ROMA TRE
UNIVERSITÀ DEGLI STUDI

iNGEGNERIA
CIVILE, INFORMATICA E DELLE
TECNOLOGIE AERONAUTICHE

# Information Visualization

## Infovis on the Web

## Graphics

*G. Da Lozzo, V. Di Donato, M. Patrignani*

# Copyright notice

# Credits

- Parts of this matherial is inspired by
  - http://www.teaching-materials.org/
  - https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial

# Raster Vs Vector graphics

- Raster graphics
  - based on pixels (picture element) arranged in a grid (= bitmaps)
    - a pixel represents the smallest unit of a video image that has specific RGBA values
  - lower abstraction level
- Vector graphics
  - based on geometrical primitives such as points, lines, curves and polygon(s)
    - mathematical expressions
  - more abstract level

# Raster properties

- Resolution
  - number of pixels to represent the image
- Color depth
  - number of bits to color a single pixel
  - typically 24-bits for RGB
    - eight bits for each component
    - $2^8$ = 256 shades per component
    - $256^3$ ~ 16 million colors
  - humans distinguish ~ 10 mil colors [Judd ‘75]
  - often 8 more bits are used for opacity

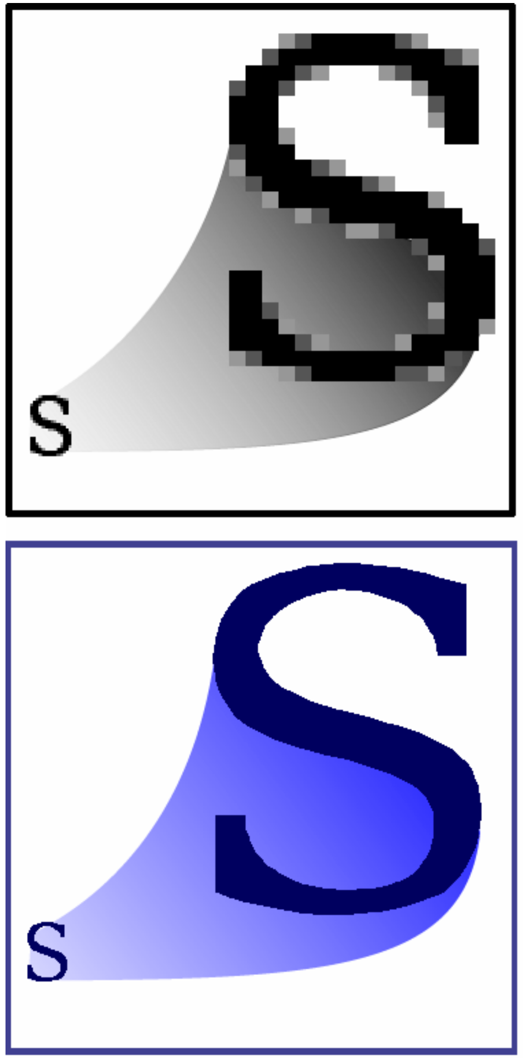
# Vector graphic properties

- Vector graphics does not keep track of each pixel
  - it only considers the mathematical information that is able to generate the picture
- It is based on shapes (ultimately on paths)
  - paths are defined by starting and ending points, together with other control points, curves, and angles along the way
  - paths are associated with styles
    - fill color, stroke-width, etc.

# Raster vs Vector: pros and cons

- Bitmaps
  - fine-grained control up to single pixels
  - best for full-color images, like photographs
  - increasing resolution or color depth affects their size
  - do not resize well

- Vector graphics
  - resolution independency
  - resize with little or no loss
  - no extremely complex images
  - time and talent needed to create it



[Image from Wikipedia.org]

# HTML5 Canvas and SVG

- HTML5 Canvas element
  - “provides scripts with a resolution-dependent bitmap canvas, which can be used for rendering graphs, game graphics, or other visual images on the fly” [W3C]
- SVG (Scalable Vector Graphics)
  - “is an XML-based format for describing 2D vector graphics” [W3C]

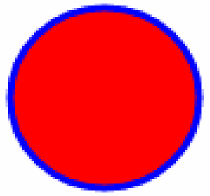
# High level summary

| HTML5 Canvas | SVG |
|---|---|
| Pixel based | Shape based |
| Single HTML element | Multiple graphical elements, which become part of the DOM |
| Modified through script only | Modified through script and CSS |
| Interaction is granular (x,y) | Interaction is abstracted (rect, path,…) |
| Performance is better with smaller surface, a larger number of objects (>10K), or both | Performance is better with larger surface, a smaller number of objects (<10K), or both |
| Stateless (next slide) | Statefull (next slide) |

# Stateless and stateful paradigms

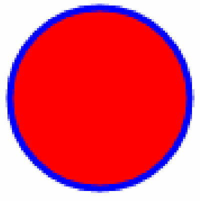
- Canvas is stateless
  - drawing primitives change the color of a pixel
    - irrespectively of previous drawing instructions
- SVG is stateful
  - drawing primitives change the properties of an object
    - which has been defined earlier
  - this implies easier animation and interaction

# A circle in Canvas



```html
<canvas id="myCanvas" width="100" height="100"></canvas>
```

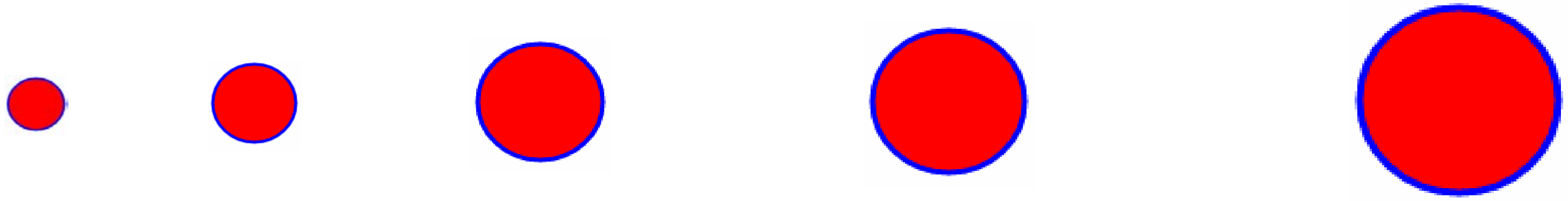
```html
<script>
function drawCircle(radius) {
  var canvas = document.getElementById('myCanvas');
  var context = canvas.getContext('2d');
  var centerX = canvas.width / 2;
  var centerY = canvas.height / 2;
  context.clearRect(0, 0, canvas.width, canvas.height);
  context.beginPath();
  context.arc(centerX, centerY, radius, 0, 2 *Math.PI);
  context.fillStyle = 'red';
  context.fill();
  context.lineWidth = 3;
  context.strokeStyle = 'blue';
  context.stroke();
}
drawCircle(40);
</script>
```

# A cycle in SVG 

```
<svg height="100"
     width="100“
     xmlns="http://www.w3.org/2000/svg">
  <circle cx="50"
          cy="50"
          r="40"
          stroke="blue"
          stroke-width="3"
          fill="red">
  </circle>
</svg>
```

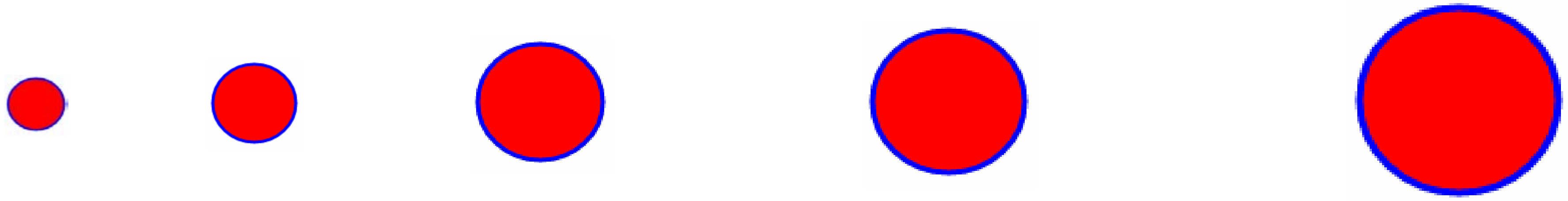
# A simple animation with Canvas

```
<script>
var i = 10;
setInterval(
  function(){
        drawCircle(i);
        if(i < 40) i++;
    }, 25);
</script>
```



# A simple animation with SVG

```
<script>
var i = 10;
setInterval(
  function(){
       document.getElementById("myCircle")
               .setAttribute("r", i);
       if(i < 40) i++;
    }, 25);
</script>
```

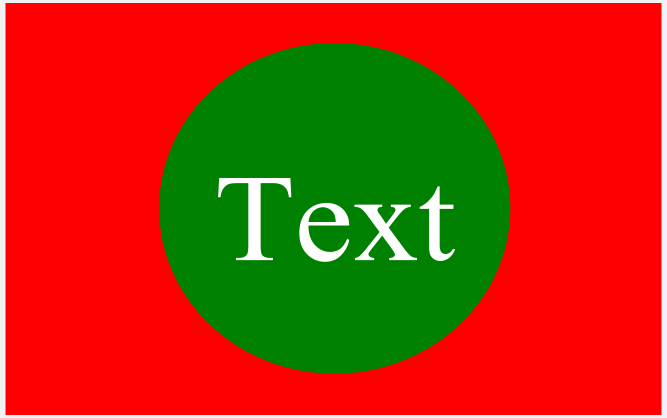


# SVG: basic example

- Objects are rendered in the order in which they are listed

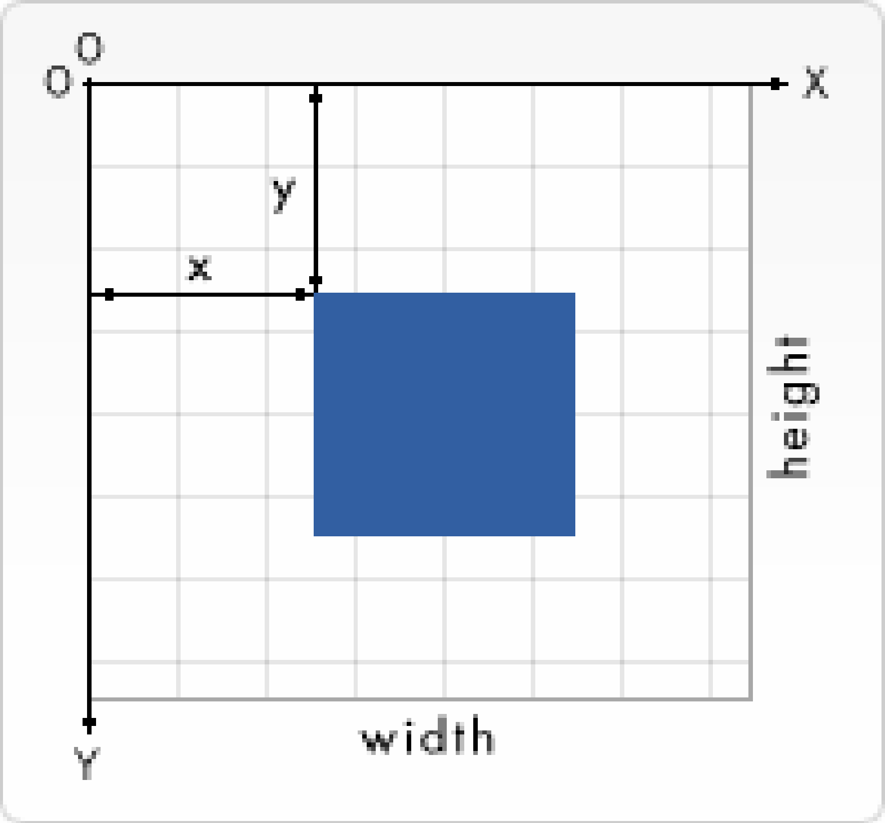
```
<svg width="300"
     height="200"
     xmlns="http://www.w3.org/2000/svg">

<rect width="100%" height="100%" fill="red" />
<circle cx="150" cy="100" r="80" fill="green" />
<text x="150"
      y="125"
      font-size="60"
      text-anchor="middle"
      fill="white"> Text
</text>
</svg>
```



# Positions

- SVG uses a grid system
- The top left corner of the document is (0,0)
- Positions are measured in **pixels** with
  - positive x direction to the right → +x
  - positive y direction to the bottom ↓ +y
- One pixel in SVG maps to one pixel on the output device

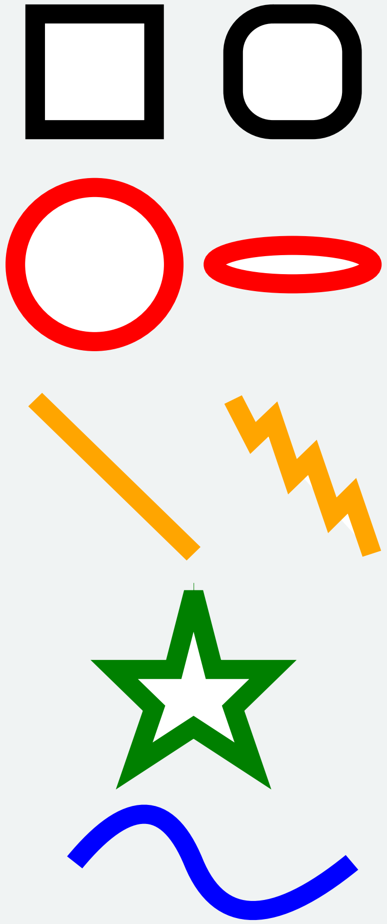


# Basic shapes

```
<svg width="200" height="250" version="1.1" xmlns="http://www.w3.org/2000/svg">
   <rect x="10" y="10" width="30" height="30" stroke="black" fill="transparent"
      stroke-width="5"/>
   <rect x="60" y="10" rx="10" ry="10" width="30" height="30" stroke="black"
      fill="transparent" stroke-width="5"/>
   <circle cx="25" cy="75" r="20" stroke="red" fill="transparent"
        stroke-width="5"/>
   <ellipse cx="75" cy="75" rx="20" ry="5" stroke="red" fill="transparent"
         stroke-width="5"/> orange
   <line x1="10" x2="50" y1="110" y2="150" stroke="" fill="transparent"
      stroke-width="5"/>
   <polyline points="60 110, 65 120, 70 115, 75 130, 80 125, 85 140, 90 135,
          95 150" stroke="orange" fill="transparent" stroke-width="5"/>
   <polygon points="50 160 55 180 70 180 60 190 65 205 50 195 35 205 40 190 30
         180 45 180" stroke="green" fill="transparent" stroke-width="5"/>

   <path d="M20,230 Q40,205 50,230 T90,230" fill="none" stroke="blue"
      stroke-width="5"/>

</svg>
```



SVG Path Commands

[MDNP, https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Paths]

# Paths: Line Commands

- M x y: “move to”
  - it does not draw anything
  - it moves the cursor
- L x y: “line to”
  - it draws a line from the current position to (x, y)
- H x or V y: “horizontal to” or “vertical to”
  - it draws horizontal and vertical lines
- Z: “close path”
  - it draws a line from the current position
    back to the first point

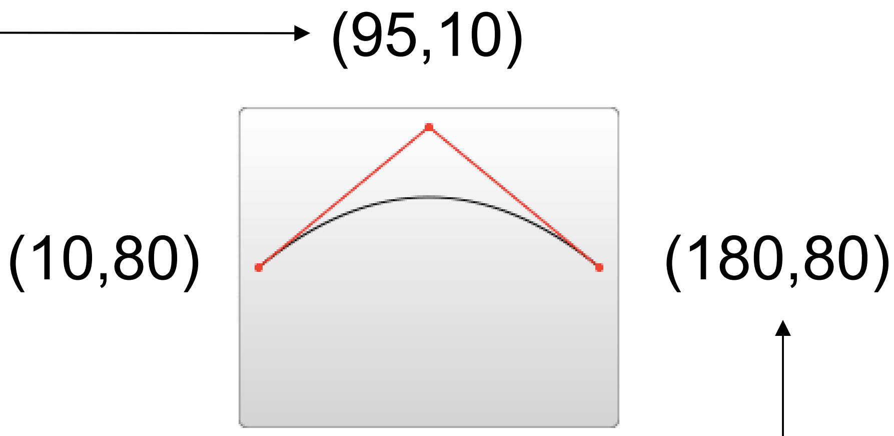
COORDINATES:

up case → absolute<br>
lw case → relatives

# Paths: quadratic bezier curves (Q)

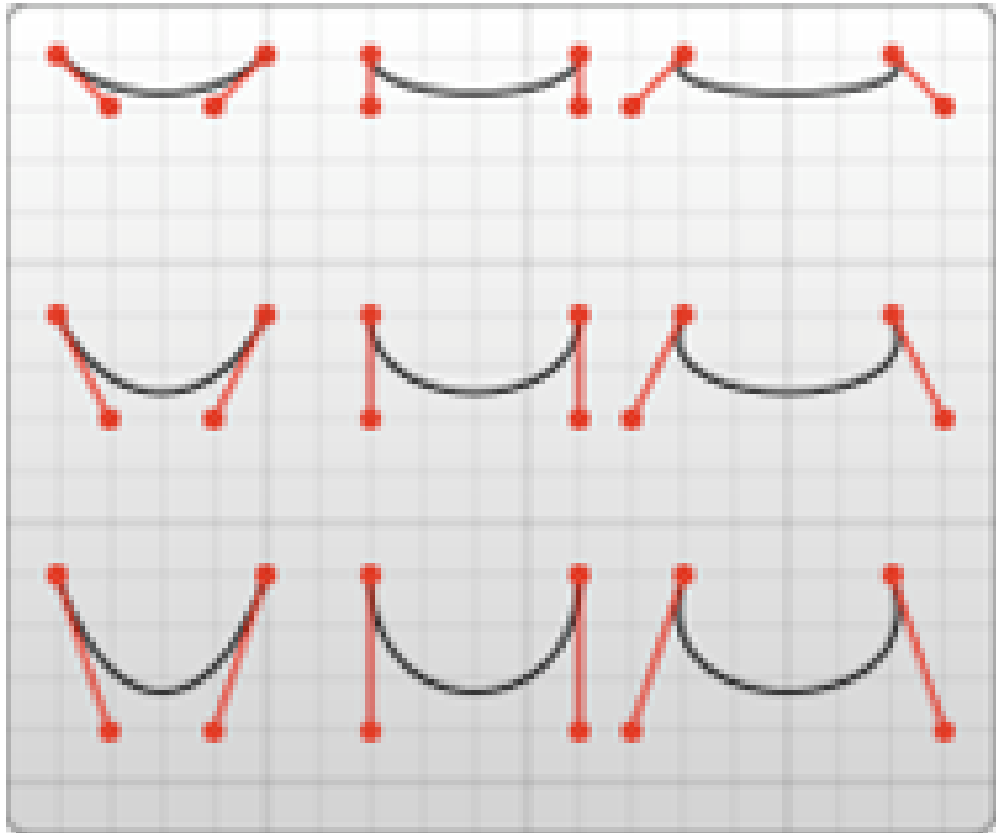
- Q x1 y1, x y
- It requires one control point
  - which determines the slope of the curve at both the start point and at the end point
- It is simpler than the cubic one

```
<path d="M10 80 Q 95 10, 180 80"
stroke="black"
fill="transparent"/>
```



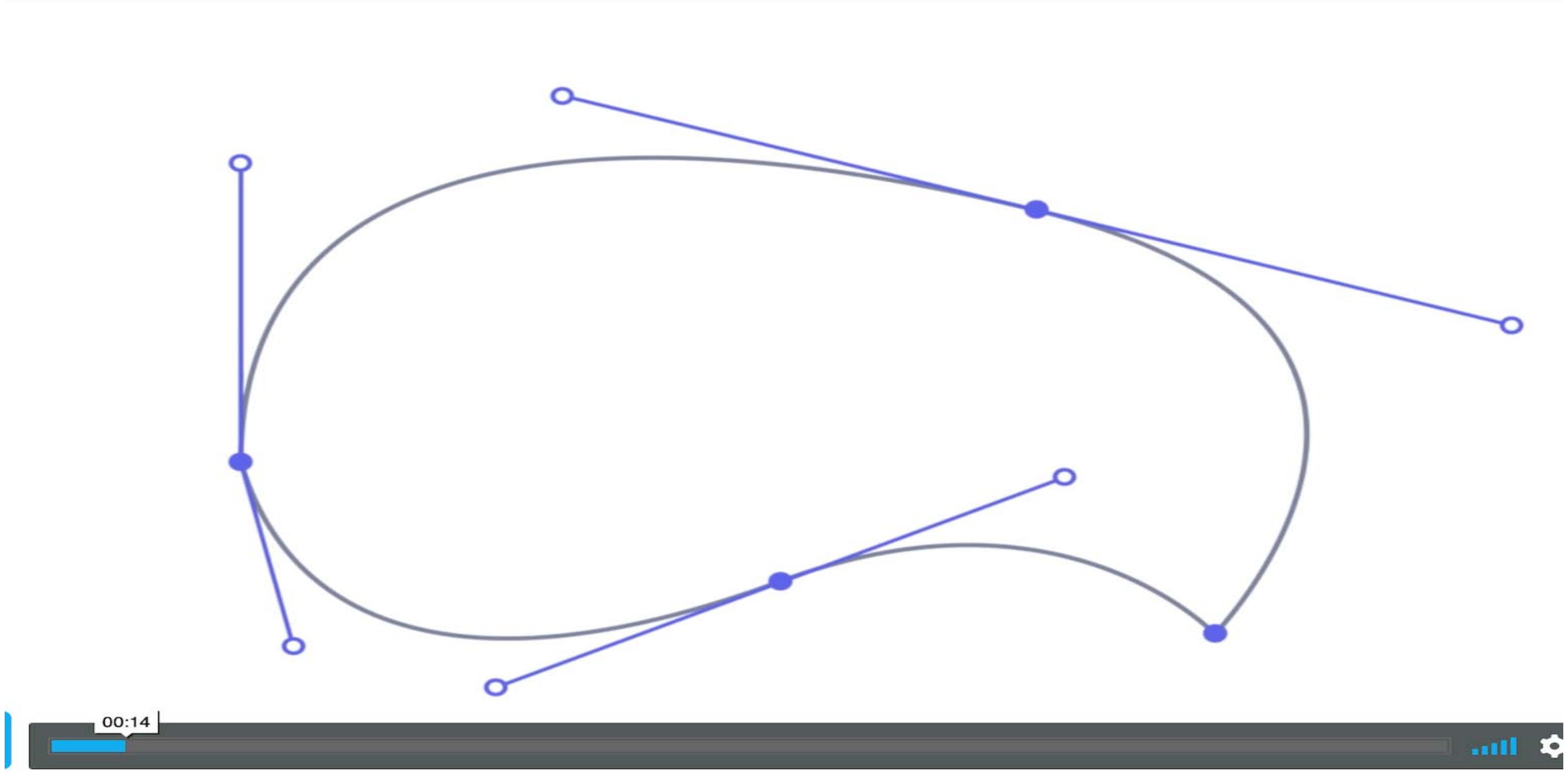
# Paths: cubic bezier curves (C)

- C x1 y1, x2 y2, x y
- It requires two control points
  - which determines the slope of the curve at both the start point and at the end point



```
<path d="M10 10 C 20 20, 40 20, 50 10"
stroke="black” fill="transparent"/>
```

# Cubic Bezier Curves
# Under the Hood

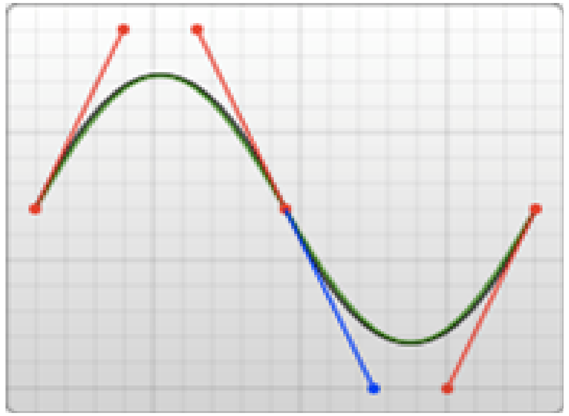


https://vimeo.com/106757336

# Paths: compose curves

- S x2 y2, x y
- You can string together several Bezier curves to create extended smooth shapes
    - the control point on one side of a point will be a reflection of the control point used on the other side to keep the slope constant
    - in this case, you can use a shortcut version of the cubic Bezier

```
<path d="M10 80 C 40 10, 65 10, 95 80 S 150 150,
180 80” stroke="black" fill="transparent"/>
```



# Elliptical Arcs (1/9)

- Arcs are sections of circles or ellipses
  - A rx ry, x-axis-rotation, l-flag, sweep-flag, x y
    - rx = x-radius
    - ry = y-radius
    - x-axis-rotation = clockwise rotation of the ellipse (degrees)
    - l-flag = 0/1

      0 means “use one of the two smaller arcs”

      1 means “use one of the two larger arcs”
    - sweep-flag = 0/1

      0 means “counter-clockwise”

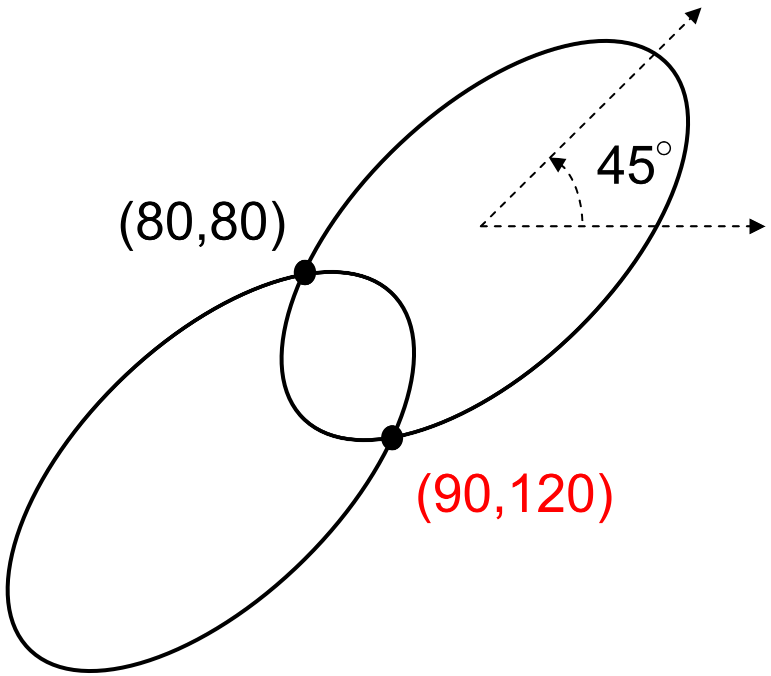
      1 means “clockwise”
    - x y = coordinates of the final point

# Elliptical Arcs (2/9)

- Given rx, ry, and x-axis-rotation there are two ellipses that can connect the starting and ending points

```
<path d="M80 80 A 50 20, 45, ..., 90 120” stroke="black"/>
```

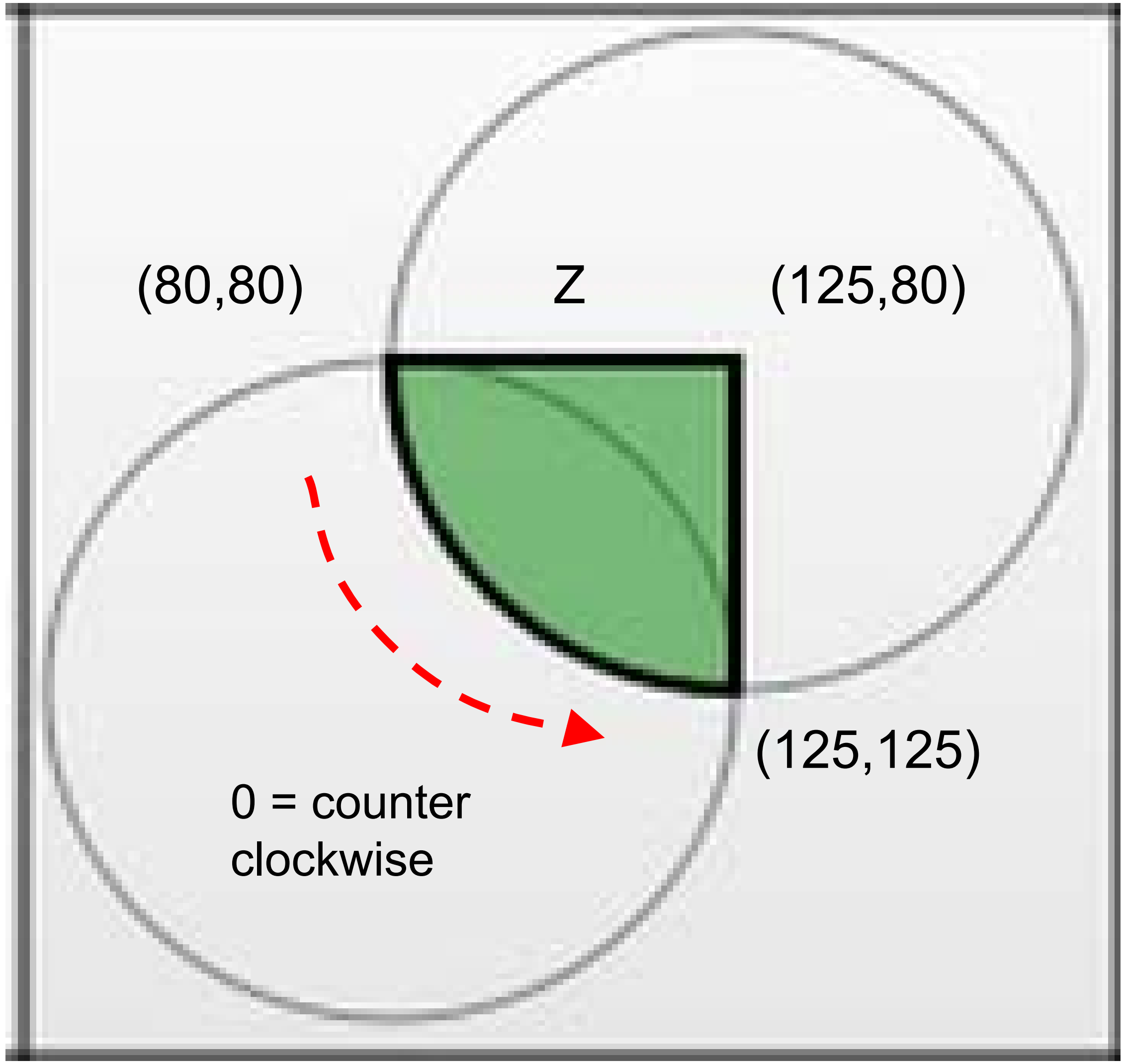
- Along each one of these two ellipses there are two possible arcs that you can use
    - overall, you have four different arcs



# Elliptical Arcs (3/9)

```
<path d="M80 80 A 45 45, 0,
0, 0, 125 125 L 125 80 Z"
fill="green"/>
```

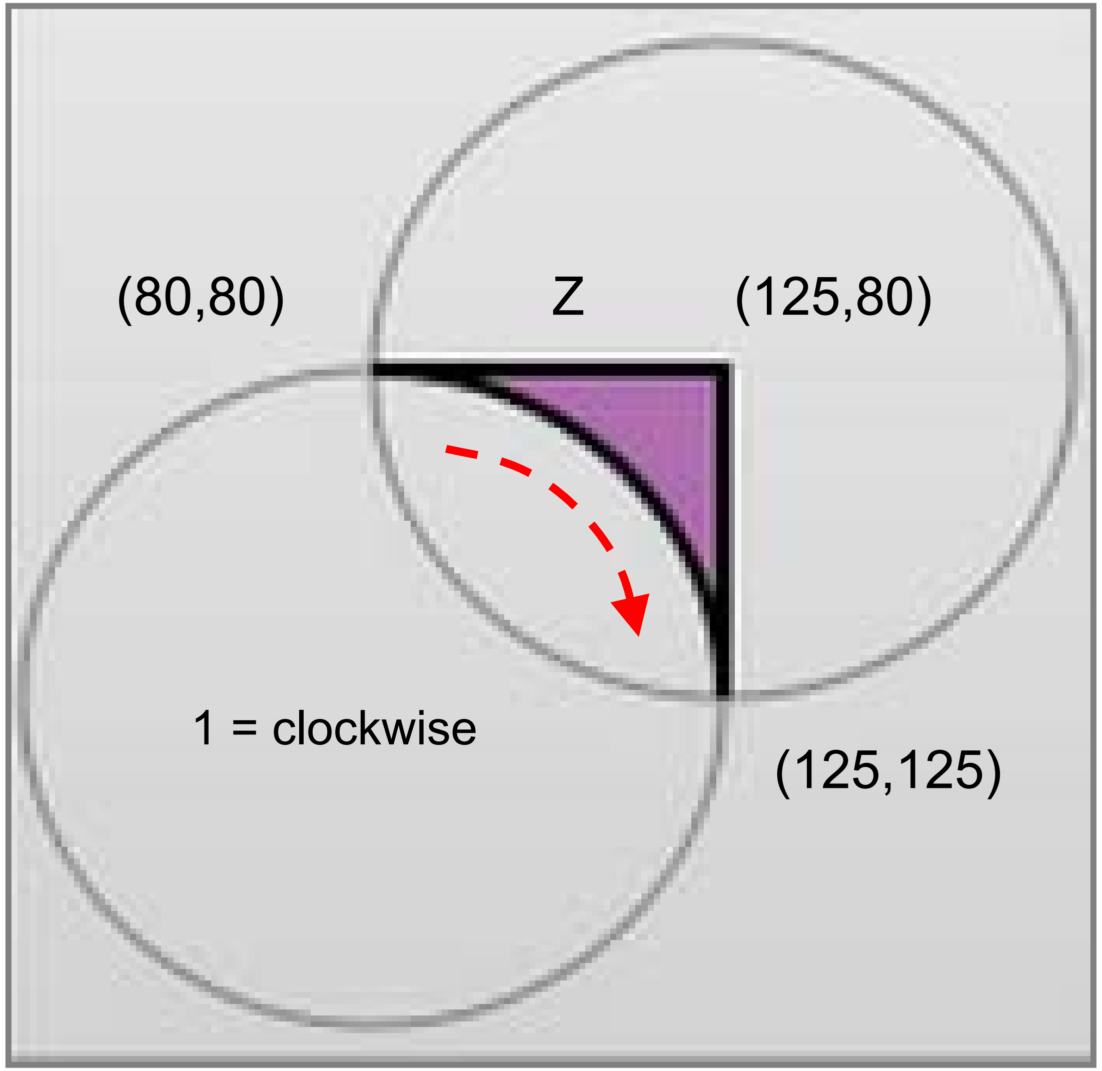
- l-flag = 0/1
  - 0 means “use one of the two smaller arcs”
  - 1 means “use one of the two larger arcs”
- sweep-flag = 0/1
  - 0 means “counter-clockwise”
  - 1 means “clockwise”



# Elliptical Arcs (4/9)

```
<path d="M80 80 A 45 45, 0,
0, 1, 125 125 L 125 80 Z"
fill="purple"/>
```

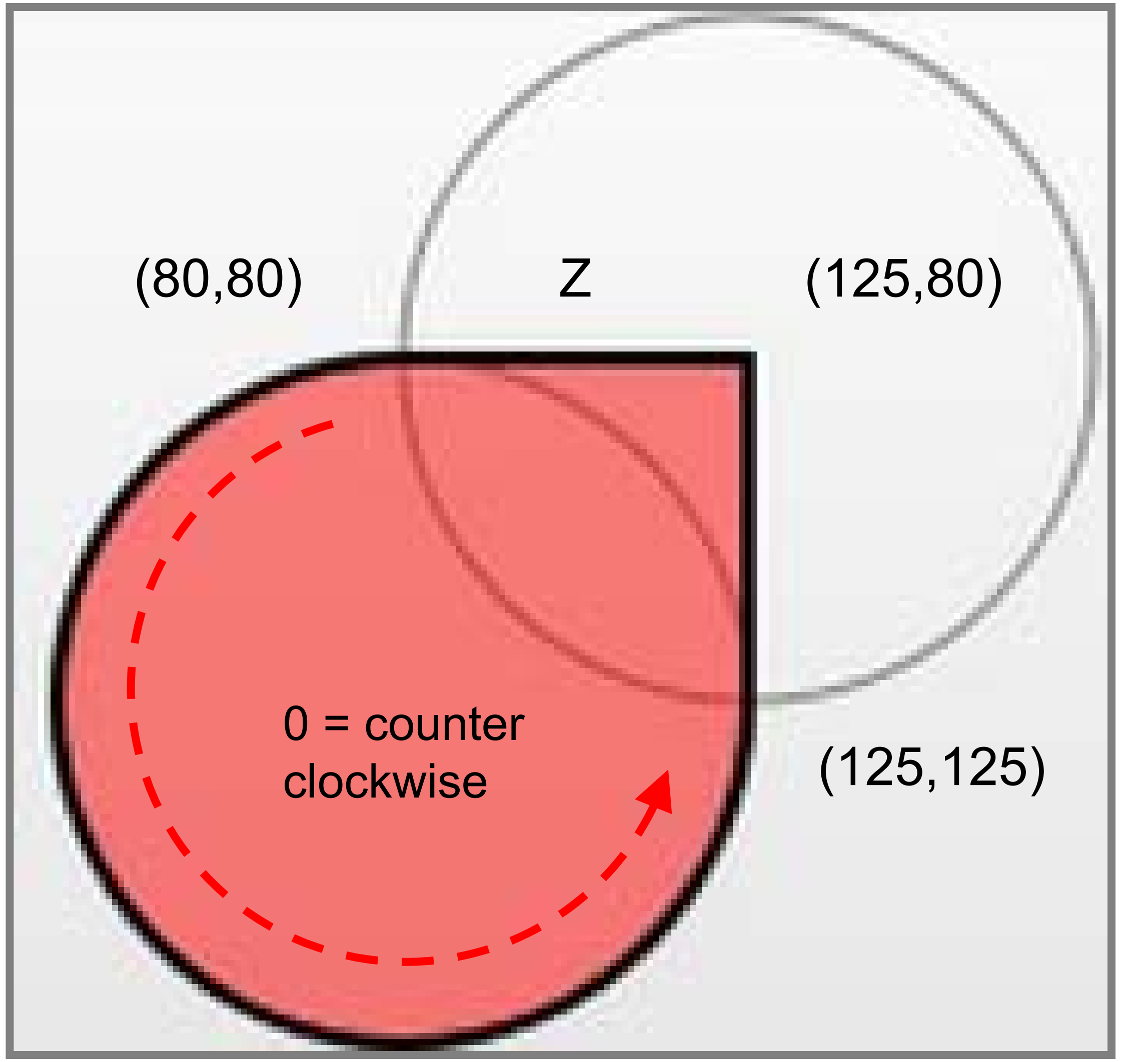
- l-flag = 0/1
  - 0 means “use one of the two smaller arcs”
  - 1 means “use one of the two larger arcs”
- sweep-flag = 0/1
  - 0 means “counter-clockwise”
  - 1 means “clockwise”



# Elliptical Arcs (5/9)

```
<path d="M80 80 A 45 45, 0,
1, 0, 125 125 L 125 80 Z"
fill="red"/>
```

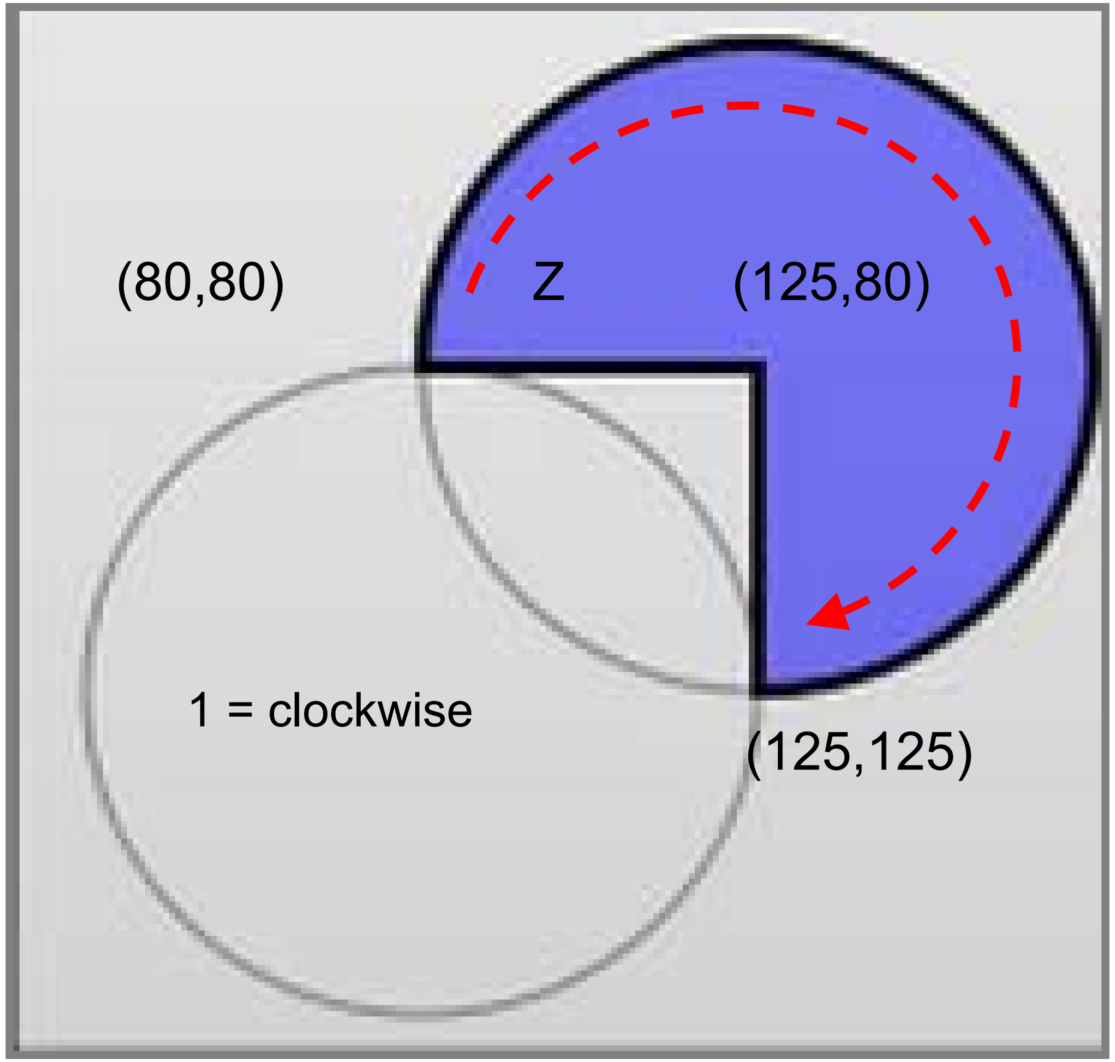
- l-flag = 0/1
  - 0 means “use one of the two smaller arcs”
  - 1 means “use one of the two larger arcs”
- sweep-flag = 0/1
  - 0 means “counter-clockwise”
  - 1 means “clockwise”



# Elliptical Arcs (6/9)

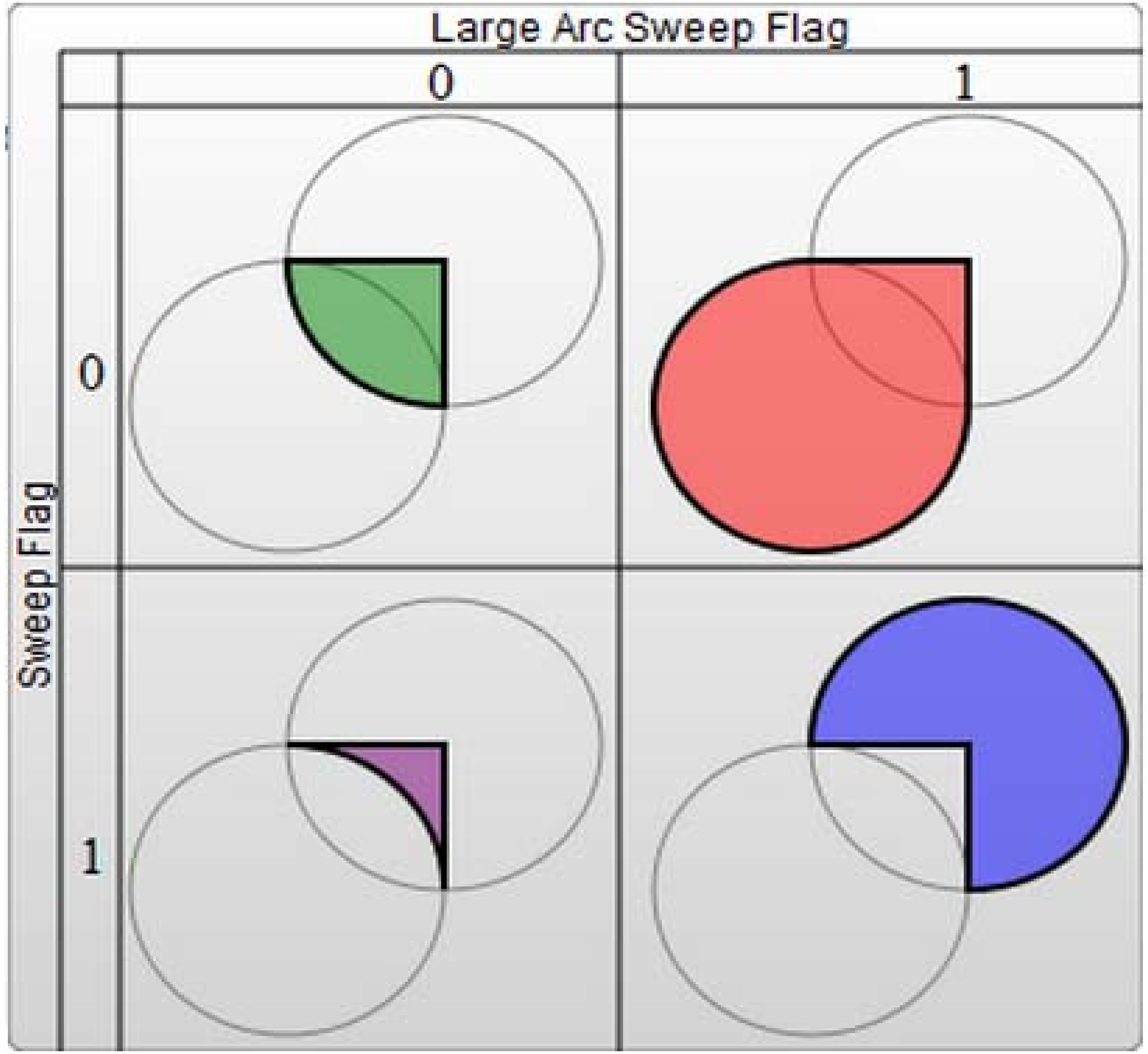
```
<path d="M80 80 A 45 45, 0,
1, 1, 125 125 L 125 80 Z"
fill="blue"/>
```

- l-flag = 0/1
  - 0 means “use one of the two smaller arcs”
  - 1 means “use one of the two larger arcs”
- sweep-flag = 0/1
  - 0 means “counter-clockwise”
  - 1 means “clockwise”



# Elliptical Arcs (7/9)

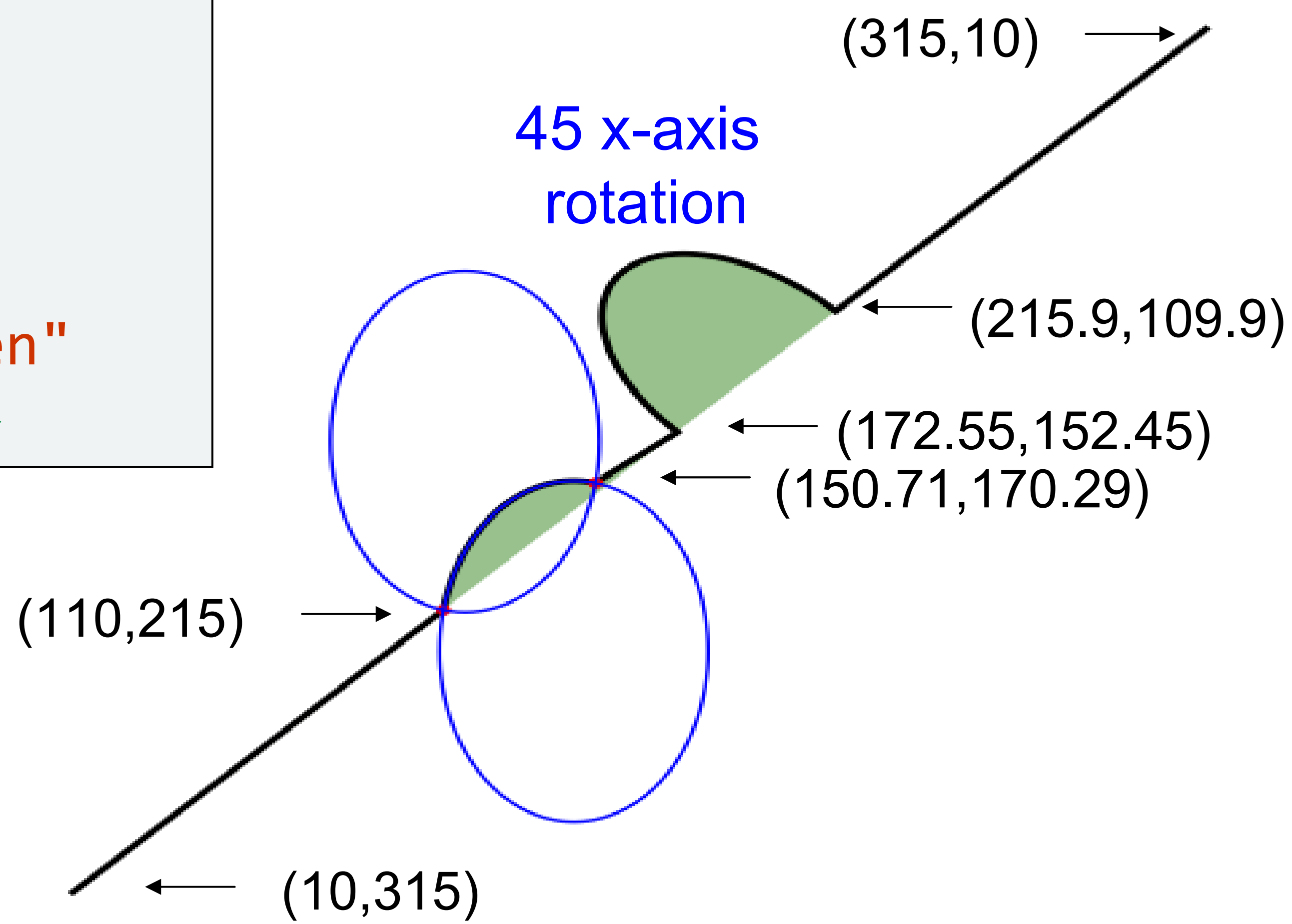
```
<path d="M80 80 A 45 45, 0,
0, 0, 125 125 L 125 80 Z"
fill="green"/>
<path d="M230 80 A 45 45, 0,
1, 0, 275 125 L 275 80 Z"
fill="red"/>
<path d="M80 230 A 45 45, 0,
0, 1, 125 275 L 125 230 Z"
fill="purple"/>
<path d="M230 230 A 45 45, 0,
1, 1, 275 275 L 275 230 Z"
fill="blue"/>
```



# Elliptical Arcs (8/9)

- Example

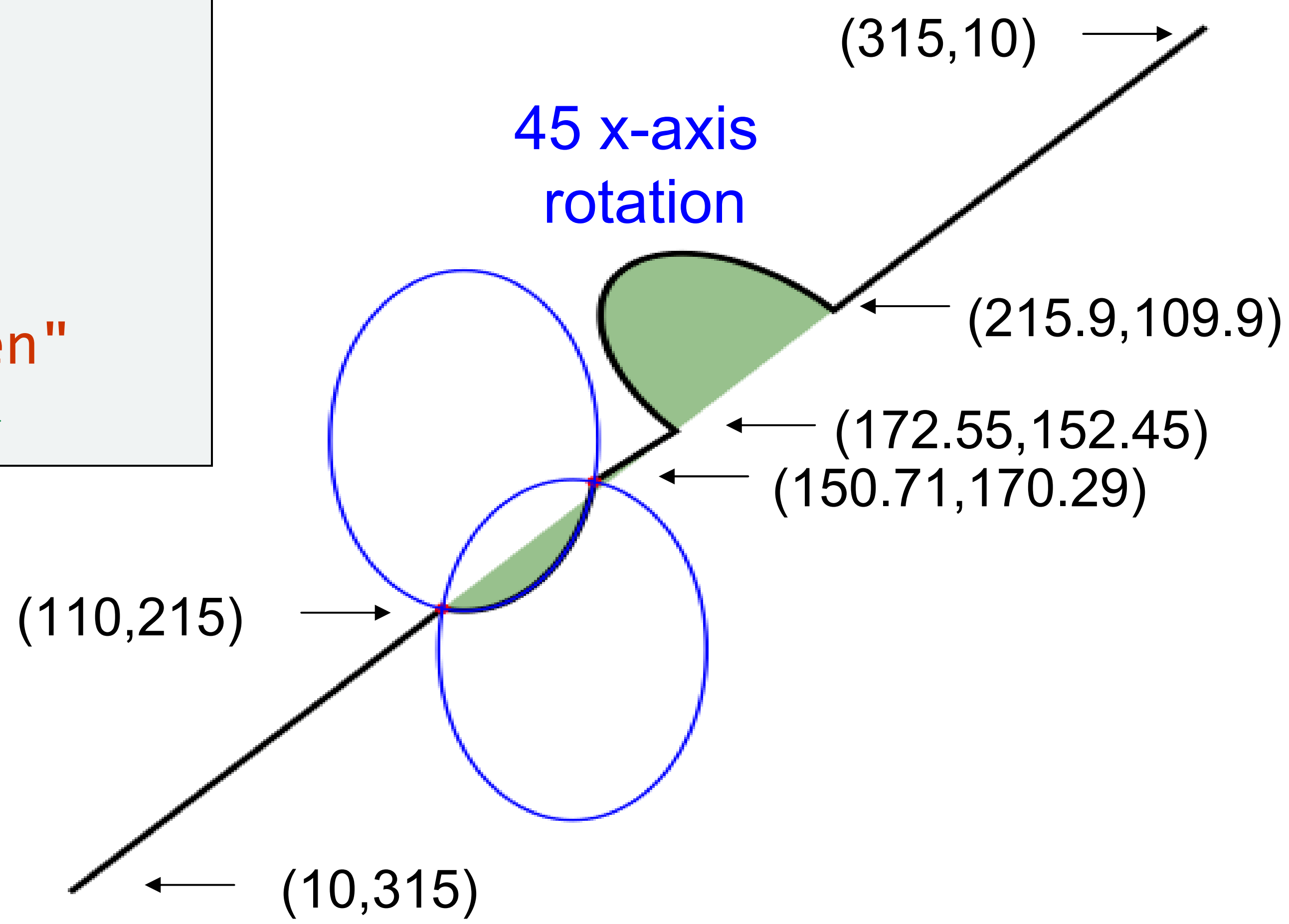
```
<path d="M10 315
  L 110 215
  A 36 60, 0, 0, 1, 150.71 170.29
  L 172.55 152.45
  A 30 50, -45, 0, 1, 215.1 109.9
  L 315 10" stroke="black" fill="green"
stroke-width="2" fill-opacity="0.5"/>
```



# Elliptical Arcs (9/9)

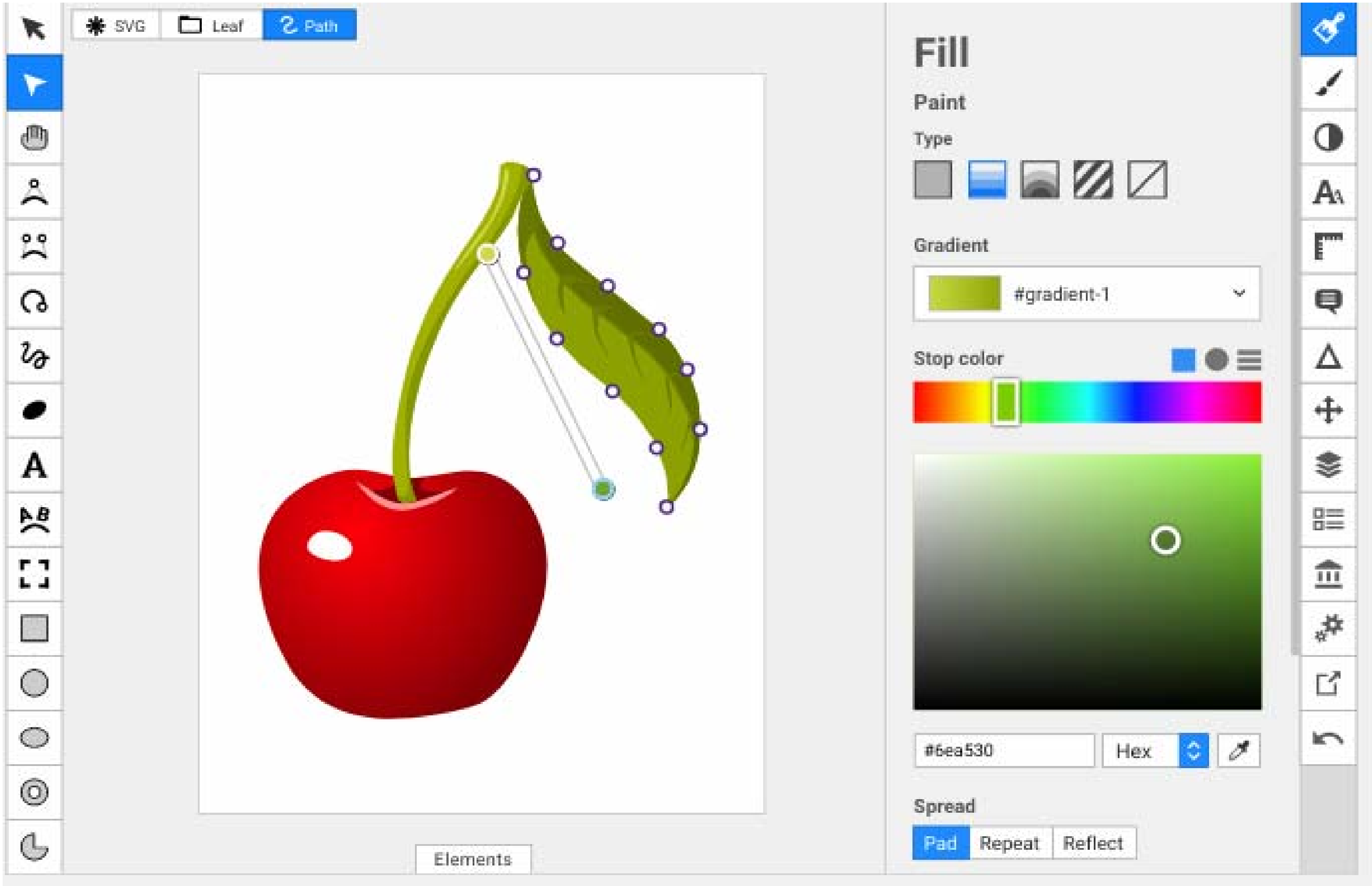
- Example

```
<path d="M10 315
  L 110 215
  A 36 60, 0, 0, 0, 150.71 170.29
  L 172.55 152.45
  A 30 50, -45, 0, 1, 215.1 109.9
  L 315 10" stroke="black" fill="green"
stroke-width="2" fill-opacity="0.5"/>
```



# SVG editors

- https://boxy-svg.com/



# Basic transformations: translations

- Translations
  - it may be necessary to move an element around
  - you can use the translate() transformation

```
<rect x="0" y="0" width="10" height="10"
      transform="translate(30,40)" />
```

  - the example will render a rectangle, translated to the point (30,40) instead of (0,0)

# Basic transformations: rotations

- Rotations
  - rotating an element is quite a common task
  - use the rotate() transformation for this

```
<rect x="20" y="20" width="20" height="20"
      transform="rotate(45)" />
```

  - this example shows a square that is rotated by 45 degrees
    - the value for rotate() is given in degrees

# Basic transformations: scalings

- Scalings
  - scale() changes the size of an element
  - it takes two numbers, evaluated as ratio by which to scale on x and y
  - 0.5 shrinks by 50%

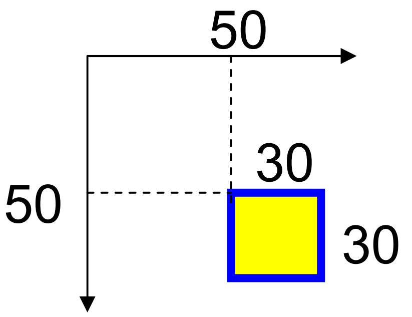
```
<rect x="20" y="20" width="20" height="20"
      transform="scale(0.8,0.8)" />
```

  - if the second number is omitted, it is assumed to be equal to the first

```
<rect x="20" y="20" width="20" height="20"
      transform="scale(0.8)" />
```

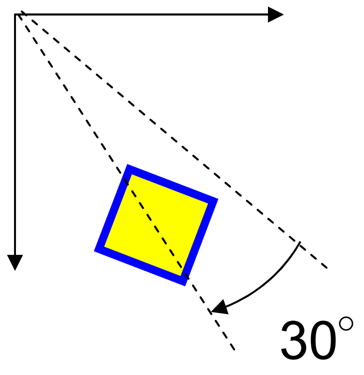
# Concatenation of transformations

- A yellow rectangle



```
<rect x="50"
      y="50"
      width="30"
      height="30"
      stroke="blue"
      stroke-width="3"
      fill="yellow"
/>
```

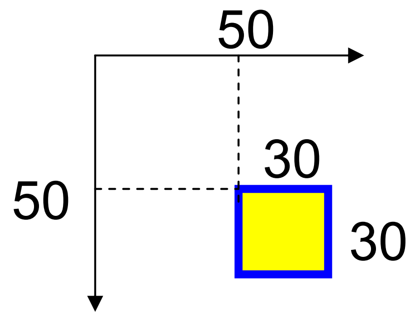
- A rotated rectangle

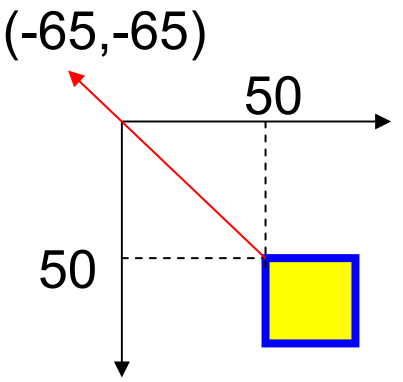


```
<rect x="50"
      y="50"
      width="30"
      height="30"
      stroke="blue"
      stroke-width="3"
      fill="yellow"
      transform="rotate(30)"
/>
```

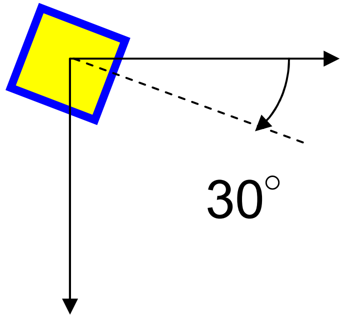
# Concatenation of transformations

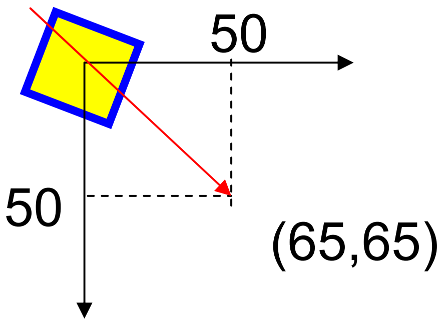
- transformations are applied in reverse order

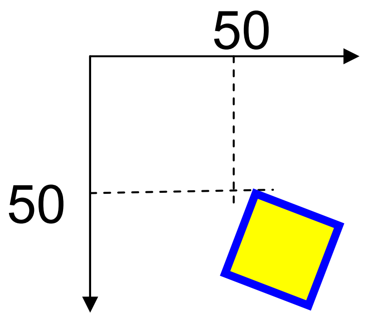




```
<rect x="50"
      y="50"
      width="30"
      height="30"
      stroke="blue"
      stroke-width="3"
      fill="yellow"
      transform="translate(65,65),
                 rotate(30),
                 translate(-65,-65)"
/>
```







# Bibliography and links

- [Murray ‘13] Scott Murray, “Interactive Data Visualization for the Web”. O’Reilly Media, 1st ed., 2013
- [Judd ‘75] Deane B. Judd, “Color in business, science and industry”. Wiley-Interscience, 3rd ed., 1975
- [W3C] http://www.w3.org/TR/SVG/intro.html
- [TM] http://www.teaching-materials.org/
- [MDN] https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial
- [MDNP] https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Paths