Coordination System and Matrix Transformation in SVG

Transformation using Matrix

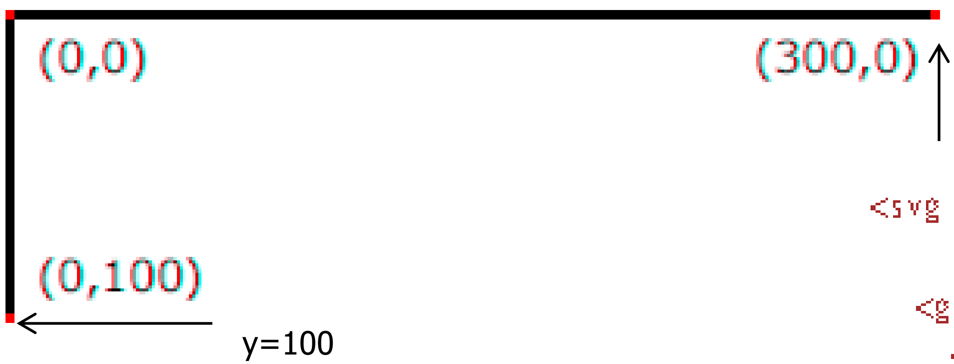
- In computer graphics, matrices are often used to represent graphics objects and operations on them
- Each operation (e.g., translation/ rotation/ scaling) can be represented by a matrix
  - A sequence of operations can be pre-computed into one single matrix and applied to a graphic element efficiently
- SVG supports the *matrix()* command
- You need to understand the general idea of matrix() as discussed in this set of slides – but you won't be expected to build something using it, as it is too 'pure' computer graphics for comp 4021

Initial User Coordinate System

- Initial viewport = Initial user Coordinate System
- Initial viewport = Outermost <SVG> element

```
<!-- SVG graphic -->
<svg xmlns='http://www.w3.org/2000/svg'
   width="100px" height="200px" version="1.1">
   <path d="M100,100 Q200,400,300,100"/>
   <!-- rest of SVG graphic would go here -->
</svg>
```

Initial User Coordinate System

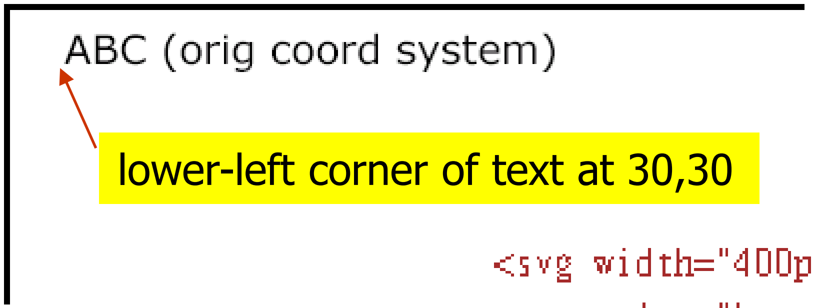


Stroke-width is 3 pts, which is thick, we position the “vertical center” of the line at y=1.5 pts so that its upper edge just touches the y=0 axis.

```
<svg width="300px" height="100px" version="1.1"
     xmlns="http://www.w3.org/2000/svg">
   <g fill="none" stroke="black" stroke-width="3" >
     <line x1="0" y1="1.5" x2="300" y2="1.5" />
     <line x1="1.5" y1="0" x2="1.5" y2="100" />
   </g>
   <g fill="red" stroke="none" >
     <rect x="0" y="0" width="3" height="3" />
     <rect x="297" y="0" width="3" height="3" />
     <rect x="0" y="97" width="3" height="3" />
   </g>
   <g font-size="14" font-family="Verdana" >
     <text x="10" y="20">(0,0)</text>
     <text x="240" y="20">(300,0)</text>
     <text x="10" y="90">(0,100)</text>
   </g>
</svg>
```

The three small red rectangles (see carefully)

Display in Current Coordinate System

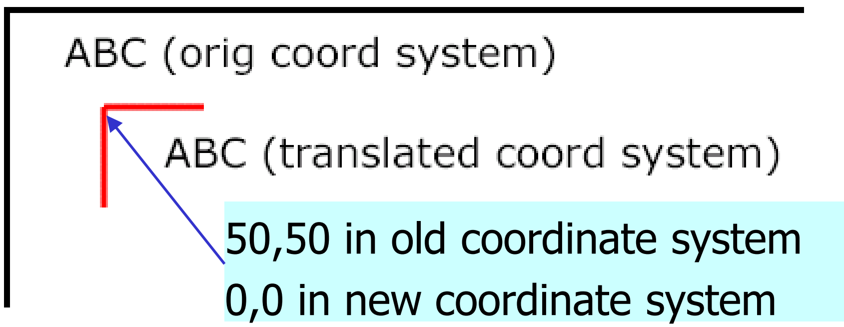


x and y axis

Text

```
<svg width="400px" height="150px"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <g fill="none" stroke="black" stroke-width="3" >
    <!-- Draw the axes of the original coordinate system -->
    <line x1="0" y1="1.5" x2="400" y2="1.5" />
    <line x1="1.5" y1="0" x2="1.5" y2="150" />
  </g>
  <g>
    <text x="30" y="30" font-size="20" font-family="Verdana" >
      ABC (orig coord system)
    </text>
  </g>
</svg>
```

Translate the Coordinate System



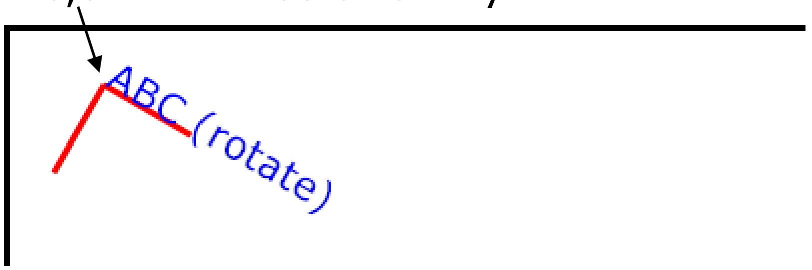
Translate the coordinate system to 50,50

```
<g transform="translate(50,50)">
  <g fill="none" stroke="red" stroke-width="3" >
    <!-- Draw lines of length 50 user units along
         the axes of the new coordinate system -->
    <line x1="0" y1="0" x2="50" y2="0" stroke="red" />
    <line x1="0" y1="0" x2="0" y2="50" />
  </g>
  <text x="30" y="30" font-size="20" font-family="Verdana" >
    ABC (translated coord system)
  </text>
</g>
```

Identical to previous slide (except the text string) but this <g> is drawn in the new coordinate system

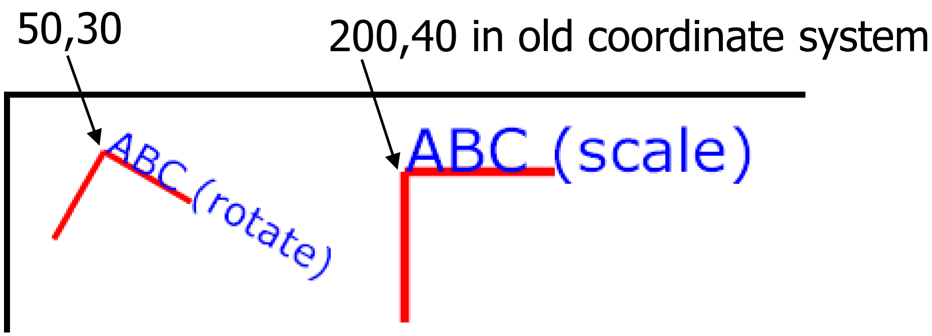
Rotate the Coordinate System

50,30 in old coordinate system
0,0 in new coordinate system




```
<g transform="translate(50,30)">
  <g transform="rotate(30)">
    <g fill="none" stroke="red" stroke-width="3" >
      <line x1="0" y1="0" x2="50" y2="0" />
      <line x1="0" y1="0" x2="0" y2="50" />
    </g>
    <text x="0" y="0" font-size="20" font-family="Verdana" fill="blue" >
      ABC (rotate)
    </text>
  </g>
</g>
```

Translate then Scale the Coordinate System

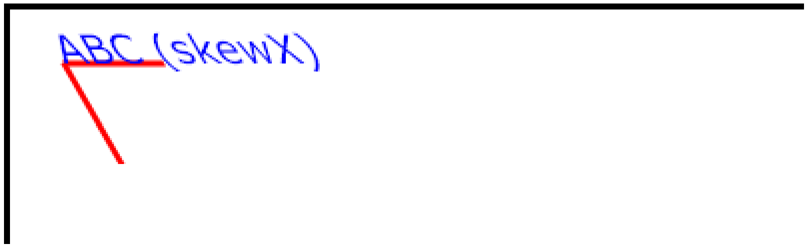



```
<g transform="translate(200,40)">
  <g transform="scale(1.5)">
    <g fill="none" stroke="red" stroke-width="3" >
      <line x1="0" y1="0" x2="50" y2="0" />
      <line x1="0" y1="0" x2="0" y2="50" />
    </g>
    <text x="0" y="0" font-size="20" font-family="Verdana" fill="blue" >
      ABC (scale)
    </text>
  </g>
</g>
```

Stroke width is the same as before but is 50% thicker now

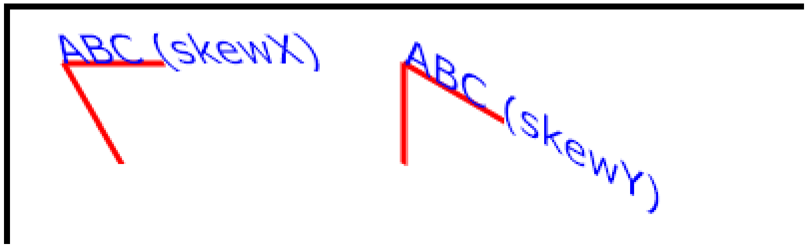
Font size is the same as before but is displayed 50% larger

Skew the Coordinate System – skewX




```
<g transform="translate(30,30)">
  <g transform="skewX(30)">
    <g fill="none" stroke="red" stroke-width="3" >
      <line x1="0" y1="0" x2="50" y2="0" />
      <line x1="0" y1="0" x2="0" y2="50" />
    </g>
    <text x="0" y="0" font-size="20" font-family="Verdana" fill="blue" >
      ABC (skewX)
    </text>
  </g>
</g>
```

Skew the Coordinate System – skewY




```
<g transform="translate(200,30)">
 <g transform="skewY(30)">
   <g fill="none" stroke="red" stroke-width="3" >
     <line x1="0" y1="0" x2="50" y2="0" />
     <line x1="0" y1="0" x2="0" y2="50" />
   </g>
   <text x="0" y="0" font-size="20" font-family="Verdana" fill="blue" >
     ABC (skewY)
   </text>
 </g>
</g>
```

Take Home Message

- The effect of manipulating an object in a coordinate system can be achieved by manipulating the coordinate system
- After you “transformed” the coordinate system, everything you put on the coordinate system is changed
  - Does the Super Mario Brother moves or the background moves?
- <g></g> transforms the coordinate system for all objects defined inside it
- <g transform=“translate(10,10)”>

  <g id=“1” transform=“rotate(30)” …> … </g>

  <g id=“2” …> … </g>

  </g>

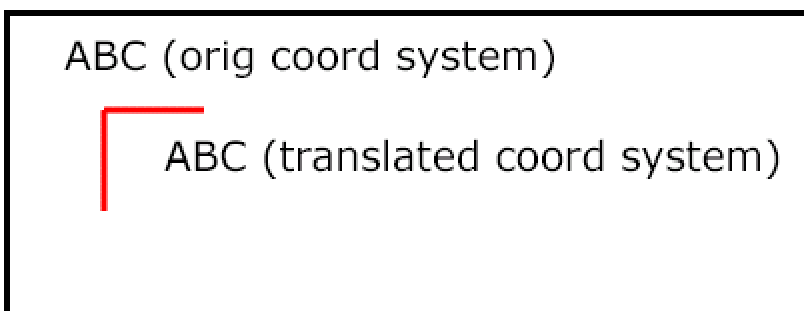
Matrix Transformation

- Matrix representation of a transformation:
- Vector form: [a b c d e f]

3x3 matrix $\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$

- Transformations map coordinates and lengths of a new coordinate system into a previous coordinate system:

$$\begin{bmatrix} x_{prevCoordSys} \\ y_{prevCoordSys} \\ 1 \end{bmatrix} = \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{newCoordSys} \\ y_{newCoordSys} \\ 1 \end{bmatrix}$$



- To draw a line (e.g., horizontal red line) in the new coordinate system, map it into a line in the original coordinate system

Matrix Transformation

- translate (tx, ty)
  vector form: [1 0 0 1 tx ty]
- E.g., (x,y) in the new coordinate system is the same as (x+tx,y+ty) in the original coordinate system, i.e., a translation of (tx,ty)

$$\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} x+tx \\ y+ty \\ 1 \end{bmatrix}$$

- scale(sx, sy)
  vector form: [sx 0 0 sy 0 0]
- 1 unit of x in the new coordinate system is sx units of x in the original coordinate system , e.g., sx=1.5 means that 1 unit of new x is equal to 1.5 units of old x
- Same for y and sy

$$\begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ & & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} sx*x \\ sy*y \\ 1 \end{bmatrix}$$

Matrix Transformation: Rotate

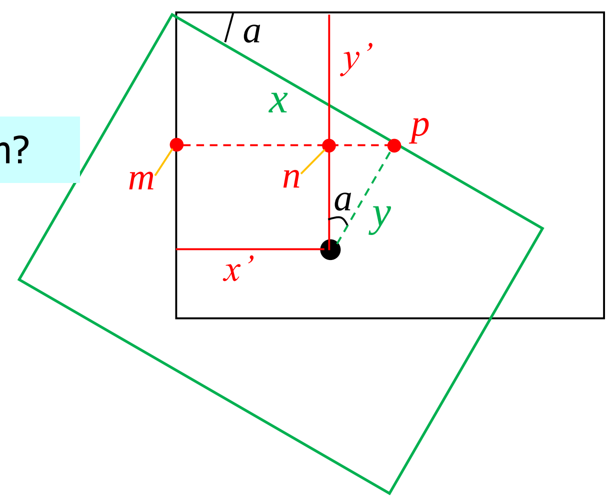
$$\begin{bmatrix} \cos(a) & -\sin(a) & 0 \\ \sin(a) & \cos(a) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- rotate(a)

  [cos(a) sin(a) –sin(a) cos(a) 0 0]

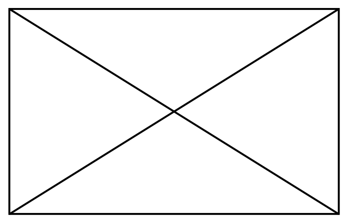
$$\begin{bmatrix} \cos(a) & -\sin(a) & 0 \\ \sin(a) & \cos(a) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} x*\cos(a) - y*\sin(a) \\ x*\sin(a) + y*\cos(a) \\ 1 \end{bmatrix}$$

- Original coordinate system
- New coordinate system with a point at x,y
- What is the point’s (x’,y’) in the original coordinates system?
  - x’ = mn = mp – np
  - mp = x*cos(a)
  - np = y*sin(a)
  - x’ = x*cos(a) – y*sin(a)
- Similarly for the point’s y’ in the original coordinate system



Matrix Transformation: Rotate at Center

- rotate(a <cx> <cy>) is equivalent to:
  translate(cx,cy) rotate(a) translate(-cx, -cy)



Matrix Transformation

- skewX(a)
  [1 0 tan(a) 1 0 0]
  $$\begin{bmatrix} 1 & \tan(a) & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- skewY(a)
  [1 tan(a) 0 1 0 0]
  $$\begin{bmatrix} 1 & 0 & 0 \\ \tan(a) & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Nested Transformation

- Sequence of transformation can be pre-computed
- Current Transformation Matrix (CTM): All transformations that have been defined on the given element and all of its ancestors up to and including the current viewport

$$\begin{bmatrix} x_{prev} \\ y_{prev} \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & c_1 & e_1 \\ b_1 & d_1 & f_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_2 & c_2 & e_2 \\ b_2 & d_2 & f_2 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{curr} \\ y_{curr} \\ 1 \end{bmatrix}$$

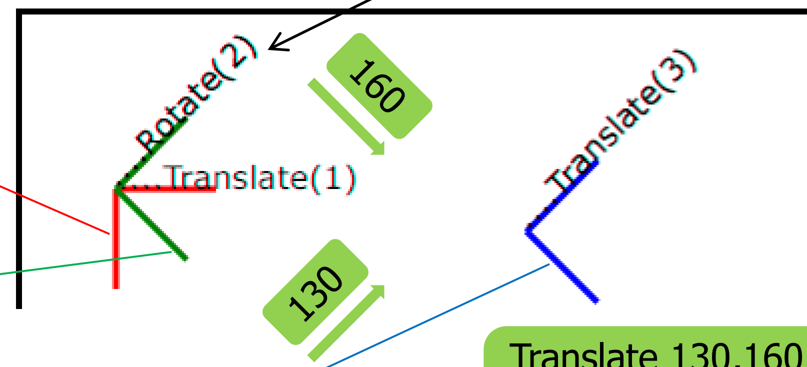
$$CTM = \begin{bmatrix} a_1 & c_1 & e_1 \\ b_1 & d_1 & f_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_2 & c_2 & e_2 \\ b_2 & d_2 & f_2 \\ 0 & 0 & 1 \end{bmatrix} \cdot \ldots \cdot \begin{bmatrix} a_n & c_n & e_n \\ b_n & d_n & f_n \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x_{viewport} \\ y_{viewport} \\ 1 \end{bmatrix} = CTM \cdot \begin{bmatrix} x_{userspace} \\ y_{userspace} \\ 1 \end{bmatrix}$$

Nested Transformation

```
<!-- First, a translate -->
<g transform="translate(50,90)">
  <g fill="none" stroke="red" stroke-width="3" >
    <line x1="0" y1="0" x2="50" y2="0" />
    <line x1="0" y1="0" x2="0" y2="50" />
  </g>
  <text x="0" y="0" font-size="16" font-family="Verdana" >
    ....Translate(1)
  </text>
  <!-- Second, a rotate -->
  <g transform="rotate(-45)">
    <g fill="none" stroke="green" stroke-width="3" >
      <line x1="0" y1="0" x2="50" y2="0" />
      <line x1="0" y1="0" x2="0" y2="50" />
    </g>
    <text x="0" y="0" font-size="16" font-family="Verdana" >
      ....Rotate(2)
    </text>
    <!-- Third, another translate -->
    <g transform="translate(130,160)">
      <g fill="none" stroke="blue" stroke-width="3" >
        <line x1="0" y1="0" x2="50" y2="0" />
        <line x1="0" y1="0" x2="0" y2="50" />
      </g>
      <text x="0" y="0" font-size="16" font-family="Verdana" >
        ....Translate(3)
      </text>
    </g>
  </g>
</g>
```

Rotate around the origin, which is now at 50,90



Translate 130,160 in the green coordinates

$$CTM = \text{translate}(50,90),\ \text{rotate}(-45),\ \text{translate}(130,160)$$

$$= \begin{bmatrix} 1 & 0 & 50 \\ 0 & 1 & 90 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} .707 & .707 & 0 \\ -.707 & .707 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 130 \\ 0 & 1 & 160 \\ 0 & 0 & 1 \end{bmatrix}$$

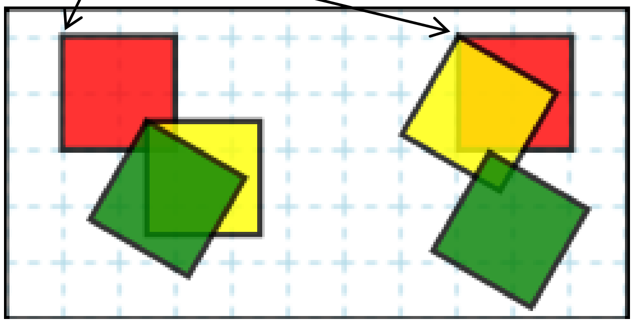
$$= \begin{bmatrix} .707 & .707 & 255.03 \\ -.707 & .707 & 111.21 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x_{initial} \\ y_{initial} \\ 1 \end{bmatrix} = CTM \cdot \begin{bmatrix} x_{userspace} \\ y_{userspace} \\ 1 \end{bmatrix}$$

Order of Transformation Matters

This means translate then rotate the coord system, not the box; however, the yellow box gives us an impression that it refers to first translate the box then rotates it, which is incorrect; see examples and discussion under svg examples in course homepage

Current point before transform



- The green square on the left is produced by translate(15,15) rotate(30) of the red square
- The green square on the right is produced by rotate(30) translate(15,15) of the red square

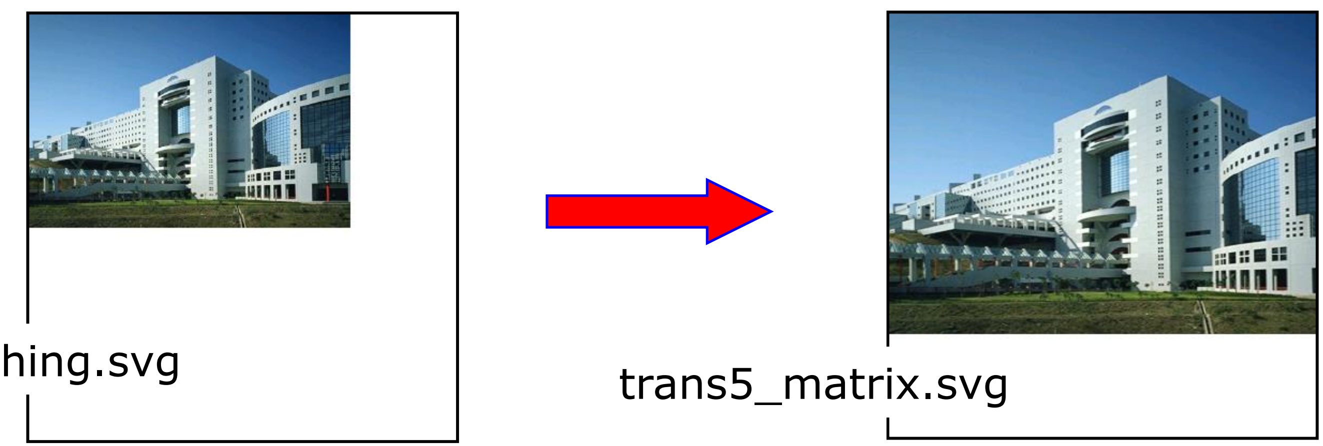
```
<!-- Translate then rotate -->

<use xlink:href="#example" fill="red" />
<g transform="translate(15, 15)" fill="yellow">
    <use xlink:href="#example" />
    <g transform="rotate(30)" fill="green">
        <use xlink:href="#example" />
    </g>
</g>

<!-- Rotate then translate -->

<g transform="translate(65)">
<use xlink:href="#example" fill="red" />
<g transform="rotate(30)" fill="yellow">
    <use xlink:href="#example" />
    <g transform="translate(15, 15)" fill="green">
        <use xlink:href="#example" />
    </g>
</g>
</g>
```

Same comment as above

Matrix Example (Scaling)

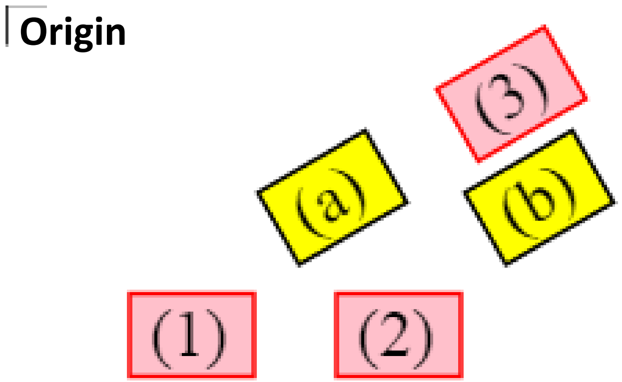
- The following matrix multiplies all x values by 1.5 and all y values also by 1.5

```
<image xlink:href="ust.jpg"
 transform="matrix(1.5 0 0 1.5 0 0)" x="0" y="0" width="300" height="200"/>
```



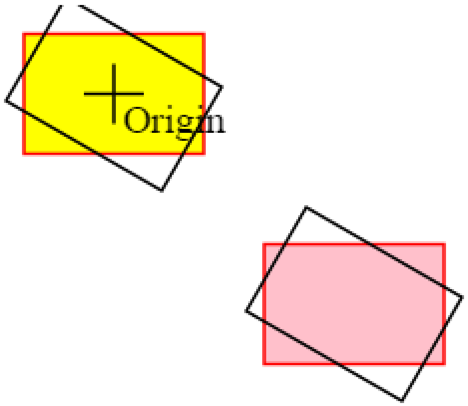
Multiple Operations Example #1

- A transform can include multiple operations performed from right to left (why not left to right?)
  - transform="**rotate(30) translate(50, 0)**"
    1) **translate** a shape by (50,0); (1) $\Rightarrow$ (2)
    2) **rotate** it -30 degrees; (2) $\Rightarrow$ (3)
  - transform="**translate(50, 0) rotate(30)** "
    1) **rotate** a shape -30 degrees; (1) $\Rightarrow$ (a)
    2) **translate** it by (50,0); (a) $\Rightarrow$ (b)



Multiple Operations Example #2

- transform="**translate(150,100),rotate(30),translate(-150,-100)**"
  1) translate a shape by (x1, y1) = (-150, -100) => yellow box
  2) rotate it 30 degrees => black box around origin
  3) translate it by (x2, y2) = (150, 100) => black box around pink box
- The above transform rotates a shape around (150,100)



Multiple Operations Example #3

- transform="**translate(200,100),rotate(30),translate(-150,-100)**"
  1) translate a shape by (x1, y1) = (-150, -100) => yellow box
  2) rotate it 30 degrees => black box around origin
  3) translate it by (x2, y2) = (200, 100) => black box around pink box

- See the following slides

General Matrix for the Three Operations

```
[ 1  0  x2]    [cos(a)  -sin(a)  0]    [ 1  0  -x1]
[ 0  1  y2] *  [sin(a)   cos(a)  0] *  [ 0  1  -y1]
[ 0  0  1 ]    [  0        0     1]    [ 0  0   1 ]
```

translate (x2, y2) =>
translate (200, 100)

rotate (a) => rotate (30)

translate (-x1, -y1) =>
translate (-150, -100)

After multiplying all three matrices, the CTM is:

```
[cos(a)   -sin(a)    -x1cos(a) + y1sin(a) + x2]
[sin(a)    cos(a)    -x1sin(a) - y1cos(a) + y2]
[  0         0                 1               ]
```

The SVG Matrix for the Example

- The equivalent SVG matrix is:

a = 30
x1 = 150, y1 = 100
x2 = -200, y2 = 100

transform =

"matrix(cos(a), sin(a),

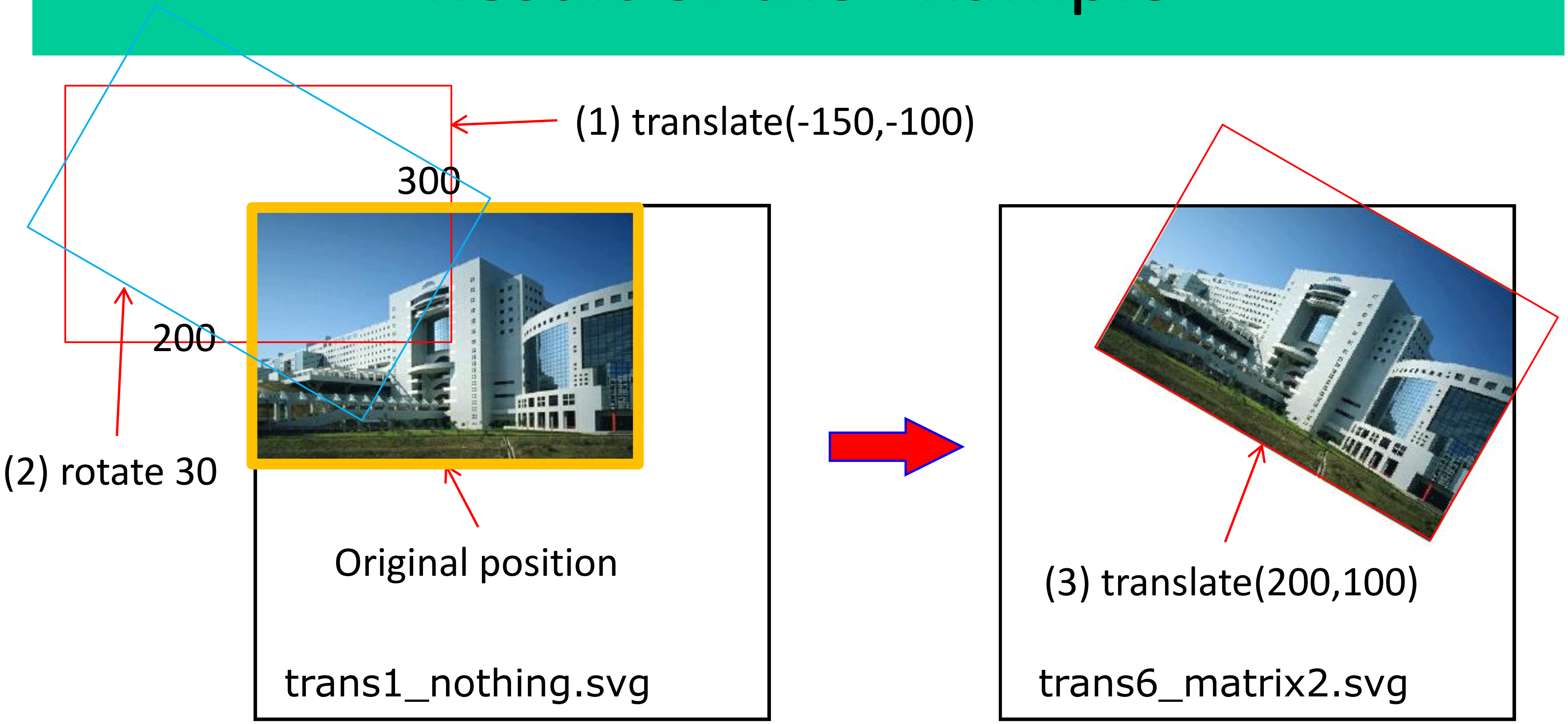
-sin(a), cos(a),

-x1cos(a) + y1sin(a) +x2, -x1sin(a) – y1cos(a) + y2)"

- In this particular case:

transform="matrix(0.866 0.5 -0.5 0.866 120.1 -61.6)"

Result of the Example



1. translate(-150,-100) from origin
2. then rotate it 30 degrees
3. then translate (200, 100)

Result of the Example

- Without using a composite matrix, the previous example can be done with (operations from right to left):

```
< image xlink:href="ust.jpg" transform="
        translate(200,100)  rotate(30)  translate(-150,-100)"
        x="0" y="0" width="300" height="200" />
```

Take Home Message

- SVG has implemented sophisticated computer graphics techniques for drawing, transforming and animating objects
- Distinguish the differences of an object manipulation in different coordinate systems
  - Transformation and coordination systems is not restricted to SVG and is applicable to other graphics packages (e.g., java 2d and java awt)
- You can use transform commands or matrix operations to manipulate objects
- Despite its apparent simplicity, SVG can produce very complex graphics