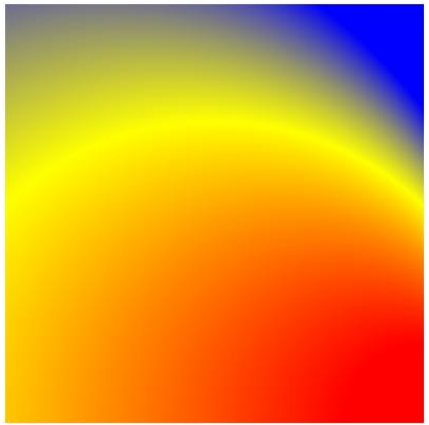


RadialGradient

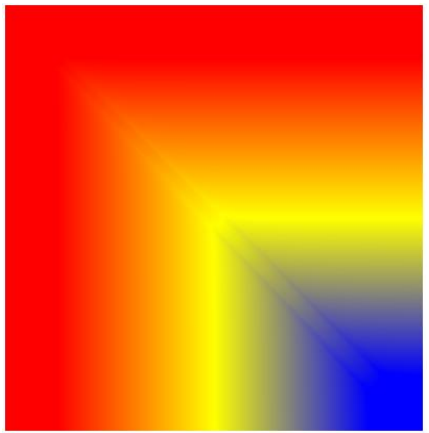
- With radial gradients, you can put the two circles where you want
  - Allows all gradient types you need



(100, 100, 30, 500, 100, 70)



(500, 500, 75, 100, 500, 500)



(150, 150, 5, 450, 450, 10)

WebGL (Canvas 3D)

Canvas 3D API (WebGL):

- Based on OpenGL
- Built in modern browsers without any plug-ins use
- Use graphic processing unit (GPU) through GLSL language

WebGL (Canvas 3D)

Canvas 3D API (WebGL):

- Some examples:
  - http://mrdoob.com/
  - http://learningwebgl.com/blog/?page_id=1217
  - http://triggerrally.com/

We need to go deeper…

- Canvas is enriched by open source libraries

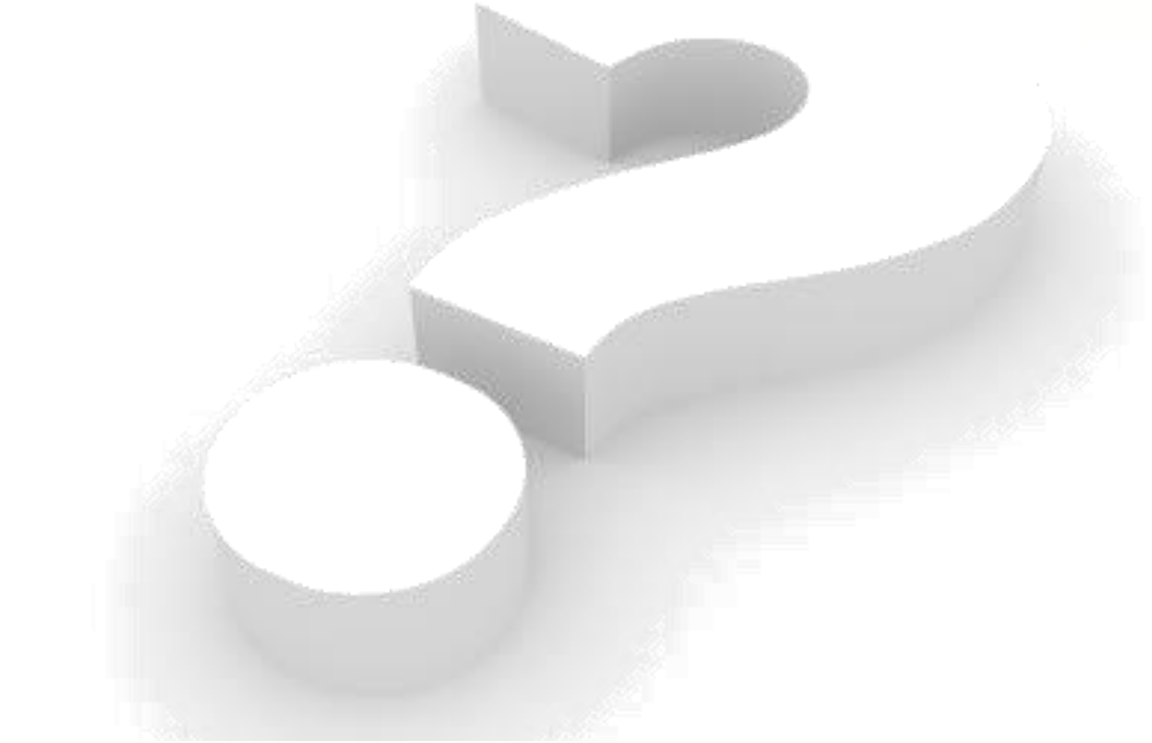
- Here are some of them:
  - Processing.js
  - KineticJS
  - Fabric.js
  - Easel.js
  - …

Web Laboratory example

- Discover a SUPINFO student gaming library
  - Use canvas to reproduce game environment
    - Gravity, collisions, object moving, NPC interactions…
  - Four classes for most cases handling
    - Element, Sprite, Game, Room

- Discover it at http://jsglib.no-ip.org/
  - Actually in version 2.1
  - For more informations, please contact Adrien Guéret

Questions?



Graphics API

SVG OR CANVAS ?

Both can draw, so which one to use?

Advantages

- SVG is great because:
  - Simple syntax
  - DOM elements allowing to use CSS and JS
  - More accessible as everything is written in markups
- Canvas is great because:
  - It’s all dynamic
  - Everything is a pixel
  - It’s really fast!

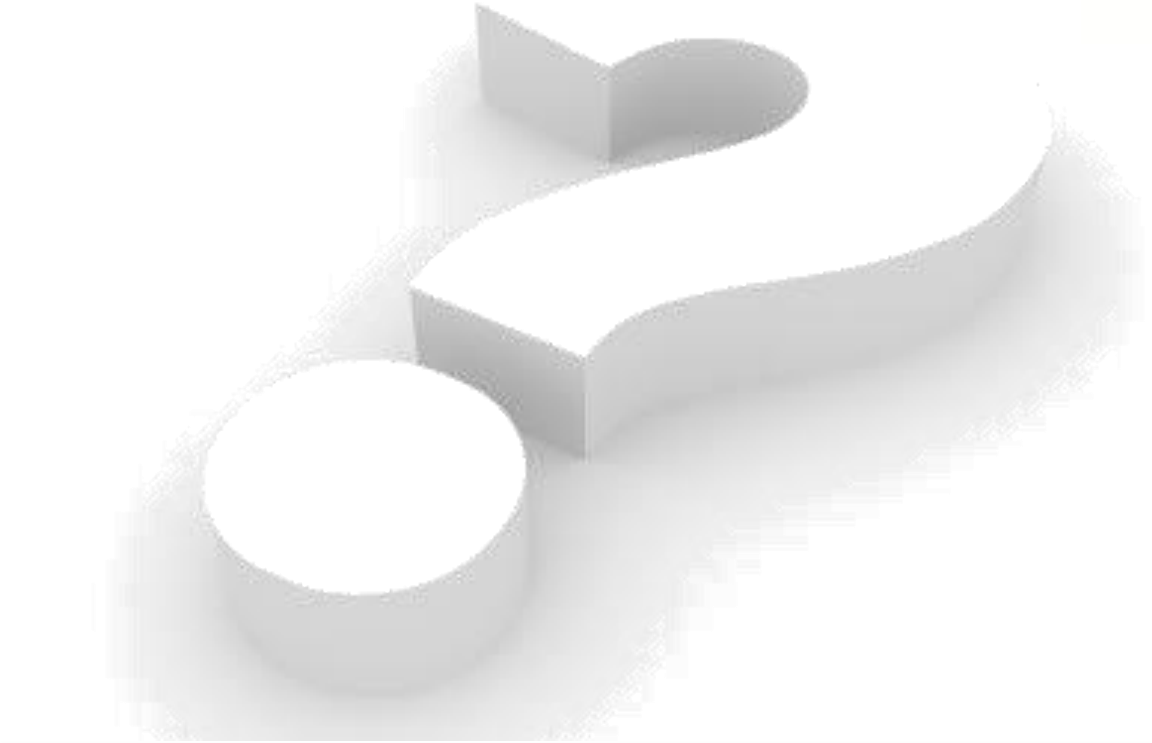
Disadvantages

- But SVG is:
  - Slow when DOM complexity increases
    - DOM API remains slightly slow
  - Only bi-dimensional
- But Canvas has:
  - No native animation API
  - Poor text rendering capabilities
  - No ARIA support

Conclusion

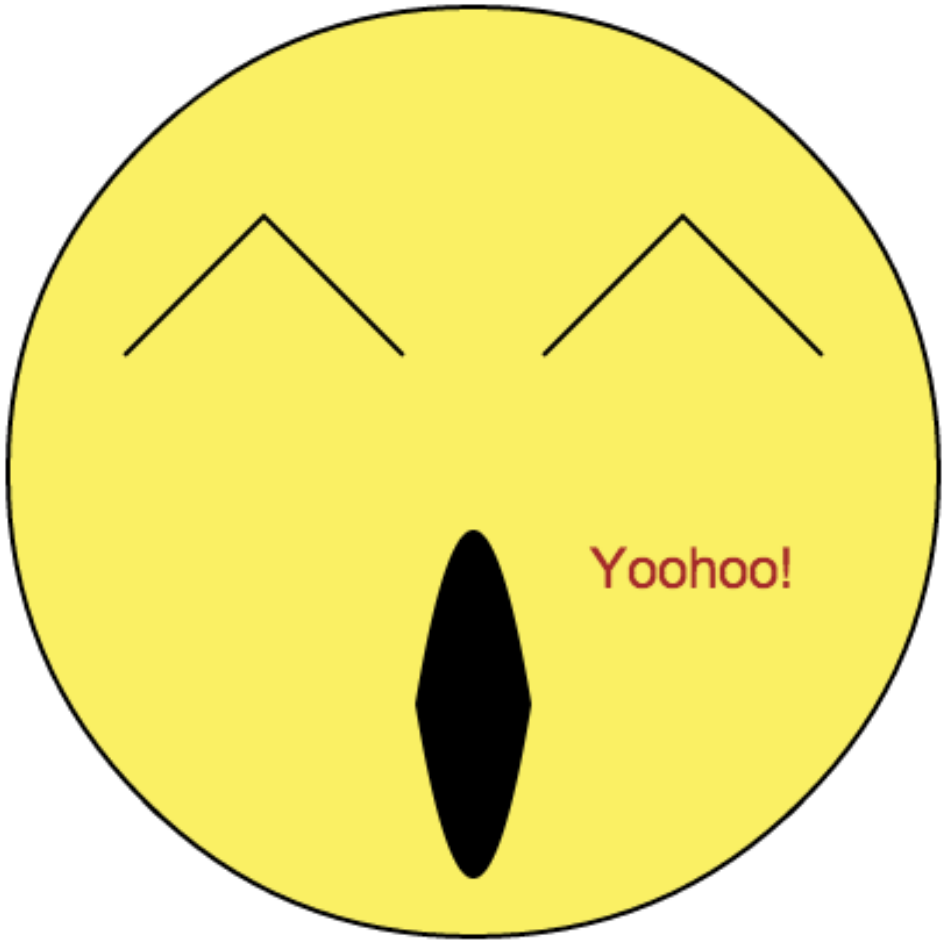
- SVG is designed for:
  - Cross-platform rendering
  - Data charts, interactive UI, …
- Canvas is designed for:
  - High speed rendering (for example games)
  - Image editing, color picker, …

Questions?



Exercise

- Reproduce this image:



Notes:

- Canvas can’t draw an ellipse as easily as SVG can. You’ll need to use two curves instead

- Filters are way more hard to do in canvas, so drop the cheekbones!

The end

SUPINFO International University

Sign of Success

Thanks for your attention