Lecture 8: More Transform Math

Review of Last Time

- Matrices and Vectors
- Linear Transformations
- Affine Transformations
- Homogeneous Coordinates

Today

- Composition and Matrices
- Rotations
- Transformations in APIs
- Oriented particles
- Affine Transforms Summary

After Today

- Curves
- 3D

Transformation as a Linear Operator

$\mathbf{x}' = f(\mathbf{x})$

$\mathbf{x}' = \mathbf{F}\,\mathbf{x}$

Composition

$$\mathbf{x}' = h(g(f(\mathbf{x}))) \quad h \xleftarrow{x''} g \xleftarrow{x'} f \leftarrow x$$

$$\mathbf{x}' = \underbrace{(h \circ g \circ f)}(\mathbf{x})$$

code order vs. math order

Composition is Matrix Multiply

$$\mathbf{x}' = h(g(f(\mathbf{x})))$$

$f(x) = F\,x$

$$\mathbf{x}' = \mathbf{H}\,\mathbf{G}\,\mathbf{F}\,\mathbf{x}$$

(Fx)

$\downarrow$

$G(\ )$

$\downarrow$

$H(\ )$

$$\mathbf{x}' = \boxed{(\mathbf{H}\,\mathbf{G}\,\mathbf{F})}\,\mathbf{x}$$

$\downarrow$

E

matrix multiply does not commute!

Order Matters

$$\mathbf{S}\mathbf{T}_1 \neq \mathbf{T}_1\mathbf{S}$$

but...

$$\mathbf{S}\mathbf{T}_1 = \mathbf{T}_2\mathbf{S}$$

Where $\mathbf{T}_2$ is a different translation

this doesn't apply in general, but it works for many transformations

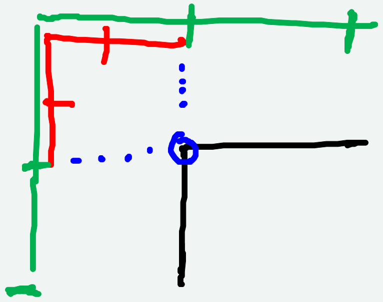
Order changing example

```
scale(2,2);
translate(1,1);
```

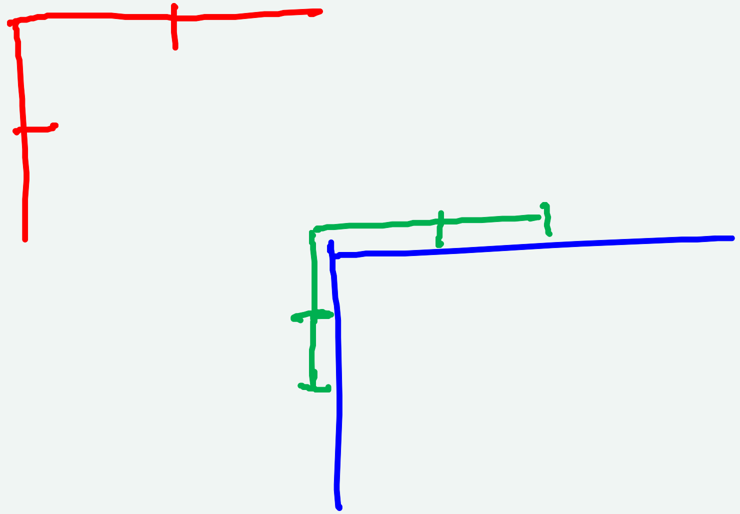
2 2

```
translate(? ,? );
scale(2,2);
```

$$\mathbf{S}_2 \;\; \mathbf{T}_{1,1}$$



$$\mathbf{T}_{?,?} \;\; \mathbf{S}_2$$



Check: put points through (backwards)

```
scale(2,2);
translate(1,1);
```

```
translate(2 ,2 );
scale(2,2);
```

$\mathbf{S_2}\ \mathbf{T_{1,1}} \leftarrow 0,0$

2,2 1,1 3,1

8,4 4,2

$\mathbf{T_{2,2}}\ \mathbf{S_2} \leftarrow 0,0$

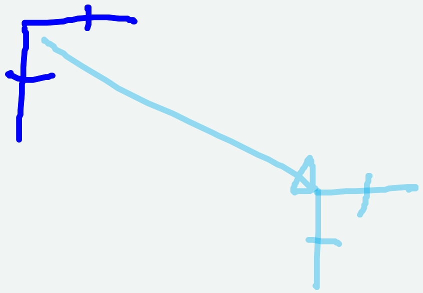
2,2 0,0 3,1

9,4 6,2

Forwards and Backwards

Coordinate systems: left (original) to right (final/current)

Points: right (local) to left (global)



Affine as Linear

$$x' = a\,x + b\,y + t_x$$
$$y' = c\,x + d\,y + t_y$$

or

$$\mathbf{x}' = \mathbf{A}\,\mathbf{x} + \mathbf{t}$$

or

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$s\begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

$$\begin{matrix} x' \\ y' \\ w' \end{matrix} \xrightarrow[\text{convert to 2D}]{} \begin{matrix} x'/w' \\ y'/w' \end{matrix}$$

Transformation Commands

```
context.save();
context.restore();

context.translate(x,y);
context.rotate(r);
context.scale(sx,sy);

context.transform(a,b,c,d,e,f);
```

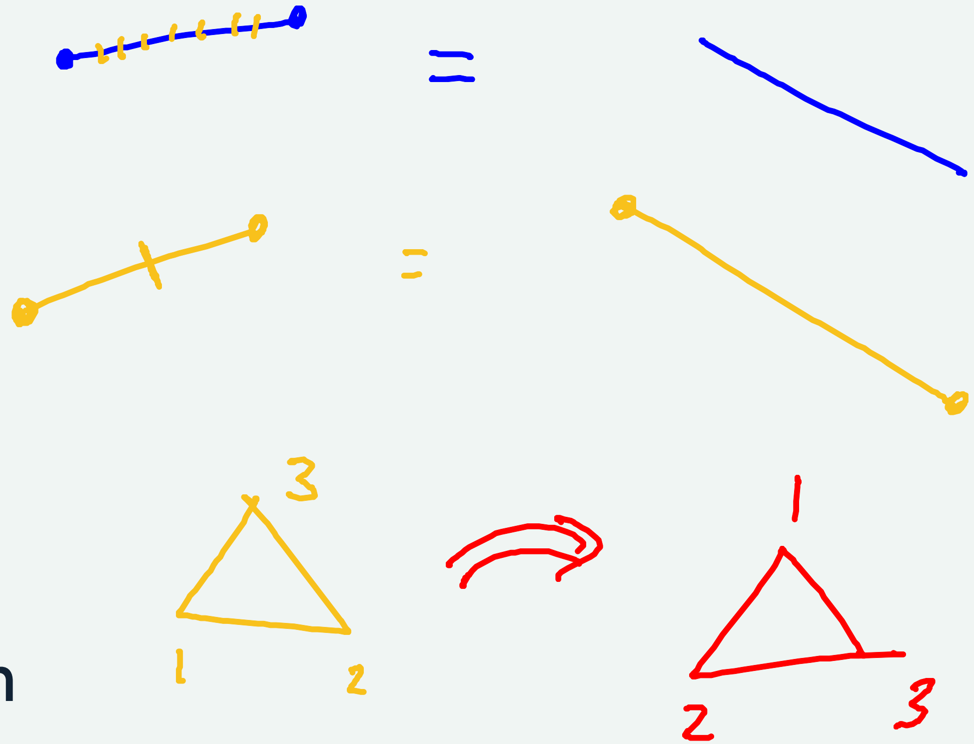
$C_{urr}\ T_{xy} \Rightarrow C_{urr}$

$*=$

$$C_{urr} \mathrel{*}= \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Affine Transformations

- Lines are preserved
  - Ok to just transform endpoints
- Ratios are preserved
  - Halfway will still be halfway
- Polygons are preserved
  - Connected stay connected
- Handedness - could have reflection
  - Clockwise -> ??

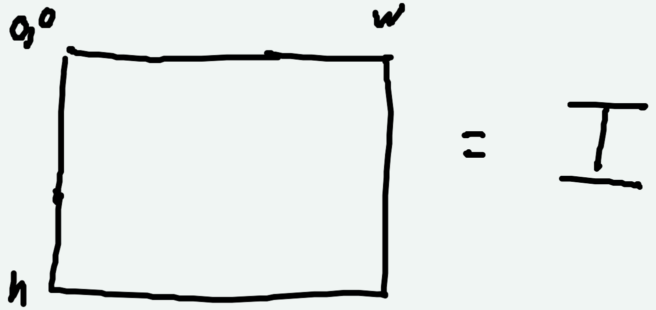


What do transformation do to shapes?

If we change each point...

- General - anything happens
- Affine Transformations - much clearer

Implementation in APIs



- Base, window, device ... coordinates
    - Canvas Coordinates
- Current coordinate system
    - Matrix (map to "Base")

$\overbrace{I \; R \; T \; S}^{curr}$

- Transformation commands multiply transform (on the right)
- Save = copy the current matrix (push onto stack)
- Restore = return to previous matrix (pop off of stack)

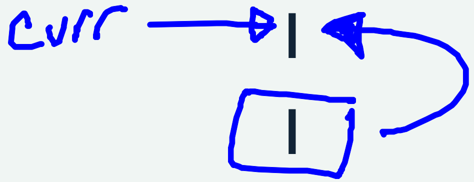
Using a Matrix (without seeing it)

Canvas Coordinates	Transform	Object Coordinates
x, y	I	x, y

```
context.moveTo(x,y);
(etc)
```

Using a Matrix (without seeing it)

Canvas Coordinates | Transform | Object Coordinates




```
context.save();
```

Using a Matrix (without seeing it)

Canvas Coordinates | Transform | Object Coordinates

$\boxed{I}\,T$

I

```
context.save();
context.translate(tx,ty);
```

Using a Matrix (without seeing it)

Canvas Coordinates — Transform — Object Coordinates

(ITR) (x,y) ← $\boxed{\boxed{\text{I T}}\text{R}}$ ← x, y

I

```
context.save();
context.translate(tx,ty);
context.rotate(a);
context.moveTo(x,y);
```

Using a Matrix (without seeing it)

Canvas Coordinates | Transform | Object Coordinates

I T R

I T R

I

```
context.save();
context.translate(tx,ty);
context.rotate(a);
context.save();
```

Using a Matrix (without seeing it)

Canvas Coordinates | Transform | Object Coordinates

$\boxed{\text{I T R S}}$ x,y

I T R

I

```
context.save();
context.translate(tx,ty);
context.rotate(a);
context.save();
context.scale(s,s);
context.moveTo(x,y); DRAW...
```

Using a Matrix (without seeing it)

Canvas Coordinates | Transform | Object Coordinates

I T R

I

```
context.save();
context.translate(tx,ty);
context.rotate(a);
context.save();
context.scale(s,s);
context.moveTo(x,y); DRAW...
context.restore();
```

Using a Matrix (without seeing it)

Canvas Coordinates	Transform	Object Coordinates
	I	

```
context.save();
context.translate(tx,ty);
context.rotate(a);
context.save();
context.scale(s,s);
context.moveTo(x,y); DRAW...
context.restore();
context.restore();
```

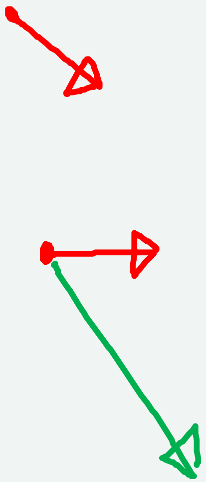
$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} \boxed{\begin{matrix} s_x & 0 \\ 0 & s_y \end{matrix}} & \begin{matrix} 0 \\ 0 \end{matrix} \\ \begin{matrix} 0 & 0 \end{matrix} & 1 \end{bmatrix}$$

$H \quad G \quad F x$

Reading a Matrix

Three Columns:

- where does the x axis go
- where does the y axis go
- where does the origin go



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

- What happens to a point?
- how to achieve goals?
- are things stretched?
- is there a rotation?
- do the axes remain orthogonal?
- decompose into simple trans

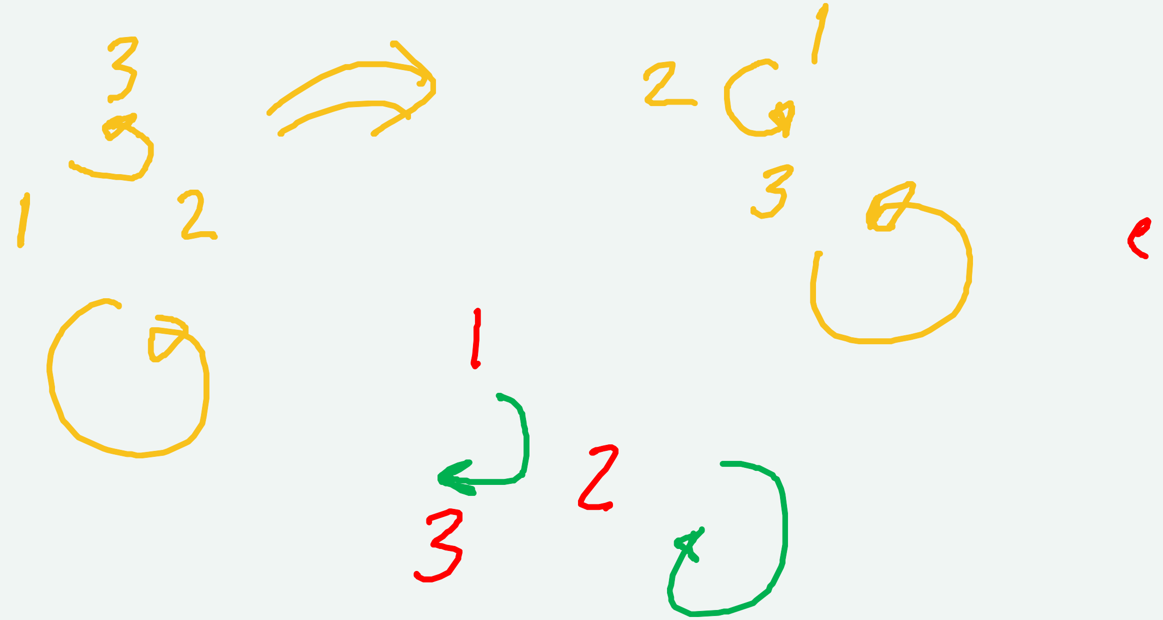
What about rotation?

A transformation that:

- preserves **distances**
- preserves **angles**
- preserves **handedness**

A matrix that:

- each row/colum is **unit length**
- the rows/colums are **orthogonal**
- the determinant is positive



How do you know it is a rotation?

$$1 \quad 0 \qquad \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$$

What happens to the unit X vector?

What happens to the unit Y vector?

- preserve distance

$$\sqrt{a^2 + c^2} = \sqrt{b^2 + d^2} = 1$$

$$\sqrt{a^2 + b^2} = \sqrt{c^2 + d^2} = 1$$

- X and Y remain orthogonal

$$[a, c] \cdot [b, d] = 0$$

- X and Y keep their handedness

  direction fro X to Y is the same

$$det(R) = ad - bc > 0$$

Facts about Rotations

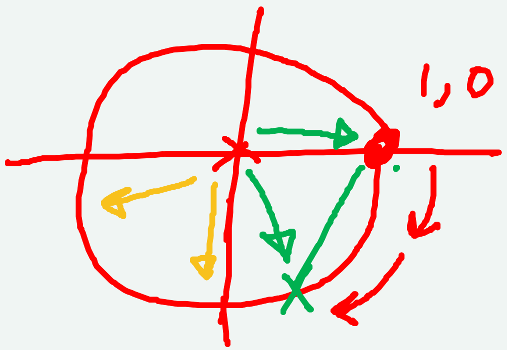
- Orthonormal matrices
- Closed under composition / multiplication
  - $\mathbf{R}_1 \circ \mathbf{R}_2 = \mathbf{R}$
- The inverse is the transpose

normal $\equiv$ unit length

$$\begin{matrix} \cos\theta & -\sin\theta \\ \cancel{-}\sin\theta & \cos\theta \end{matrix}$$

Rotations

- Set of 2D rotations = set of 2D rotation matrices
- How "many" are there?
- One matrix for every point on the unit circle
- Parameterization
    - a "name" for every matrix
    - complex number (point on circle)
    - distance around circle (angle)



A 2D Rotation Matrix

$$\begin{bmatrix}\cos\theta & -\sin\theta \\ sin\theta & cos\theta\end{bmatrix}$$

Things you cannot do...

Given a rotation matrix, you cannot:

- multiply by a scalar
- add a (non-zero) matrix
- multiply by a scale

and get a rotation matrix

What happens if you try to interpolate?

Linear Interpolation

Interpolate (has values at specified points)

$u=0$ $u=1$

Parameter (u)

$$\mathrm{lerp}(\mathrm{a},\mathrm{b},\mathrm{u}) = (1-u)\,a + u\,b$$

goes from a to b as u goes from 0 to 1

works if a and b are scalars, vectors, matrices, ...

Linear Interpolation of Rotation Matrix?

Zero rotation

$$\underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}$$

Halfway

$$\begin{bmatrix} .5 & -.5 \\ .5 & .5 \end{bmatrix}$$

$$\frac{1}{2}(R_1 + R_2)$$

not a rotation!

90 degrees

$$\underbrace{\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}}$$

Linear Interpolation of Rotation Matrix?

Zero rotation

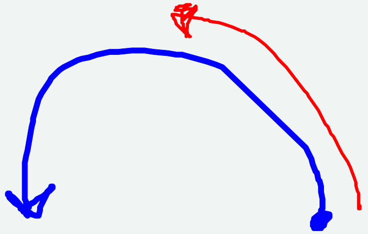
$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Halfway

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

180 degrees

$$\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

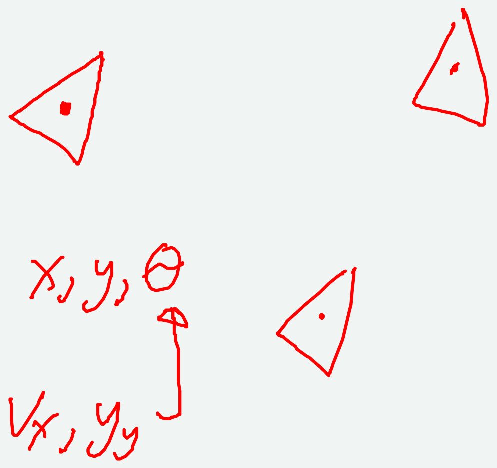


Interpolate an interpolatable representation!

A Use for Rotations...

Oriented "Particles"

"Boids" - Bird-like objects (they flock)



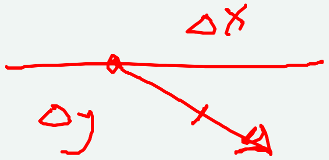
- Keep a constant speed
- Change direction slowly (turn)
- More generally: controlled acceleration and turning

Representation

State (current information)

- Position x, y
- Velocity (vector) - assume it has speed 1

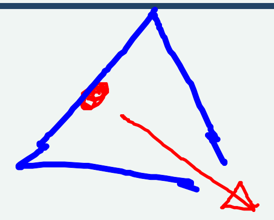
- Position
- Orientation (angle)



Drawing

Face the direction of travel

- compute angle and rotate
- build matrix
- Just use the vector (need the "other direction")



Update

- Position += velocity * time step
- velocity updates?
  - keep magnitude (length)
  - change angle a little
  - rotate

About that update

Stepwise integration

$$\mathbf{p}' = \mathbf{p} + \mathbf{v}$$
$$\mathbf{v}' = \mathbf{A}\ \mathbf{v}$$

A is a *rotation* matrix

or...

$$\mathbf{v_x}' = \cos\theta * speed$$
$$\mathbf{v_y}' = \sin\theta * speed$$

Local models (flocking)

- Decide how to turn by looking at neighbors and world
- Each boid decides independently
- Interesting behaviors emerge from simple rules
  - Flock (align with neighbors)
  - Chase / Avoid

Be careful when doing math on angles (wraparound)

Summary: Transformation Math

$f(x)$ $g\,f(x)$

- Think in terms of functions (composition)
- Think in terms of matrices (linear, affine)
- Homogeneous coordinates make affine linear (in higher dimension)
- Composition by multiplication
- Rotations are special

- All this comes back in 3D (4x4 homogeneous transformations)
  - viewing transforms (projection 3D->2D)