Lecture 6

More Transformations

Strategy:

1. Learn to **use** transformations

   Workbook 3 (this week)

2. Then see how they work

   "under the hood"

   Workbook 4 (next week)

Most books do it in the other order

Chicken-and-egg:

- why do they work in this weird way?
- what are they useful for?

Review of Last Time

- Transformations
- Translate, Scale, Rotate
- Composition
- Rotate/Scale about Center

Today

- Review of some details
- Hierachy
- Retained Mode APIs (SVG)

probably not:

- math review

After Today

- Transformation Math
- Using Transformations

- Shape / Curves
- 3D

Handedness in 2D

- We measure angles (in 2D) from the X axis to the Y axis

- with Canvas, this is clockwise
- rotation doesn't change this
- reflection does
- (other systems start with CCW)

Why "Handedness"

- Right-hand, Canvas coords X->Y - thumb goes into the screen
- Rotations preserve "right thumb goes into screen"
- Reflection makes left thumb go into screen

- this makes more sense in 3D (just wait)
- this is explained via the 3D cross product (just wait)
- in 3D there is the "right hand rule" (just wait)

Transformation basics

- draw in the **current coordinate system**
- transformations change the coordinate system
- rotate and scale around the **origin**
- compose transformations

Composed Transforms

1. Any sequence of translations can be a single translation
2. Any sequence of scalings can be a single scaling
3. Any sequence of rotations can be a single rotation

Scale and Translate

Scale then Translate => (different) Translate then Scale

$s\,(x+t) = t' + sx$

$t' = s\,t$

Rotate and Translate

Rotate then Translate => Translate (different) then Rotate

Composed Transforms

1. Any sequence of translations can be a single translation
2. Any sequence of scalings can be a single scaling
3. Any sequence of rotations can be a single rotation

4. Any sequence of scale/rotations/translations can be a single scale/rot/trans
5. (S R T)* => S R T (uniform scale)
6. Non-Uniform Scales do weird things

Non-Uniform Scales Can Be Weird

- Mirror reflection (useful for Y up)

- Stretch in a different direction

Combining things...

- same transformations, easy
- uniform scale and rotation - reorder
- uniform scale and translation - change and reorder
- rotation and translation - change and reorder

given an arbitrary sequence, you can always convert...

or use matrix math (next week)

rotate around a point

```
context.translate(cx,cy);
context.rotate(angle);
context.translate(-cx,-cy);
drawThing();
```

- make the object
- move the center point to origin
- rotate around the origin
- move the center point back
- **this is reading backwards**

yes, you can find a single TR or RT

easier to think about it this way

One more transformation: Shear (Skew)

One for each direction (ShearX and ShearY)

Hierarchy

Things made of pieces

Pieces made of smaller pieces

Articulated Chains

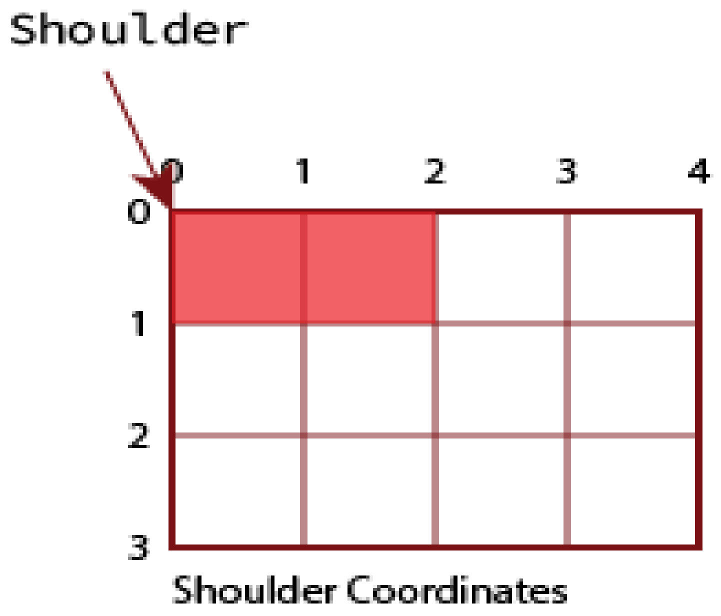
Important special case: Pieces rotate relative to each other

Arms:

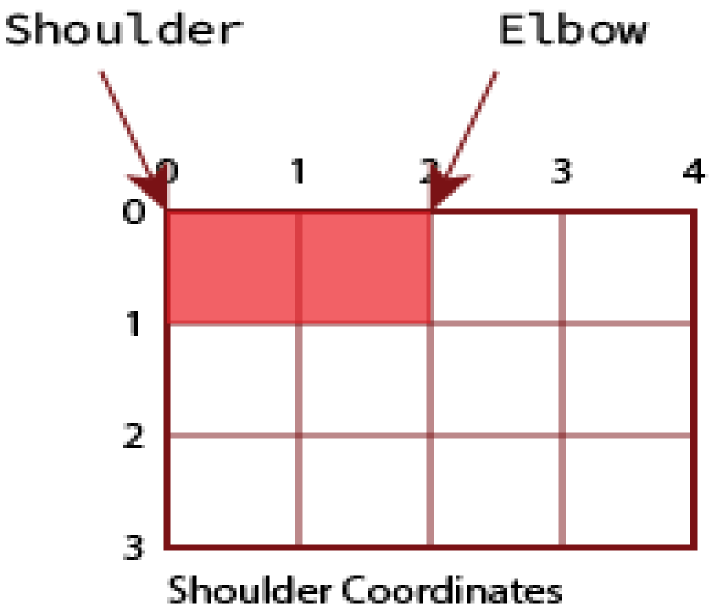
Upper Arm, Fore Arm, Hand

should stay connected

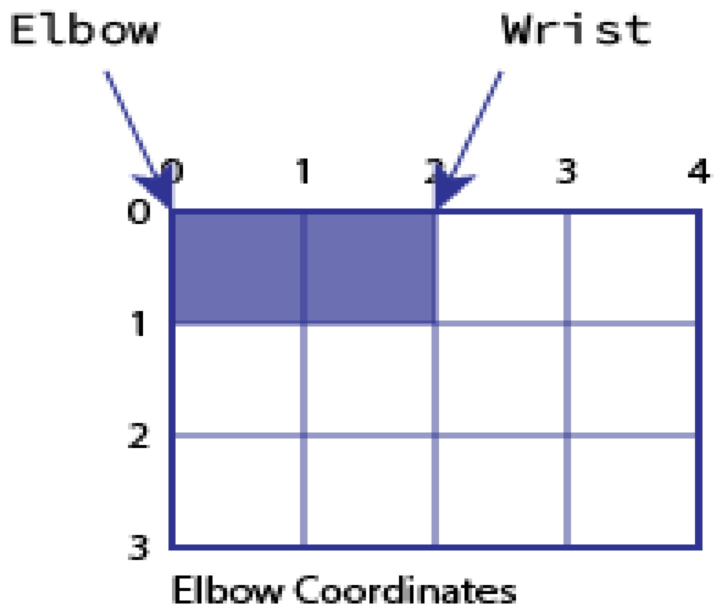
change angle between them



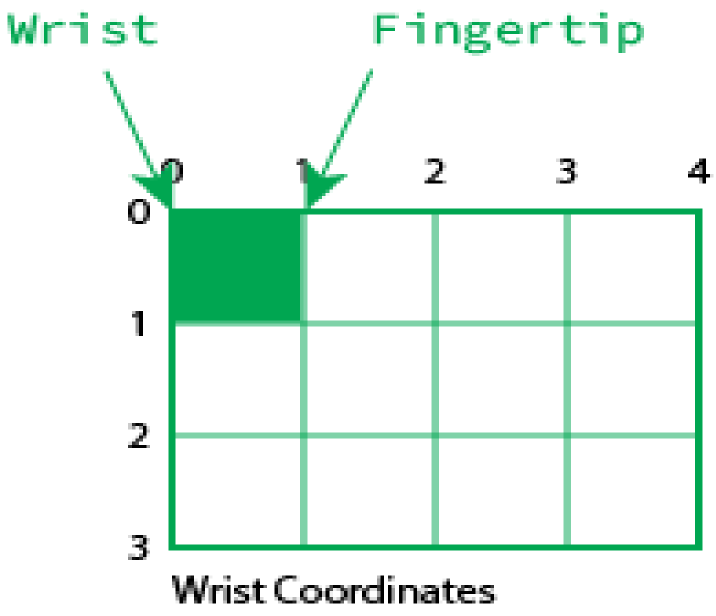

```
drawUpperArm();
```



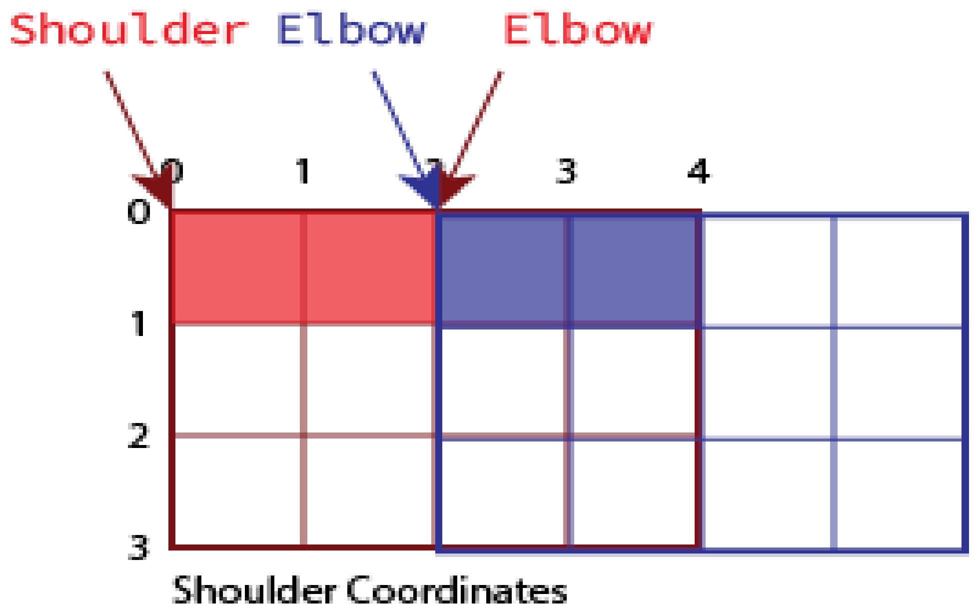

```
drawUpperArm();
// move to elbow
context.translate(2,0);
context.rotate(0);
```



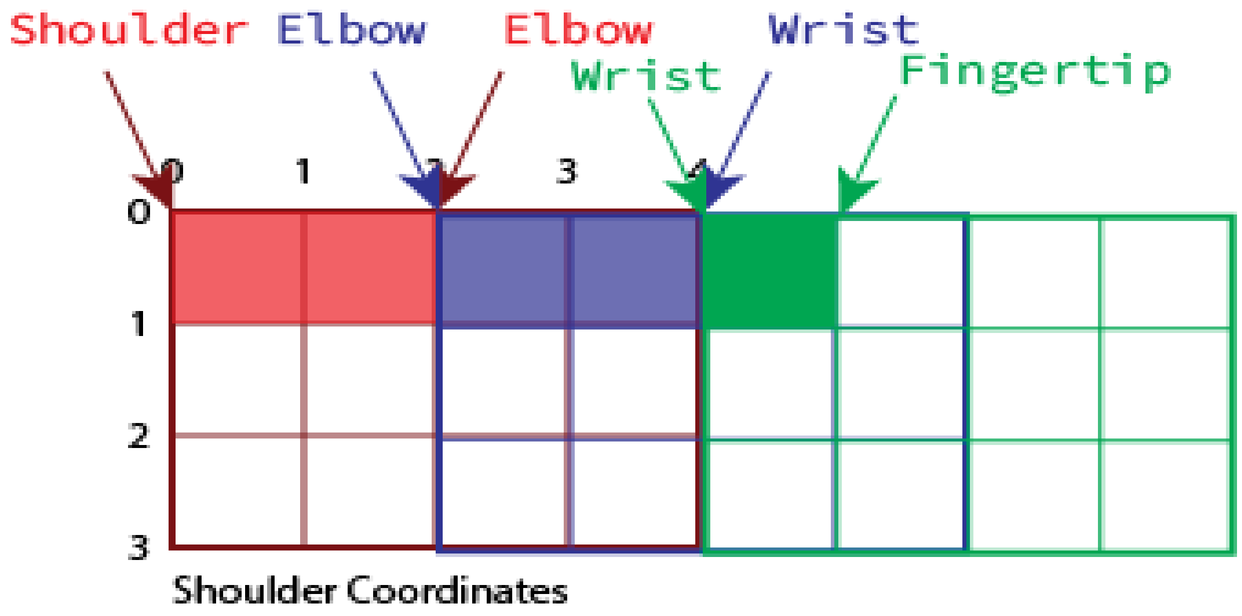

```
drawLowerArm();
// move to wrist
context.translate(2,0);
context.rotate(0);
```



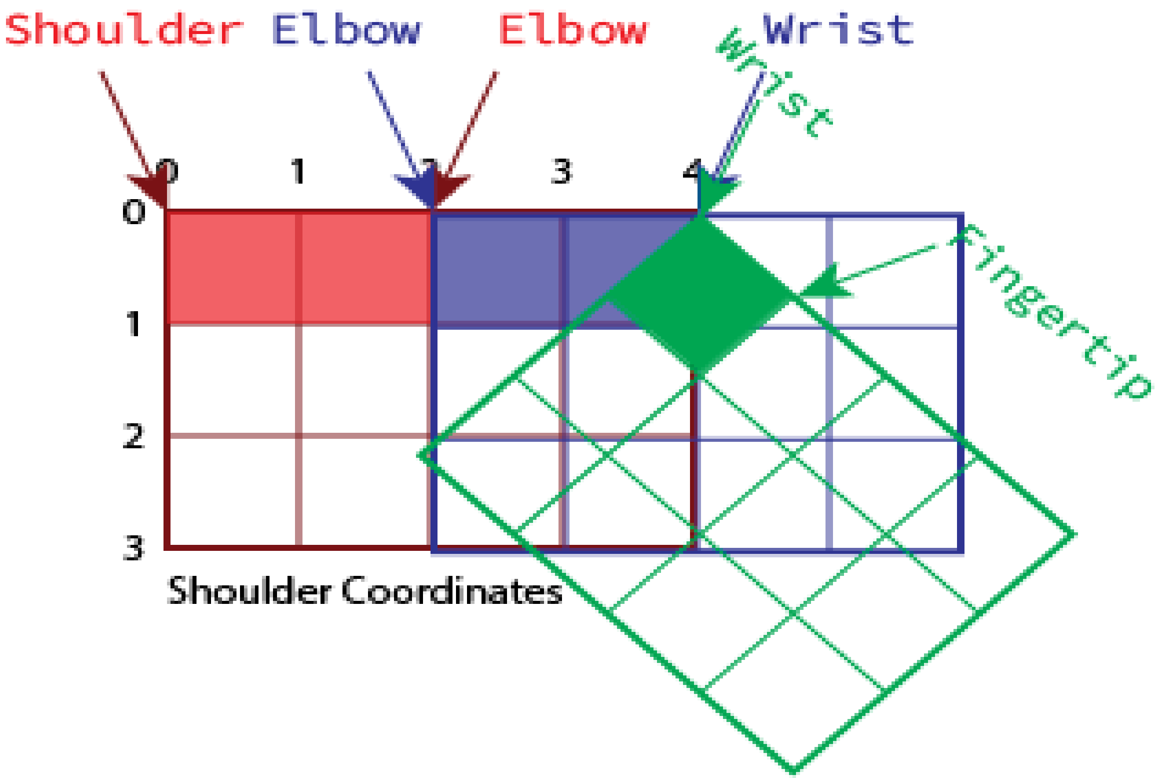

```
drawHand();
```



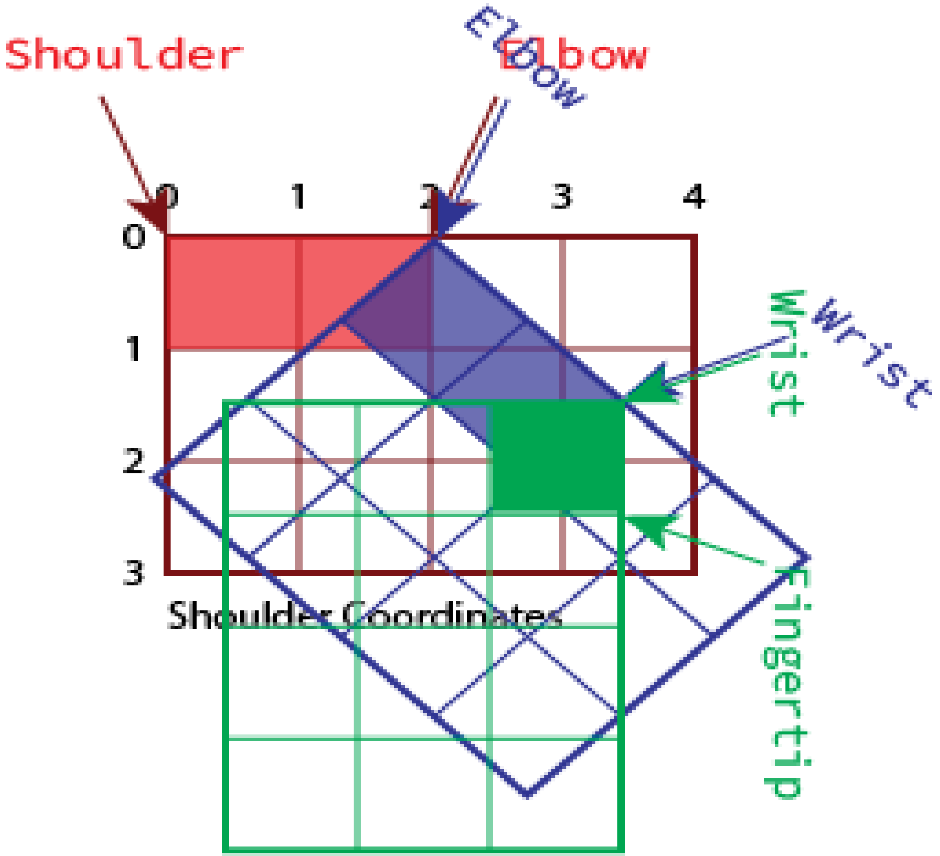

```
drawUpperArm();
// move to elbow
context.translate(2,0);
context.rotate(0);
drawLowerArm();
```



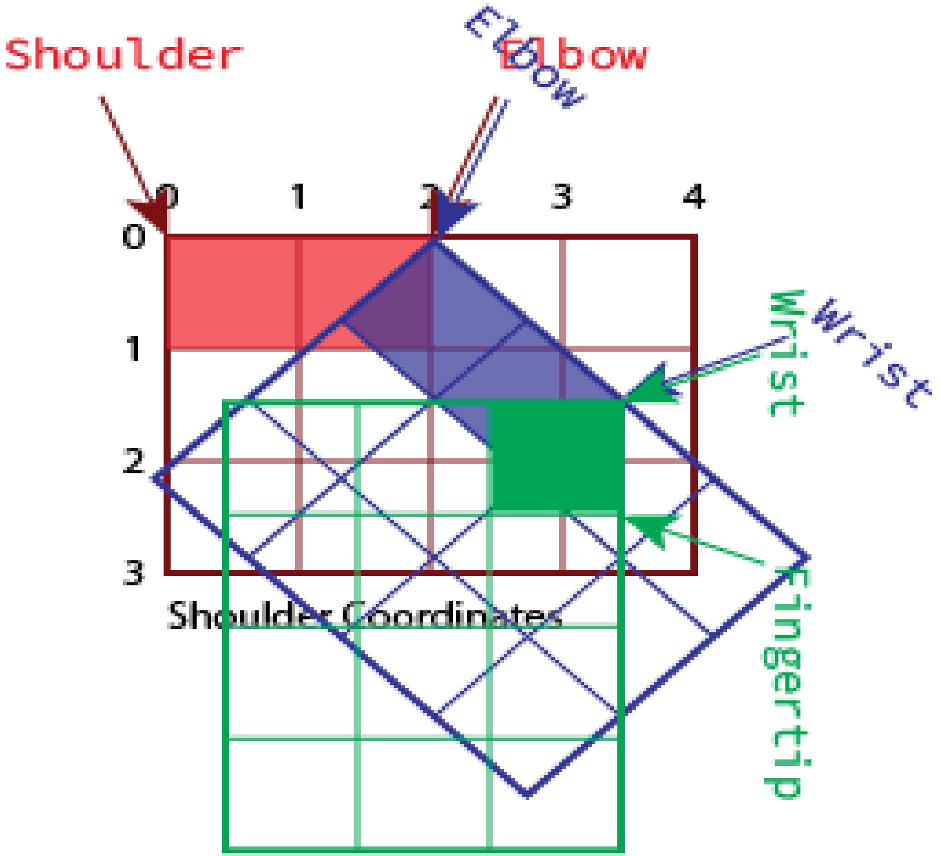

```
drawUpperArm();
// move to elbow
context.translate(2,0);
context.rotate(0);
drawLowerArm();
// move to wrist
context.translate(2,0);
context.rotate(0);
drawHand();
```



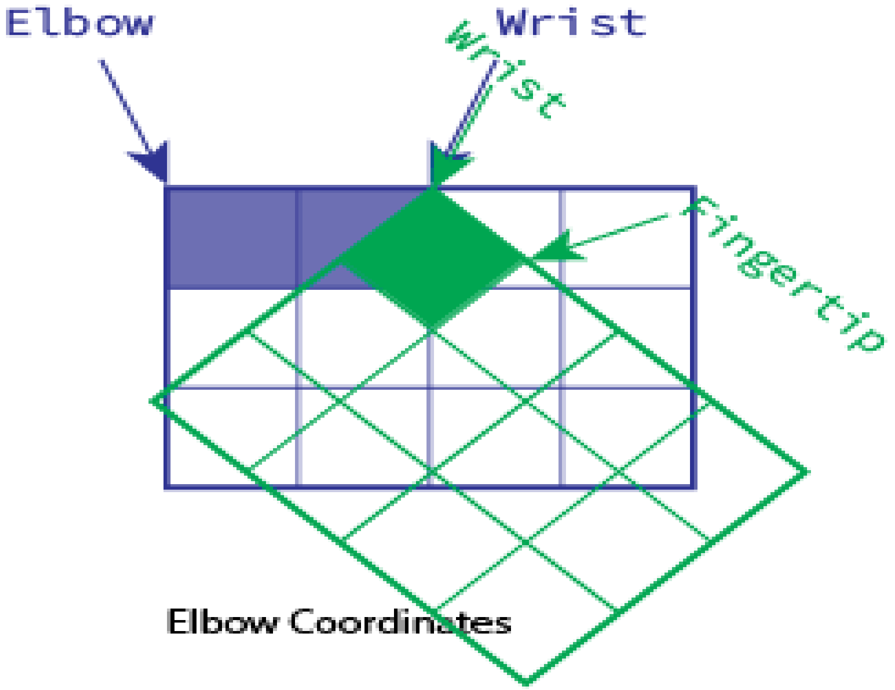

```
drawUpperArm();
// move to elbow
context.translate(2,0);
context.rotate(0);
drawLowerArm();
// move to wrist
context.translate(2,0);
context.rotate(45);
drawHand();
```




```
drawUpperArm();
// move to elbow
context.translate(2,0);
context.rotate(45);
drawLowerArm();
// move to wrist
context.translate(2,0);
context.rotate(45);
drawHand();
```




```
drawUpperArm();
// move to elbow
context.translate(2,0);
context.rotate(45);
drawLowerArm();
// move to wrist
context.translate(2,0);
context.rotate(45);
drawHand();
```




```
drawLowerArm();
// move to wrist
context.translate(2,0);
context.rotate(45);
drawHand();
```

This example is in the workbook

More generally, a **Tree** of parts

Hierarchical Modeling! (Car, Wheel, Street, City)

Trees and DAGs

instancing by copying or multiple links

Its not a List - Its a Graph/Tree

Where is the Tree (Graph) Stored

Immediate Mode: Object Representation **implicit in code**

Retained Mode: **Explicit** representation of objects

Immediate vs. Retained APIs

Immediate Mode

Drawing commands turned to pixels

We have to keep track of objects

Retained Mode

API creates objects

We have to build structures

sometimes known as **scene-graph** APIs

SVG
A scene-graph API

Every graphics object is a DOM **Element**

Rectangles, arc, circles, paths

Each element can have:

- HTML Stuff (ids, classes, event handlers, styles, ...)
- Graphics Stuff (styles, shapes, transformations, ...)

An SVG Object

SVG can be represented in text (like HTML)

SVG can be manipulated like other DOM objects

```
<circle id="mycirc" class="someclass"
        cx="60" cy="60" r="60"
        style="stroke:red; stroke-width:5; fill:pink"
        >
</circle>
```

Later, in JavaScript...

```
let circ = document.getElementById("mycirc");
circ.setAttribute('cx',"65");
```

SVG Paths and Transformations

Important information in attributes

```
<path d="some string in path language"
    transform="some transformation string"
    style="stroke:black; fill:gray; stroke-width:5" />
```

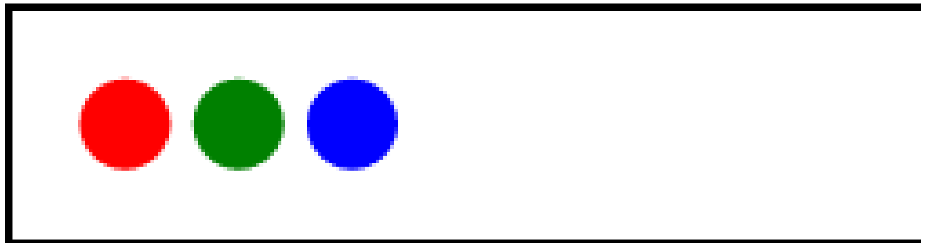
Note:

- Close Tag
- Path strings are their own language
- Transformation strings are their own language

SVG Hierarchy

- Objects can contain other objects
- usually special "group" objects `g`
- text form doesn't allow nested tags for primitives

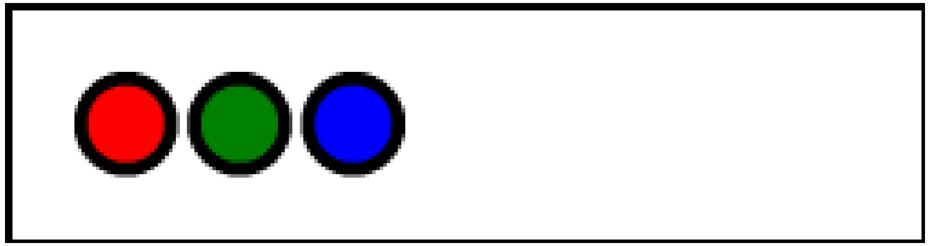
```
<g>
    <circle cx="25" cy="25" r="10" fill="red" />
    <circle cx="50" cy="25" r="10" fill="green" />
    <circle cx="75" cy="25" r="10" fill="blue" />
</g>
```



Attributes are Inherited

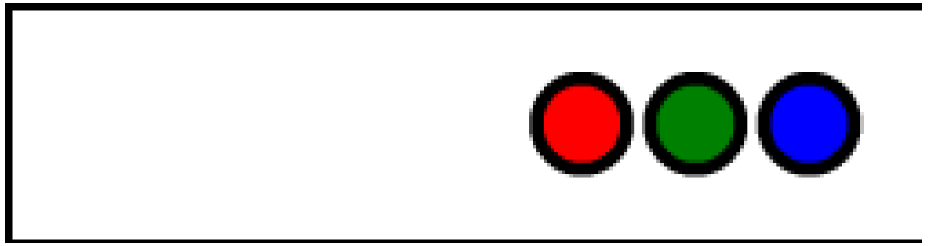
Groups can have attributes

```
<g style="stroke:black; stroke-width:3" fill="red" >
    <circle cx="25" cy="25" r="10" />
    <circle cx="50" cy="25" r="10" fill="green" />
    <circle cx="75" cy="25" r="10" fill="blue" />
</g>
```



That includes transforms

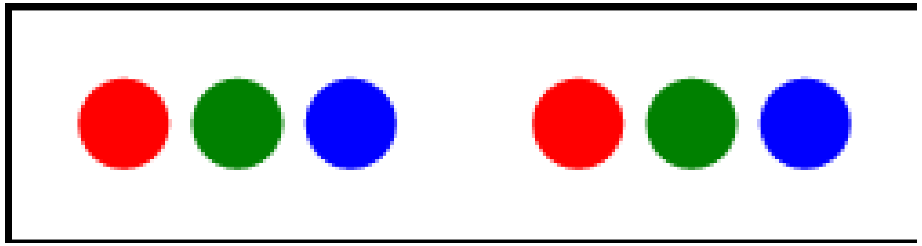
```
<g style="stroke:black; stroke-width:3" fill="red"
    transform="translate(100,0)">
    <circle cx="25" cy="25" r="10" />
    <circle cx="50" cy="25" r="10" fill="green" />
    <circle cx="75" cy="25" r="10" fill="blue" />
</g>
```



Re-Use! (Instancing)

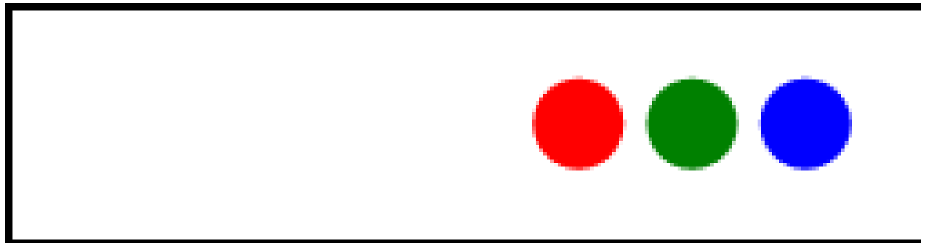
Give things names - and use them again!

```
<g id="circs">
    <circle cx="25" cy="25" r="10" fill="red" />
    <circle cx="50" cy="25" r="10" fill="green" />
    <circle cx="75" cy="25" r="10" fill="blue" />
</g>

<use href="#circs" transform="translate(100,0)" />
```

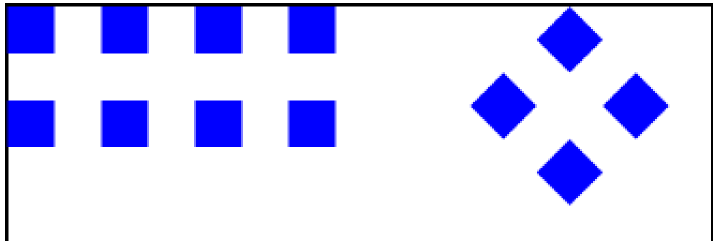


Use `def` to avoid using it the first time...

```
<def>
  <g id="circs">
    <circle cx="25" cy="25" r="10" fill="red" />
    <circle cx="50" cy="25" r="10" fill="green" />
    <circle cx="75" cy="25" r="10" fill="blue" />
  </g>
</def>

<use href="#circs" transform="translate(100,0)" />
```



```
<def>
    <rect id="asq" width="10" height="10" fill="blue"/>
</def>
<g id="four">
    <use href="#asq" x="0" y="0" />
    <use href="#asq" x="20" y="0" />
    <use href="#asq" x="0" y="20" />
    <use href="#asq" x="20" y="20" />
</g>
<use href="#four" x="40" />
<use href="#four"
     transform="translate(120,0),rotate(45)" />
```



Where do the transformations live?

Each object has a transformation

It is relative to its parent

Groups have objects (including groups) as children

Hierarchy in SVG and Canvas

Trees and Tree Traversals

SVG is cool...

- Can read in text (HTML) form
- Can create / manipulate objects in JavaScript
  - each object can have `class` and an `id`
- Can style with CSS (style sheets)
- Browser redraws when things change

and you'll learn about it in the tutorials in the workbook

Why Choose SVG or Canvas?

Why Choose SVG or Canvas?

Pedagogy (for CS559 in 2020/2021):

- wanted an immediate mode API and a scene graph API
- wanted an API with explicit transformations

Canvas is differently useful than SVG

SVG

Animation:

Objects can be changed to animate

Must make changeable objects

Interaction:

Easy to have objects respond

Harder to have global behaviors

Scalability:

Objects can do lots of things

Objects are "heavyweight"

Canvas

Animation:

Just redraw things

Need to keep track of things

Interaction:

Easy to interact with Canvas

Hard to interact with Objects

Scalability:

Objects are up to us

We don't make what we don't need

Redraw Performance

Isn't it wasteful to clear / redraw every frame?

Yes, but...

1. We don't have to
    - partial redraw (just clear/redraw part)
    - precomputed backgrounds
2. Partial redraw is hard!
    - need to figure out what changed
    - need to figure out what other objects are affected
    - often takes more work than just redrawing everything

Canvas vs. SVG: smart redraw?

Canvas: we have to do it ourselves

SVG: the web browser **might** do it for us

Thinking about Transformations

Points in local coordinates

Transformations move between coordinate systems

Summary

1. Transformations to change coordinate systems
2. Translate, Scale, Rotate
3. Composition to combine transformation
4. Examples of useful combinations
5. Hierarchical modeling
6. Immediate vs. Retained APIs

But how do transformations work?

What is the mathematics behind this?

Math you need to know...

Linear algebra in 5 slides or less

(not really)

Just the parts we'll use

Quickly today... practice later

Why? Transformations are best expressed as **matrices**

Points vs. Vectors vs. Tuples

Vectors (and points)

- addition
- multiply by scalar
- linear combination
- norms / magnitude
- dot product
- row vectors vs. column vectors

more vector stuff

- vector spaces
- projection
- and some things for 3D (and higher)
    - cross product

Matrices

- matrix as a set of row vectors
- matrix as a set of column vectors
- matrix * vector
    - right multiply
    - left multiply
- matrix * matrix

Matrix Properties

- orthogonality
- orthonormality
- determinants
- inverses
- full-rank vs. rank-deficient

Transformations as Functions

Points go in (local), points go out (global)

Function Composition

$$f(g(x)) = (f \circ g)(x)$$
$$f(g(h(x))) = (f \circ g \circ h)(x)$$

but since x is really x,y (and someday x,y,z)

$$f(g(\vec{x})) = (f \circ g)(\vec{x})$$
$$f(g(h(\vec{x}))) = (f \circ g \circ h)(\vec{x})$$

vector notation $\vec{x}$ vs. (x, y) vs. $\mathbf{x}$

What if the functions are **linear**?

Each function is "multiply by a matrix"