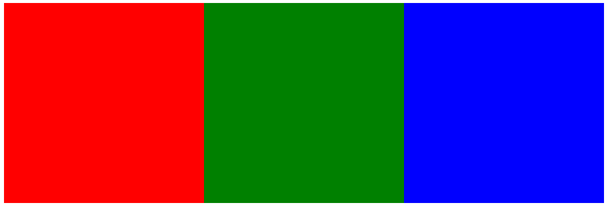
COM1008: Web and Internet Technology

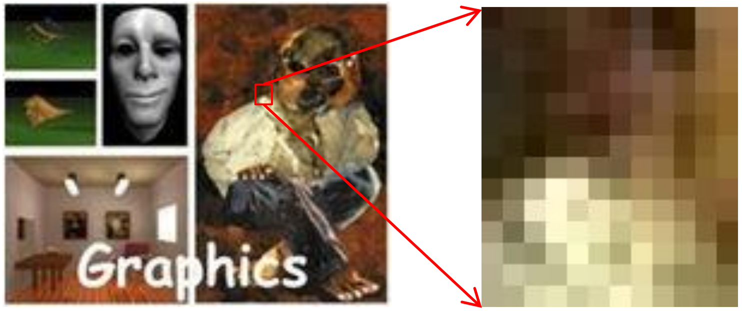
Lecture 16: SVG



Dr. Steve Maddock
s.maddock@sheffield.ac.uk

1. Introduction

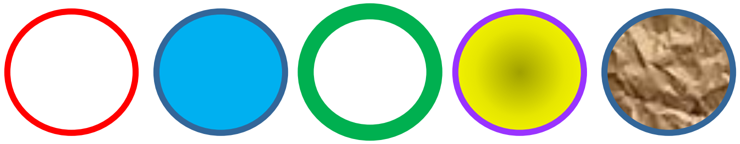
- In general, two kind of computer graphics:
- Raster graphics
  - Drawing/painting commands produce sets of coloured pixels on the screen
  - Screen image is made of a rectangular grid of pixels – bitmap



Today

- Vector graphics
  - Image is a list of geometric shape with attributes
- Scalable vector Graphics (SVG)

```
List of shapes:
   Circle(centre,  radius,
            border_colour,  fill_colour,
            fill_image)
   Circle(...
   ...
```



2. Scalable Vector Graphics (SVG)

- SVG is the W3C’s recommended markup language for vector graphics
- It is defined in XML and included in the HTML5 specification

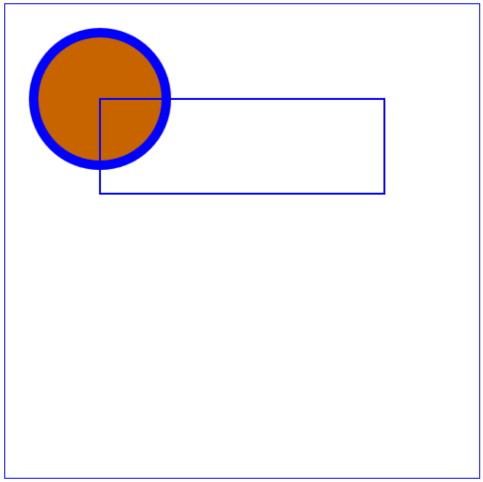
```
<svg xmlns="http://www.w3.org/2000/svg"
     version="1.1"
     id="example"
     width="500" height="500">
  <!--
    content here
  -->
</svg>
```

- (Most vector graphics on the Web is done using SWF (Flash) files.)

2.1 Example

svg1.css

```
svg { border: 1px solid blue; }
```



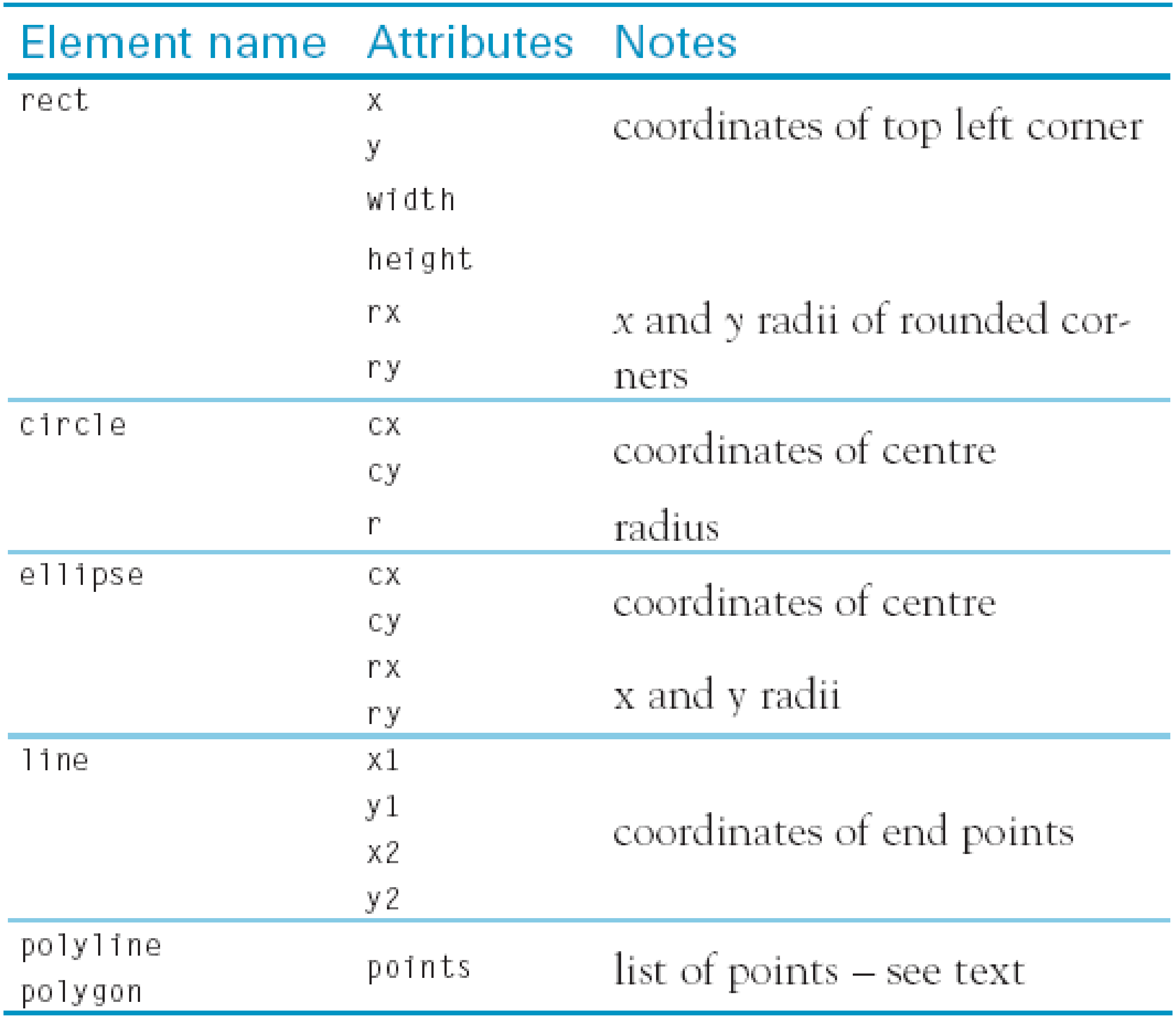
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>SVG test</title>
  <link rel="stylesheet" href="./css/svg1.css" />
</head>
<body>
  <svg xmlns="http://www.w3.org/2000/svg" version="1.1"
       id="example" width="500" height="500">
    <circle cx="100" cy="100" r="70"
     fill="rgb(200,100,0)"
     stroke="blue" stroke-width="10" />
    <rect x="100" y="100" width="300" height="100"
     fill="none"
     stroke="blue" stroke-width="10" />
</svg>
</body>
</html>
```

- The drawing area is 500x500

- Two shapes are defined: a **circle** and a **rect**angle

2.2 Shapes

- Shapes are defined by elements, whose attributes' values specify their size and position



Element name	Attributes	Notes

`rect`	`x`
	`y`
	`width`
	`height`
	`rx`
	`ry`
		coordinates of top left corner
		x and y radii of rounded corners

`circle`	`cx`
	`cy`
	`r`
		coordinates of centre
		radius

`ellipse`	`cx`
	`cy`
	`rx`
	`ry`
		coordinates of centre
		x and y radii

`line`	`x1`
	`y1`
	`x2`
	`y2`
		coordinates of end points

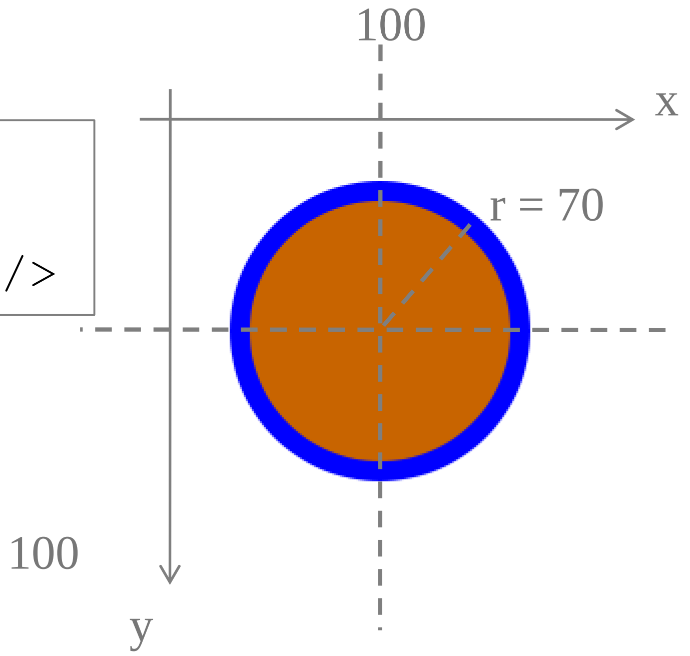
`polyline` `polygon`	`points`	list of points – see text

Chapman, N and J. Chapman, Web Design: A complete introduction, John Wiley & Sons, 2006.

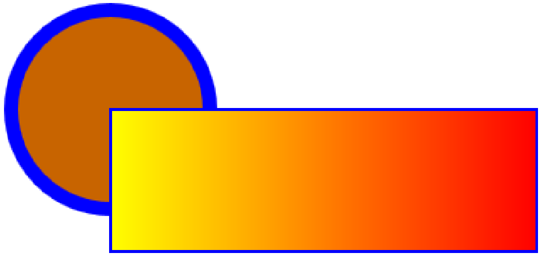
2.3 Fills

- The fill attribute: none; a colour, as in CSS; a paint server (definition of a gradient or pattern)

```
<circle cx="100" cy="100" r="70"
  fill="rgb(200,100,0)"
  stroke="blue" stroke-width="10" />
```



2.4 Fills: linear gradient



- Gradient examples:
  - https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Gradients

```
<defs>
  <linearGradient id="MyGradient">
     <stop offset="0%" stop-color="rgb(255,255,0)"/>
     <stop offset="100%" stop-color="rgb(255,0,0)"/>
   </linearGradient>
</defs>

<rect x="100" y="100" width="300" height="100"
  fill="url(#MyGradient)"
  stroke="blue" stroke-width="2" />
```

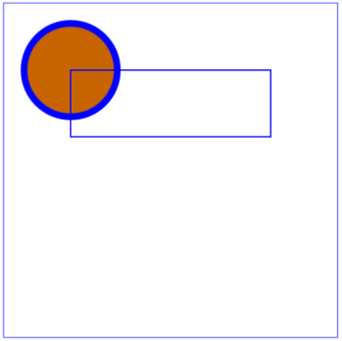
3. SVG and CSS

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>SVG test</title>
  <link rel="stylesheet" href="./css/svg1.css"
  <link rel="stylesheet"
        href="./css/svg1b.css" />
</head>
<body>
  <svg xmlns="http://www.w3.org/2000/svg" version="1.1"
       id="example" width="500" height="500">
    <circle id="c1" cx="100" cy="100" r="70" />
    <rect id="r1" x="100" y="100" width="300" height="100" />

  </svg>
</body>
</html>
```

svg1b.css

```
#c1 {
  fill: rgb(200,100,0);
  stroke: blue;
  stroke-width: 10;
}

#r1 {
  fill: none;
  stroke: blue;
  stroke-width: 2;
}
```

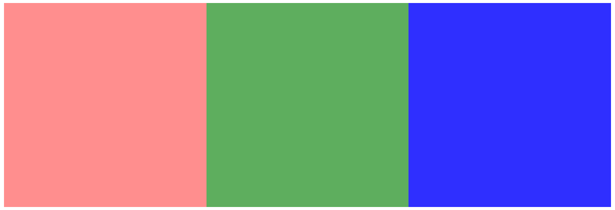
- Same example as previous, but attributes are now set using CSS

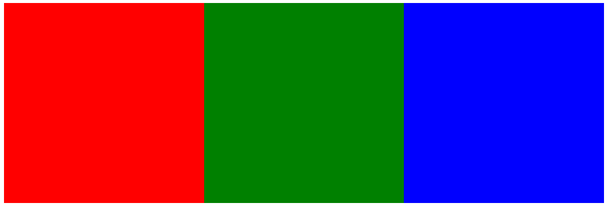


3. SVG and CSS

- Attributes can also be animated

```
<rect x="50" y="100" width="100" height="100" fill="red">
  <animate attributeType="CSS" attributeName="opacity"
    from="0" to="1" dur="1s" repeatCount="9" />
</rect>
```

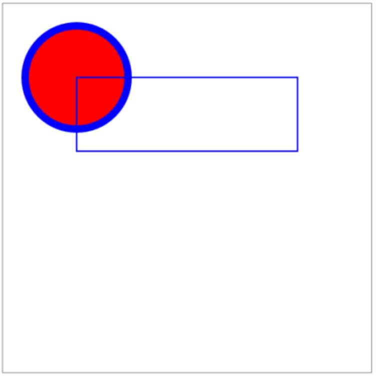




4. SVG and JavaScript

- SVG has full DOM support, so individual shapes can have event handlers and be accessed using JavaScript

```
<body>
  <svg xmlns="http://www.w3.org/2000/svg" version="1.1"
       id="example" width="500" height="500">

    <circle id="c1" cx="100" cy="100" r="70"
       fill="rgb(255,0,0)"
       stroke="blue" stroke-width="10" />
    <rect x="100" y="100" width="300" height="100"
       fill="none"
       stroke="blue" stroke-width="2" />

  </svg>
  <script src="./js/svg2.js"></script>
</body>
```

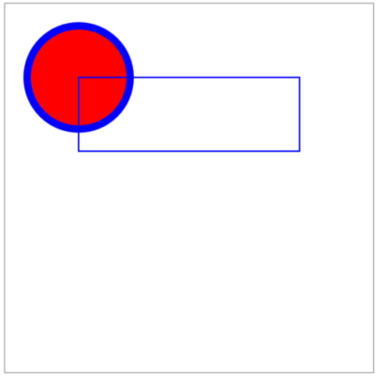


4. SVG and JavaScript

- When circle object is clicked, the function changeColor is called

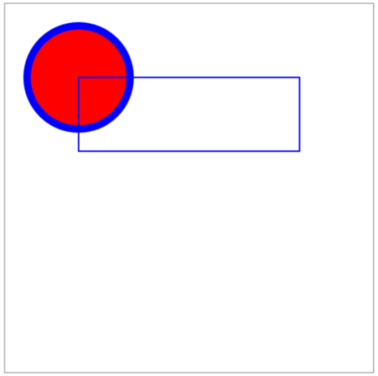
```
var colors = ["red", "green", "blue"];
var currentColor = 0;

// main program body
init();

// functions

function changeColor() {
  var circle = document.getElementById("c1");
  currentColor = (currentColor+1) % colors.length;
  circle.setAttribute("fill", colors[currentColor]);
}

function init() {
  document.getElementById("c1").addEventListener("click",
                                                 changeColor);
}
```

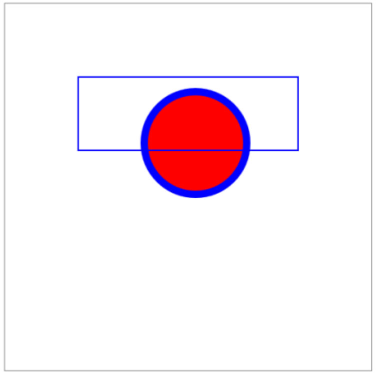
- Cycle round these colors as the circle is clicked on using the mouse



4.1 Example

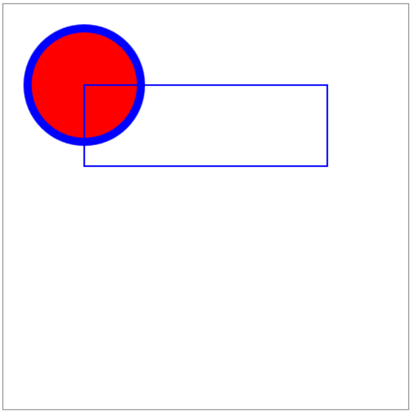
- The arrow keys on the keyboard are used to control the position of the circle
- Need to retrieve current centre position of the circle
  - var attribute = element.getAttribute(attributeName)
  - attribute is a string
- Then add/subtract x and y based on which key was pressed
- Then change the centre position of the circle
  - element.setAttribute(attributeName, value)
  - attributeName is the name of the attribute as a string
  - value is the desired new value of the attribute

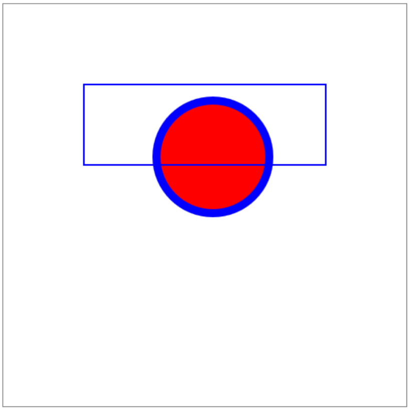




4.1 Example

```
<body>
  <svg xmlns="http://www.w3.org/2000/svg" version="1.1"
       id="example" width="500" height="500">
    <circle id="c1" cx="100" cy="100" r="70"
       fill="rgb(255,0,0)"
       stroke="blue" stroke-width="10" />
    <rect x="100" y="100" width="300" height="100"
       fill="none"
       stroke="blue" stroke-width="2" />
  </svg>
  <script src="./js/svg2b.js"></script>
</body>
```

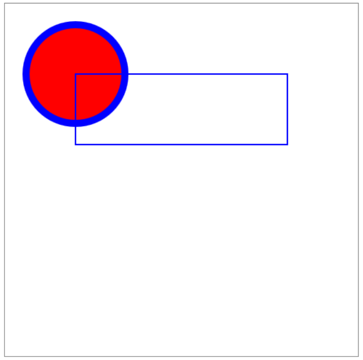


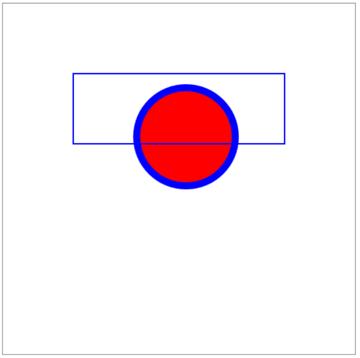


- Every time an arrow key is pressed the centre coordinates of the circle are updated

4.1 Example

```
var colors = ["red", "green", "blue"];
var currentColor = 0;

// main program body
init();

// functions

function init() {
  var circle = document.getElementById("c1");
  circle.addEventListener("click", changeColor);
  window.addEventListener("keydown", changeCentre);
}
```

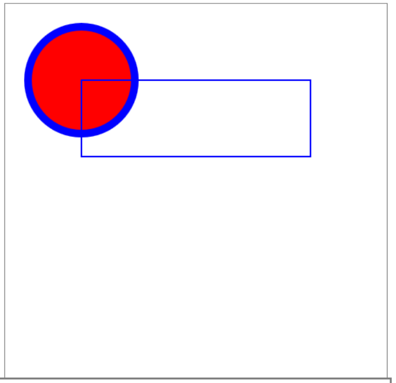




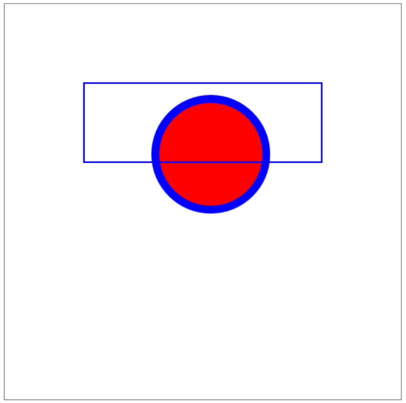
- Every time an arrow key is pressed the centre coordinates of the circle are updated

4.1 Example

```
function changeCentre(ev) {
  //console.log("changeY");
  var circle = document.getElementById("c1");
  var x=0, y=0, ix=0, iy=0;
  switch (ev.keyCode) {
    case 37:  /* Left arrow was pressed */
      ix = -5;
      break;
    case 39:  /* Right arrow was pressed */
      ix = 5;
      break;
    case 38:  /* Up arrow was pressed */
      iy = -5;
      break;
    case 40:  /* Down arrow was pressed */
      iy = 5;
      break;
  }
  x = parseInt(circle.getAttribute("cx")) + ix;
  y = parseInt(circle.getAttribute("cy")) + iy;
  //console.log(x+","+y);
  circle.setAttribute("cx", x.toString());
  circle.setAttribute("cy", y.toString());
}
```

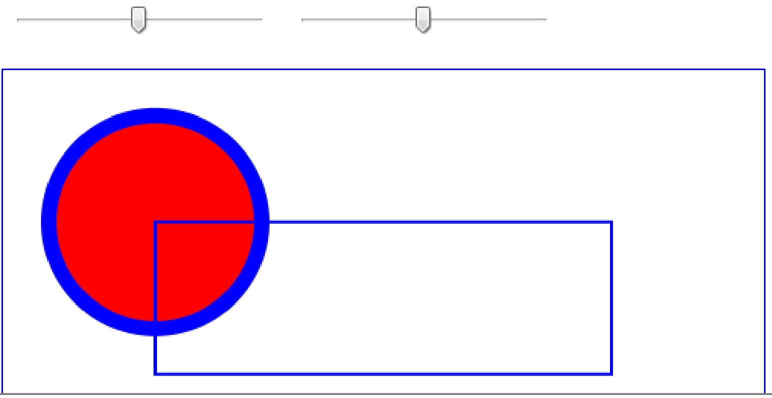


- Every time an arrow key is pressed the centre coordinates of the circle are updated



4.2 Example 2

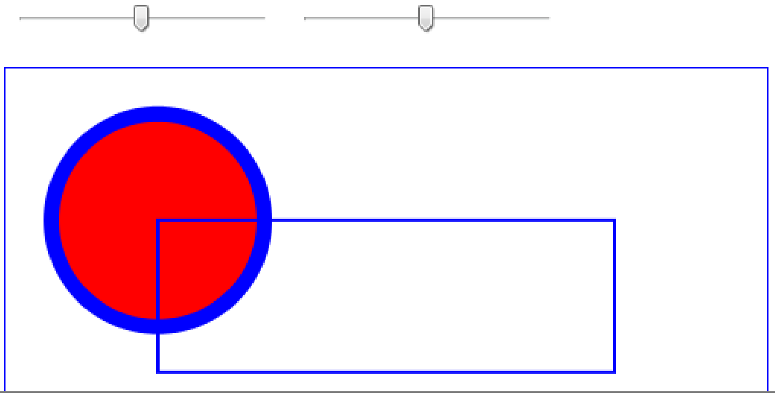
- Use sliders to control circle position



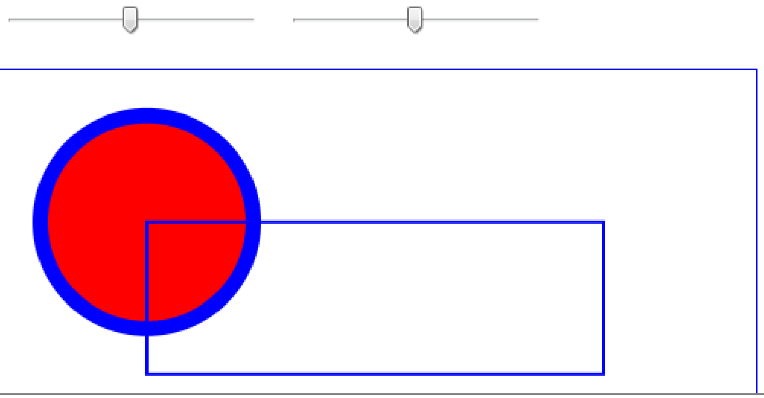
```
<body>
  <p>
    <input type="range" id="xslider" min="0" max="500" step="1" />
    <input type="range" id="yslider" min="0" max="500" step="1" />
  </p>
  <svg xmlns="http://www.w3.org/2000/svg" version="1.1"
       id="example" width="500" height="500">
    <circle id="c1" cx="100" cy="100" r="70"
       fill="rgb(255,0,0)"
       stroke="blue" stroke-width="10" />
    <rect x="100" y="100" width="300" height="100"
       fill="none"
       stroke="blue" stroke-width="2" />
  </svg>
  <script src="./js/svg2c.js"></script>
</body>
```

4.2 Example 2

- Use sliders to control circle position



```
function init() {
  var circle = document.getElementById("c1");
  circle.addEventListener("click", changeColor);
  var myinput = document.getElementById("xslider");
  myinput.addEventListener('input', changeXcentre);
  myinput = document.getElementById("yslider");
  myinput.addEventListener('input', changeYcentre);
}
```



4.2 Example 2

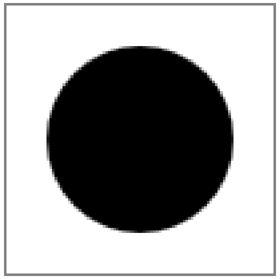
- Use sliders to control circle position

```
function changeXcentre(ev) {
  var xslider = document.getElementById("xslider");
  var x = parseInt(xslider.value)
  var circle = document.getElementById("c1");
  circle.setAttribute("cx", x.toString());
}

function changeYcentre(ev) {
  var yslider = document.getElementById("yslider");
  var y = parseInt(yslider.value)
  var circle = document.getElementById("c1");
  circle.setAttribute("cy", y.toString());
}
```

5. Viewport and aspect ratio

- Initially, both a viewport coordinate system and a user coordinate system that are equivalent are set up for an SVG element
- The viewBox attribute can be used to create a different user coordinate system
- Aspect ratio can also be altered using preserveAspectRatio attribute

```
 <svg xmlns="http://www.w3.org/2000/svg" version="1.1"
       id="example"
       width="100px" height="100px"
       viewBox="0 0 200 200">

  <circle id="c1" cx="100" cy="100" r="70" />

</svg>
```



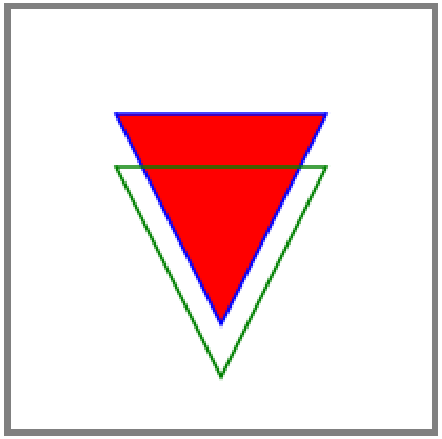
6. Paths

- Paths represent outline of a shape, which can be stroked and filled
- Simple graphics language based on a ‘pen’ and ‘current point’
    - M x y: **move to** absolute point (x,y)
    - m x y: relative **move** from current point (cx, cy) **to** (cx+x, cy+y)
    - L x y: draw **line** from current point **to** absolute point (x,y)
    - l x y: relative lineto
    - V or v: vertical lineto
    - H or h: horizontal lineto
    - C or c: Bezier curve data
    - S or s: smooth curveto data, for joining Bezier curves
    - Z or z: close the current subpath by joining last point to start point with a straight line

6.1 Example 1

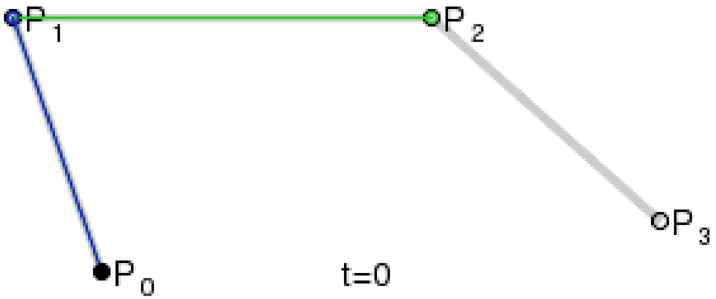
- <path d=“ … ” />

```
<svg ... viewBox="0 0 400 400" ...>
  ...
  <path d="M 100 100 L 300 100 L 200 300 z"
        fill="red" stroke="blue" stroke-width="3" />
  <path d="M 100 150 l 200 0 l -100 200 z"
        fill="none" stroke="green" stroke-width="3" />
</svg>
```

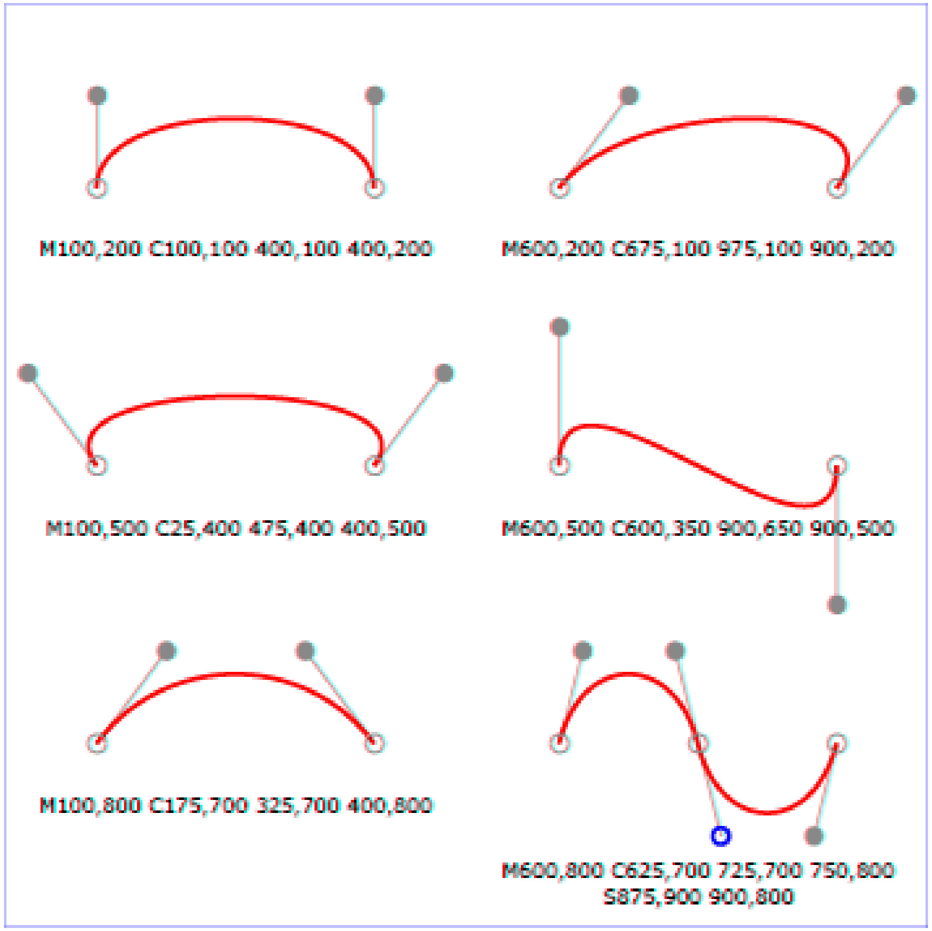


6.2 Bezier curves

- More details: http://www.w3.org/TR/SVG/paths.html



http://en.wikipedia.org – Bezier curves


M100,200 C100,100 400,100 400,200

M600,200 C675,100 975,100 900,200

M100,500 C25,400 475,400 400,500

M600,500 C600,350 900,650 900,500

M100,800 C175,700 325,700 400,800

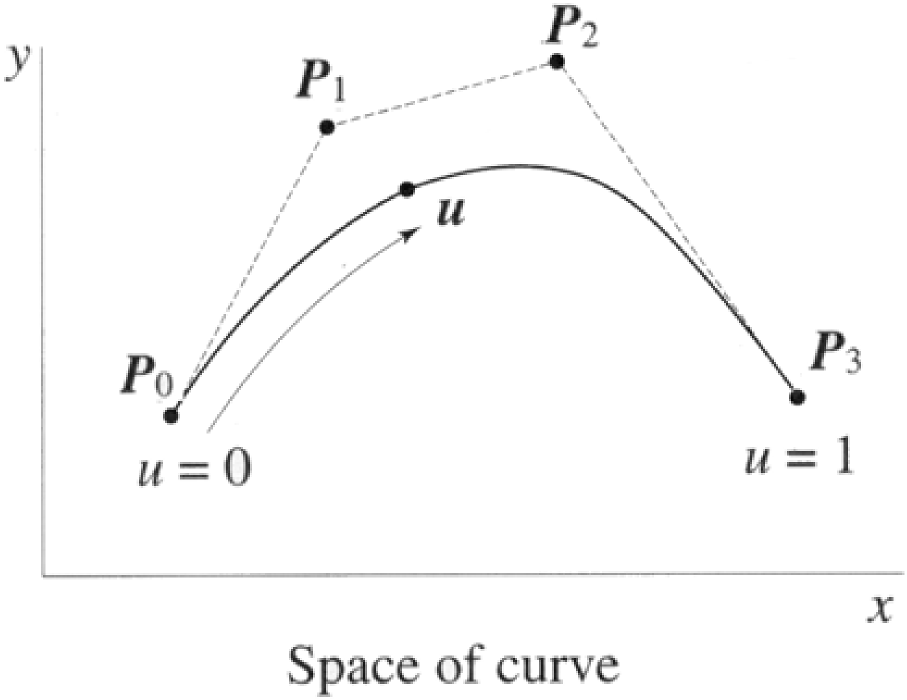
M600,800 C625,700 725,700 750,800
S875,900 900,800

6.2.1 Bezier cubic curves

- $\mathbf{P}_0$ and $\mathbf{P}_3$ are the endpoints of the curve, and $\mathbf{P}_1$ and $\mathbf{P}_2$ control the shape of the curve.

$$\mathbf{Q}(u) = \mathbf{P}_0(1-u)^3 + \mathbf{P}_1 3u(1-u)^2 + \mathbf{P}_2 3u^2(1-u) + \mathbf{P}_3 u^3$$

$$\mathbf{Q}(u) = \sum_{i=0}^{3} \mathbf{P}_i \, B_i(u)$$

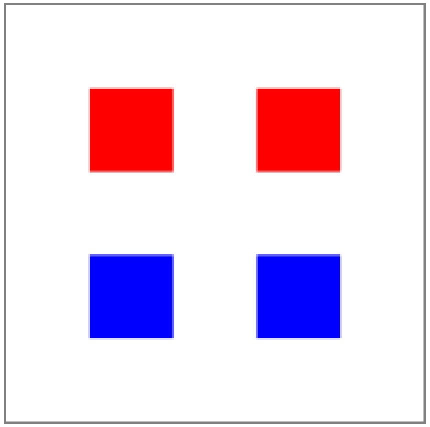


Space of curve

7. The g element

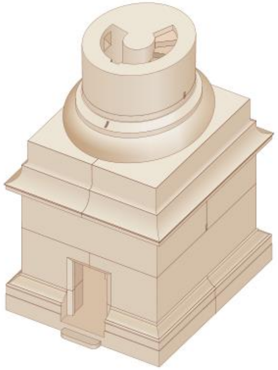
- The `g` element can be used to group other elements together
- Similar to the `div` element in html

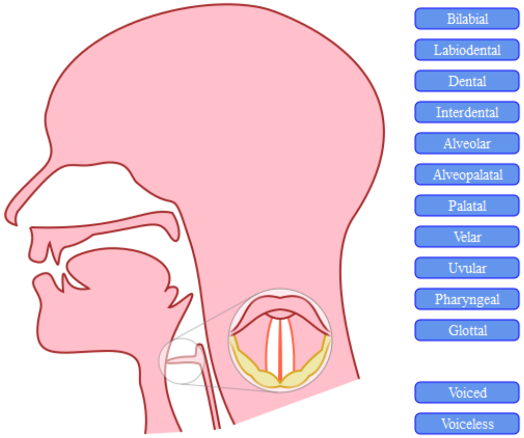
```
<svg xmlns="http://www.w3.org/2000/svg" versio
     width="5cm" height="5cm" >
  <desc>Two groups, each of two rectangles
  </desc>
  <g id="group1" fill="red" >
    <rect x="1cm" y="1cm" width="1cm" height="1cm" />
    <rect x="3cm" y="1cm" width="1cm" height="1cm" />
  </g>
  <g id="group2" fill="blue" >
    <rect x="1cm" y="3cm" width="1cm" height="1cm" />
    <rect x="3cm" y="3cm" width="1cm" height="1cm" />
  </g>
</svg>
```



http://www.w3.org/TR/SVG/struct.html#GElement

8. SVG examples





http://upload.wikimedia.org/wikipedia/commons/6/6c/Trajans-Column-lower-animated.svg
By Hk kng (own work) [CC-BY-SA-3.0 (www.creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

http://www1.plurib.us/svg_gallery/

http://svg-whiz.com/svg/linguistics/theCreepyMouth.svg

9. SVG versus canvas

SVG

- Shapes have DOM support. They can have event handlers bound directly to them
- When lots of objects, DOM is slow relative to canvas
- Raphaël JavaScript library useful when dealing with older browsers (http://raphaeljs.com/)

canvas

- Made up of pixels; event handler is at the level of the overall canvas
  - The shapes do not 'exist'. They are just sets of pixels
  - Remember where shapes were drawn in a separate data structure
  - Then, check mouse coordinates for containment against each item in this data structure
- Canvas is suitable for fast games and animation of lots of objects

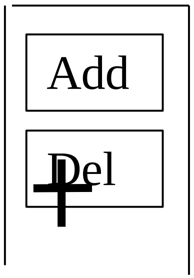
Detailed comparison: http://www.svgopen.org/2009/papers/54-SVG_vs_Canvas_on_Trivial_Drawing_Application/

9.1 Containment and closest object tests

- Containment
  - Collection of regions is tested to determine which one the cursor is in
  - Often used for picking from a menu

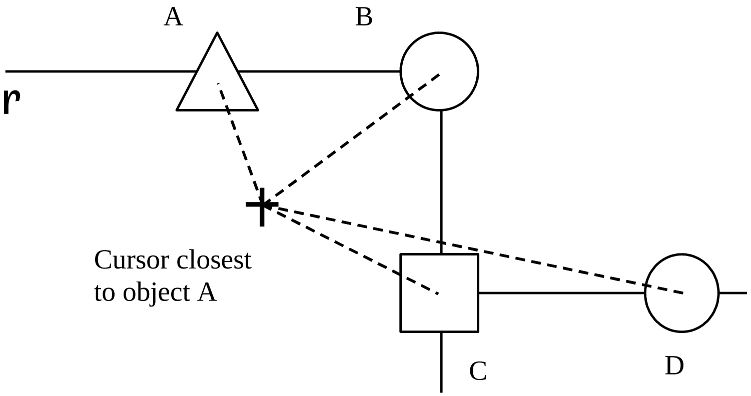
```
inside = (x<delBox.xmax)
         &&(x>delBox.xmin)
         &&(y<delBox.ymax)
         &&(y>delBox.ymin);
```

Cursor inside box



- Minimum distance
  - Calculate Euclidean distance from cursor to centre of each of a list of objects.

$$\text{distanceSquared} = (px-obj\mathit{i}.x)^2 + (py-obj\mathit{i}.y)^2$$



Cursor closest to object A

10. Summary

- Raster graphics
  - Screen image is made of a rectangular grid of pixels – bitmap
  - HTML5 canvas element
- Vector graphics
  - Image is a list of geometric shapes, together with stroke and fill information
  - SVG is the W3C's recommended markup language for vector graphics
  - https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial
- Interaction
  - Can use JavaScript, buttons, keyboard events and mouse events to make both canvas and SVG interactive
- CSS3 and animation (of any elements, including HTML elements):
  - http://www.smashingmagazine.com/2011/05/an-introduction-to-css3-keyframe-animations/

Appendix A. Alternative ways to include SVG

- Using an `object` element

```
<body>...
<object id="svgExample" type="image/svg+xml" width="500"
        height="500" data="svg1c.svg">
        <span>Text to display if a problem occurs</span>
</object>
...</body>
```

- Or as the background image for an element:

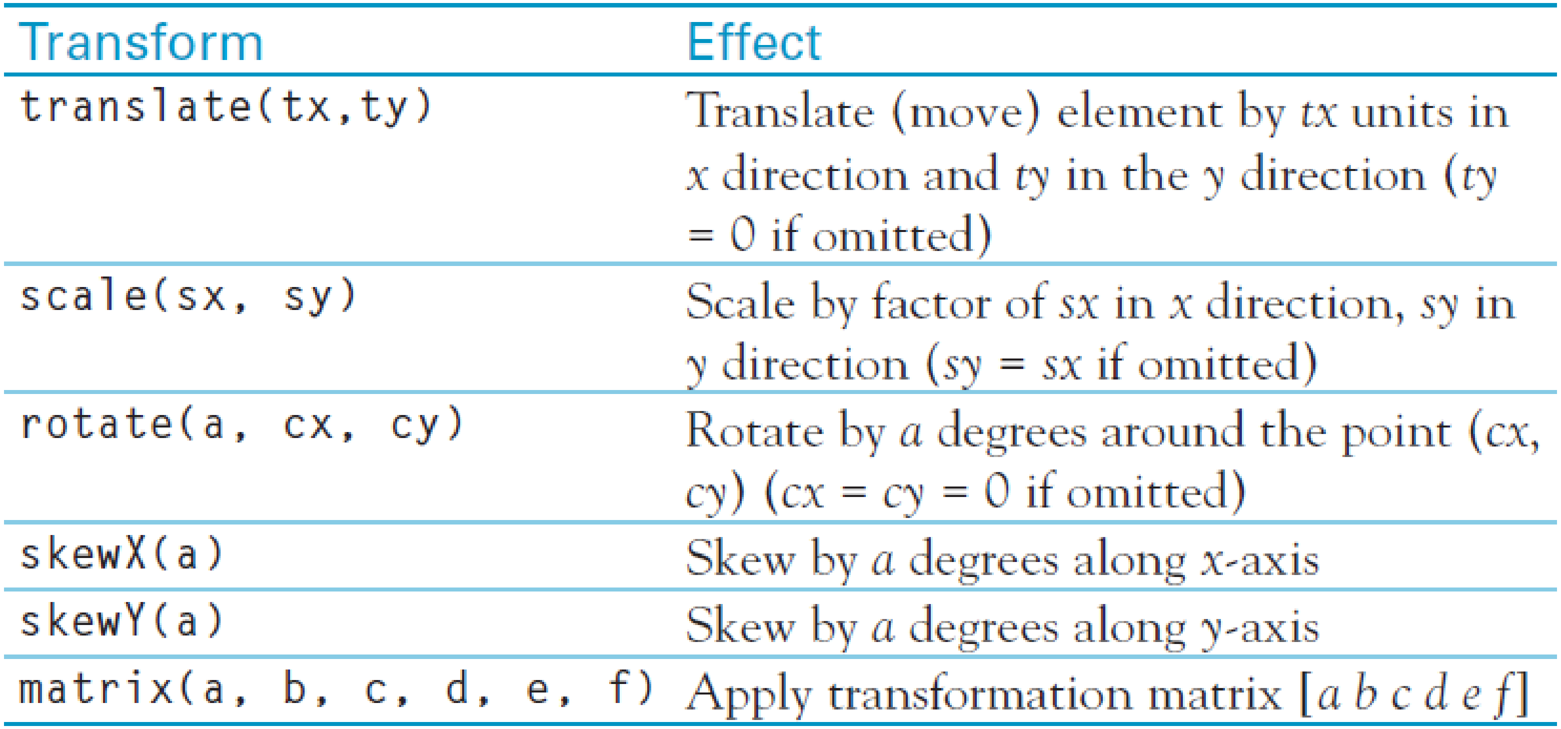
```
body {
 background-image: url(svg1c.svg);
 background-repeat: no-repeat;
}
```

- Or as an img

```
<img src = "svg1c.svg" alt="description" />
```

Appendix B. Transformations

- Transformations are represented as the value of the `transform` attribute.

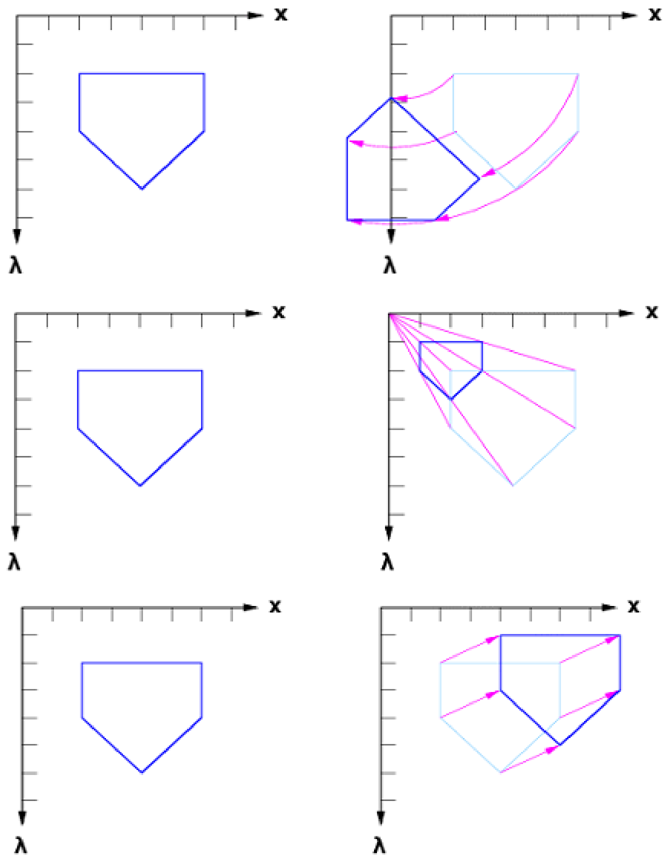


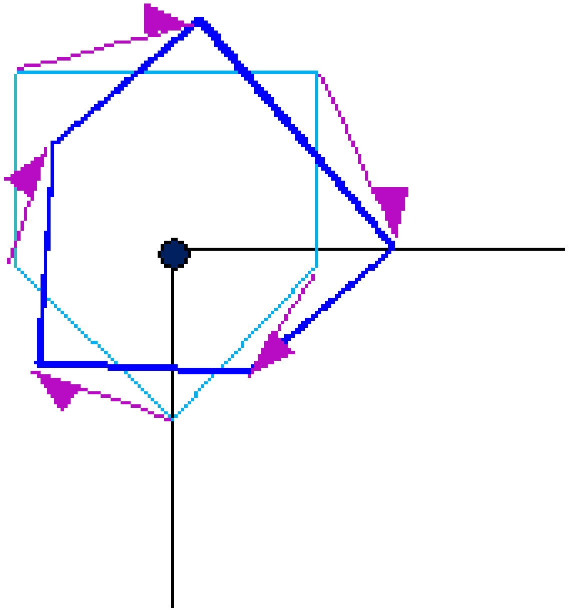
Transform	Effect
`translate(tx,ty)`	Translate (move) element by *tx* units in *x* direction and *ty* in the *y* direction (*ty* = 0 if omitted)
`scale(sx, sy)`	Scale by factor of *sx* in *x* direction, *sy* in *y* direction (*sy* = *sx* if omitted)
`rotate(a, cx, cy)`	Rotate by *a* degrees around the point (*cx*, *cy*) (*cx* = *cy* = 0 if omitted)
`skewX(a)`	Skew by *a* degrees along *x*-axis
`skewY(a)`	Skew by *a* degrees along *y*-axis
`matrix(a, b, c, d, e, f)`	Apply transformation matrix [*a b c d e f*]

Chapman, N and J. Chapman, Web Design: A complete introduction, John Wiley & Sons, 2006.

B. Transformations

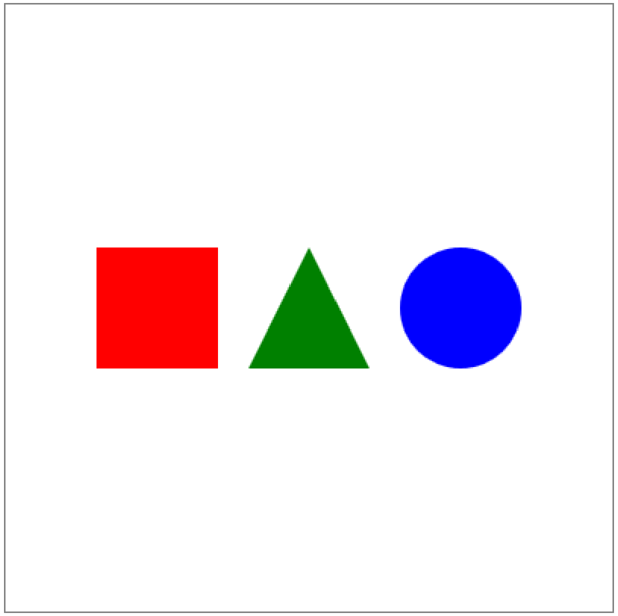
- Transformations are about the origin (0,0)





B.1 Transformations and the g element

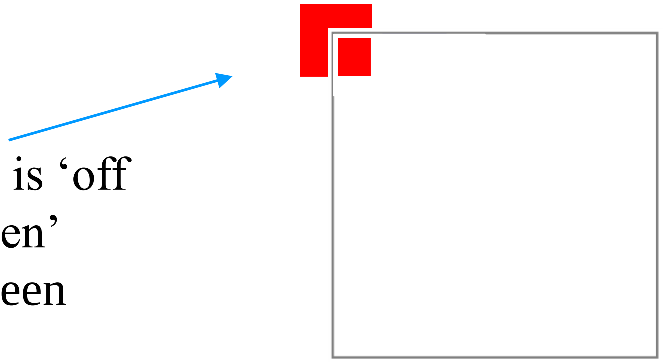
- This is what we want to achieve →
- We could do this by drawing the elements in the correct position
- However, we can also achieve it by drawing all the elements in the same initial position and then using transformations to position them elsewhere on the screen
    - This is more flexible for complex scenes
- Use g element to group transformations for an object



B.1 Transformations and the g element

- Draw a red square ‘centred on the origin’

```
<rect fill="red" x="-40" y="-40" width="80" height="80" />
```

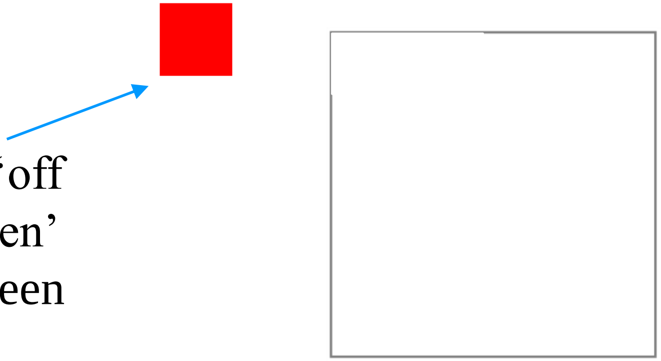


This bit is ‘off the screen’ and unseen

B.1 Transformations and the g element

- Use g element to group transformations for an object

```
<g transform="translate(-100,0)">
  <rect fill="red" x="-40" y="-40" width="80" height="80" />
</g>
```

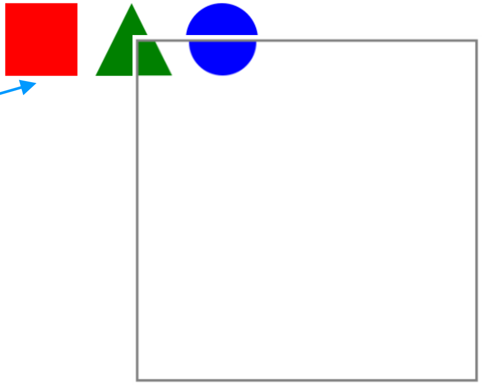


This is ‘off the screen’ and unseen

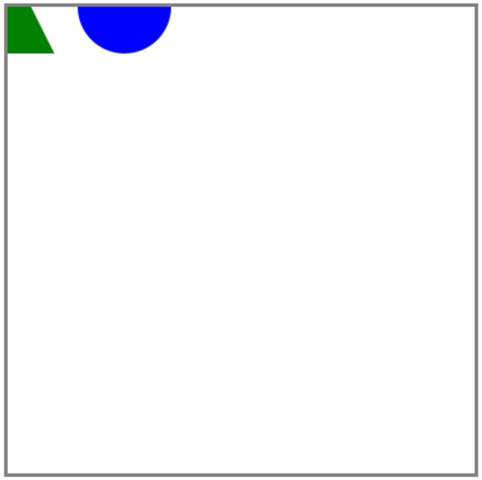
B.1 Transformations and the g element

- Use g element to group transformations for an object

```
<g transform="translate(-100,0)">
  <rect fill="red" x="-40" y="-40" width="80" height="80" />
</g>
<g transform="translate(0,0)">
  <polygon fill="green" points="-40,40 0,-40 40,40" />
</g>
<g transform="translate(100,0)">
  <circle fill="blue" cx="0" cy="0" r="40" />
</g>
```



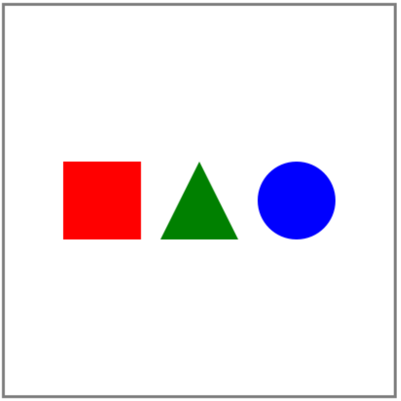
These bits are
'off the screen'



B.2 Nested transformations

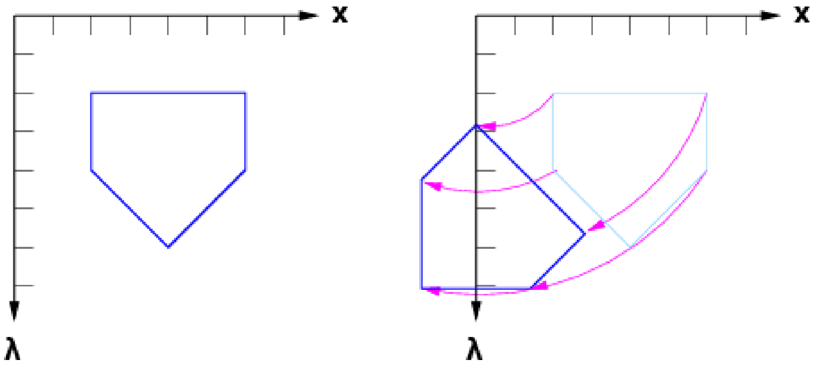
- Use g element to nest transformations
- Transformations applied in reverse order they are specified

```
<g transform="translate(200,200)">
  <g transform="translate(-100,0)">
    <rect fill="red" x="-40" y="-40" width="80" height="80" />
  </g>
  <g transform="translate(0,0)">
    <polygon fill="green" points="-40,40 0,-40 40,40" />
  </g>
  <g transform="translate(100,0)">
    <circle fill="blue" cx="0" cy="0" r="40" />
  </g>
</g>
```



B.2 Nested transformations

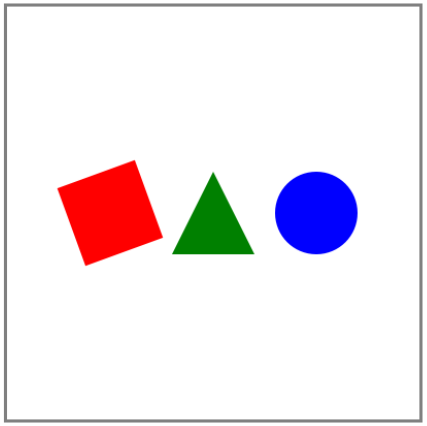
- Order of transformations matters.
- Consider rotation:
  - Rotation matrix rotates points about the origin of the coordinate system



B.2 Nested transformations

- Order of transformations matters

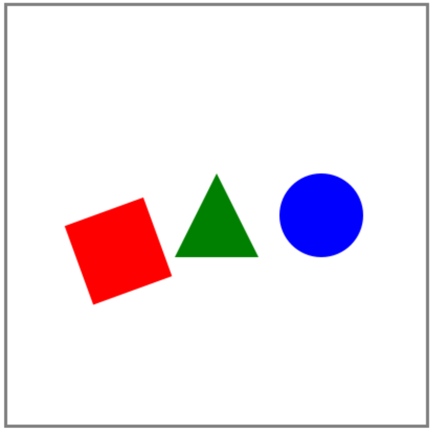
```
<g transform="translate(200,200)">
  <g transform="translate(-100,0) rotate(-20,0,0)">
    <rect fill="red" x="-40" y="-40" width="80" height="80" />
  </g>
  <g transform="translate(0,0)">
    <polygon fill="green" points="-40,40 0,-40 40,40" />
  </g>
  <g transform="translate(100,0)">
    <circle fill="blue" cx="0" cy="0" r="40" />
  </g>
</g>
```



B.2 Nested transformations

- Order of transformations matters
  - Matrix multiplication is not commutative

```
<g transform="translate(200,200)">
  <g transform="rotate(-20,0,0) translate(-100,0)">
    <rect fill="red" x="-40" y="-40" width="80" height="80" />
  </g>
  <g transform="translate(0,0)">
    <polygon fill="green" points="-40,40 0,-40 40,40" />
  </g>
  <g transform="translate(100,0)">
    <circle fill="blue" cx="0" cy="0" r="40" />
  </g>
</g>
```



B.3 Transformations as matrices

- Position: P (x,y,1) (homogenous coordinates of a 2D point)
- New position: Q (x', y', 1)
- Transformation matrix M

$$Q = MP \quad \text{where} \quad \mathrm{M} = \begin{pmatrix} \mathrm{a} & \mathrm{b} & \mathrm{t_x} \\ \mathrm{c} & \mathrm{d} & \mathrm{t_y} \\ 0 & 0 & 1 \end{pmatrix}$$

$$\mathrm{M} = \begin{pmatrix} \mathrm{s_x} & 0 & 0 \\ 0 & \mathrm{s_y} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

scale

$$\mathrm{M} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

rotate (anti-clockwise)

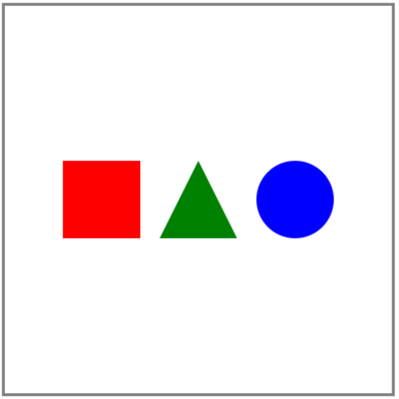
$$\mathrm{M} = \begin{pmatrix} 1 & 0 & \mathrm{t_x} \\ 0 & 1 & \mathrm{t_y} \\ 0 & 0 & 1 \end{pmatrix}$$

translate

B.3 Transformations as matrices

- Space: (0,0) to (400,400)
- M_1 = **translate(200,200)**; M_2 = translate(-100,0); M_3 = translate(0,0); M_4 = translate(100,0);
- To draw red square: Q = $\mathbf{M_1}M_2P$
- Green triangle: Q = $\mathbf{M_1}M_3P$
- Blue circle: Q = $\mathbf{M_1}M_4P$

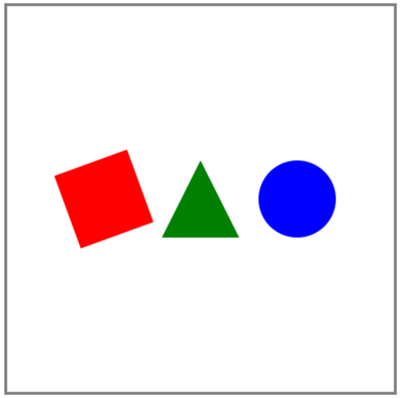
```
<g transform="translate(200,200)">
  <g transform="translate(-100,0)">
    <rect fill="red" x="-40" y="-40" width="80" height="80" />
  </g>
  <g transform="translate(0,0)">
    <polygon fill="green" points="-40,40 0,-40 40,40" />
  </g>
  <g transform="translate(100,0)">
    <circle fill="blue" cx="0" cy="0" r="40" />
  </g>
</g>
```



B.3 Transformations as matrices

- Space: (0,0) to (400,400)
- M_1 = **translate(200,200)**; M_2 = translate(-100,0); M_3 = translate(0,0); M_4 = translate(100,0); M_5 = rotate(-20,0,0);
- To draw red square: Q = $M_1M_2M_5P$
- Green triangle: Q = M_1M_3P
- Blue circle: Q = M_1M_4P

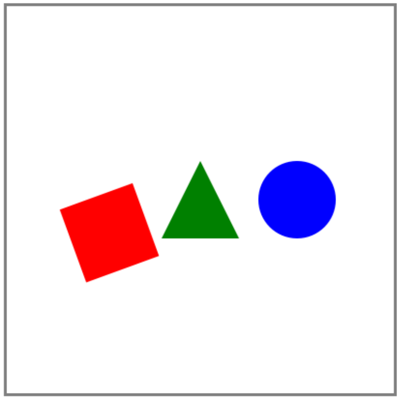
```
<g transform="translate(200,200)">
  <g transform="translate(-100,0) rotate(-20,0,0)">
    <rect fill="red" x="-40" y="-40" width="80" height="80" />
  </g>
  <g transform="translate(0,0)">
    <polygon fill="green" points="-40,40 0,-40 40,40" />
  </g>
  <g transform="translate(100,0)">
    <circle fill="blue" cx="0" cy="0" r="40" />
  </g>
</g>
```



B.3 Transformations as matrices

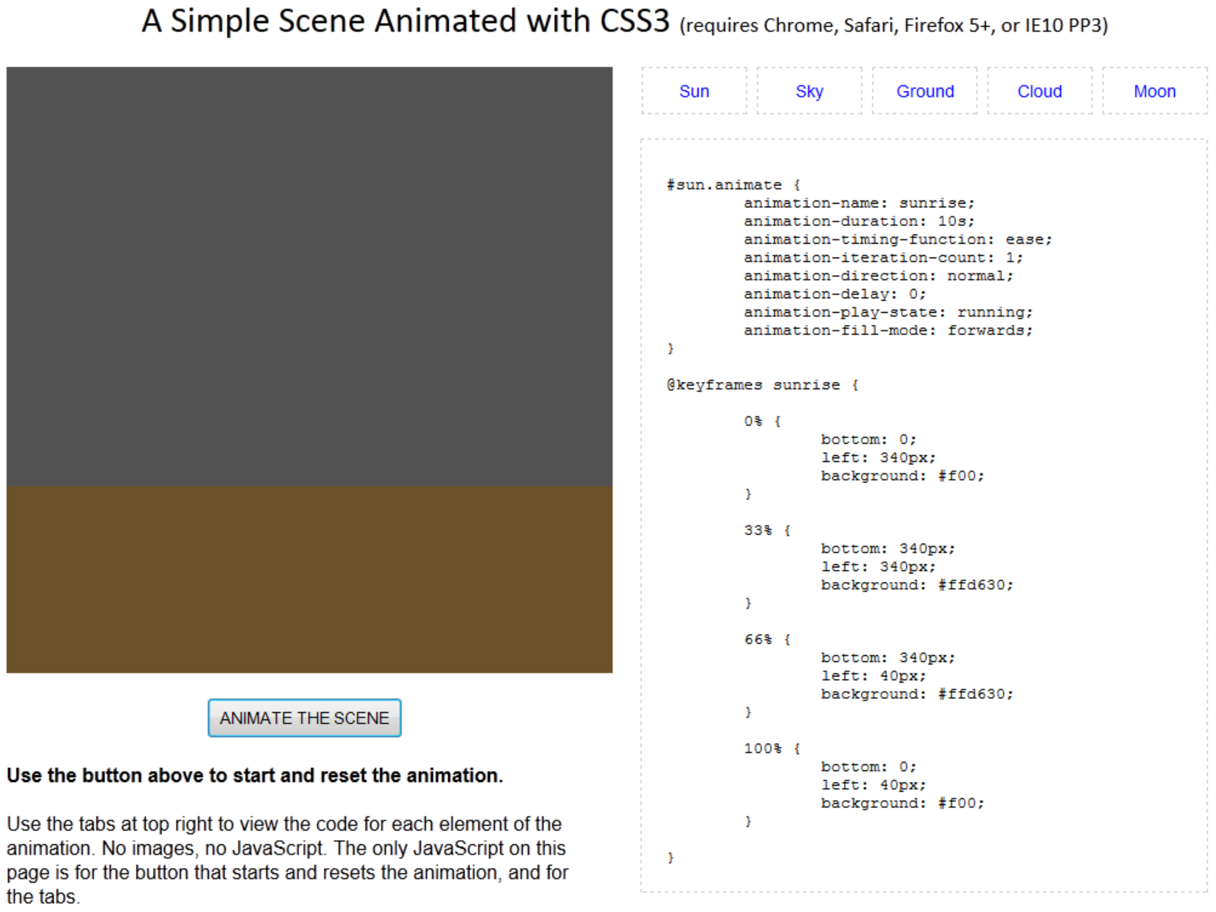
- Space: (0,0) to (400,400)
- M_1 = **translate(200,200)**; M_2 = translate(-100,0); M_3 = translate(0,0); M_4 = translate(100,0); M_5 = rotate(-20,0,0);
- To draw red square: Q = $M_1M_5M_2P$
- Green triangle: Q = M_1M_3P
- Blue circle: Q = M_1M_4P

```
<g transform="translate(200,200)">
  <g transform="rotate(-20,0,0) translate(-100,0)">
    <rect fill="red" x="-40" y="-40" width="80" height="80" />
  </g>
  <g transform="translate(0,0)">
    <polygon fill="green" points="-40,40 0,-40 40,40" />
  </g>
  <g transform="translate(100,0)">
    <circle fill="blue" cx="0" cy="0" r="40" />
  </g>
</g>
```



Appendix C: CSS Animation

- http://www.impressivewebs.com/demo-files/css3-animated-scene/



A Simple Scene Animated with CSS3 (requires Chrome, Safari, Firefox 5+, or IE10 PP3)

```
#sun.animate {
        animation-name: sunrise;
        animation-duration: 10s;
        animation-timing-function: ease;
        animation-iteration-count: 1;
        animation-direction: normal;
        animation-delay: 0;
        animation-play-state: running;
        animation-fill-mode: forwards;
}

@keyframes sunrise {

        0% {
                bottom: 0;
                left: 340px;
                background: #f00;
        }

        33% {
                bottom: 340px;
                left: 340px;
                background: #ffd630;
        }

        66% {
                bottom: 340px;
                left: 40px;
                background: #ffd630;
        }

        100% {
                bottom: 0;
                left: 40px;
                background: #f00;
        }

}
```

Use the button above to start and reset the animation.

Use the tabs at top right to view the code for each element of the animation. No images, no JavaScript. The only JavaScript on this page is for the button that starts and resets the animation, and for the tabs.