Lecture 8: More Transform Math

Review of Last Time

- Matrices and Vectors
- Linear Transformations
- Affine Transformations
- Homogeneous Coordinates
- Composition
- Transformations in APIs

Today

- Review
- Details of specific transforms (rotations)
- Oriented particles
- Affine Transforms Summary
- [?] some programming tricks

After Today

- Curves
- 3D

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);
```

current transformation

operators right multiply

transformation "stack" (save/restore)

apply current matrix to all points

Example - read top to bottom (move c-systems)

```
context.save();
context.translate(10,0);
context.rotate(Math.PI/2);
context.scale(2,2);
context.moveto(x,y); // and so on

context.save();
context.translate(0,10);
context.rotate(Math.PI/2);
context.scale(2,2);
context.moveto(x,y); // and so on
context.restore();

context.moveto(x,y); // and so on
context.restore();
context.moveto(x,y);
```

In SVG?

Each object has its own coordinate system

Express coordinate systems relative to parent

Composition

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

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

code order vs. math order

Composition is Matrix Multiply

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

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

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

matrix multiply does not commute!

Compose Transformations by multiply

Any sequence of affine transformations can be combined into one

Order Matters

$$\mathbf{ST_1} \neq \mathbf{T_1S}$$

but...

$$\mathbf{ST_1} = \mathbf{T_2S}$$

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);
```

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

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

$$\mathbf{T_{?,?}} \ \ \mathbf{S_2}$$

Check: put points through (backwards)

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

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

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

$\mathbf{T}_{2,2} \ \mathbf{S}_2$

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

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

Reading (or writing) a Matrix

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

Where does the **origin** go?

Where does the **unit X vector** go?

Where does the **unit Y vector** go?

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 -> ??
- Composition
  - any sequence of affine transforms is an affine transform

Reading a Matrix

Three Columns:

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

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

- What happens to a point?
- how to achieve goals?

- are things stretched?
- is there a rotation?
- do the axes remain orthogonal?

- decompose into simple steps

What about rotation?

A transformation that:

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

A matrix that:

- each row/column is **unit length**
- the rows/columns are **orthogonal**
- the determinant is positive

How do you know it is a rotation?

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

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

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)

Parameter (u)

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

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	Halfway	90 degrees
$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$		$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$

Linear Interpolation of Rotation Matrix?

Zero rotation

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

Halfway

180 degrees

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

Interpolate an interpolatable representation!

A Mathematical Aside...

What is **half** of a rotation?

Zero rotation

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

90 degrees

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

180 degrees

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

- half the angle (divide by 2) - angles add
- **M** = **H H** - matrices multiply

  half of a transformation is... the square root!

  matrix square roots are not commonly taught in linear algebra

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
- 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
- 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...

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

How to change direction?

- flip when you hit a wall
  be careful if you cross the wall
- other things ...

Maintain speed

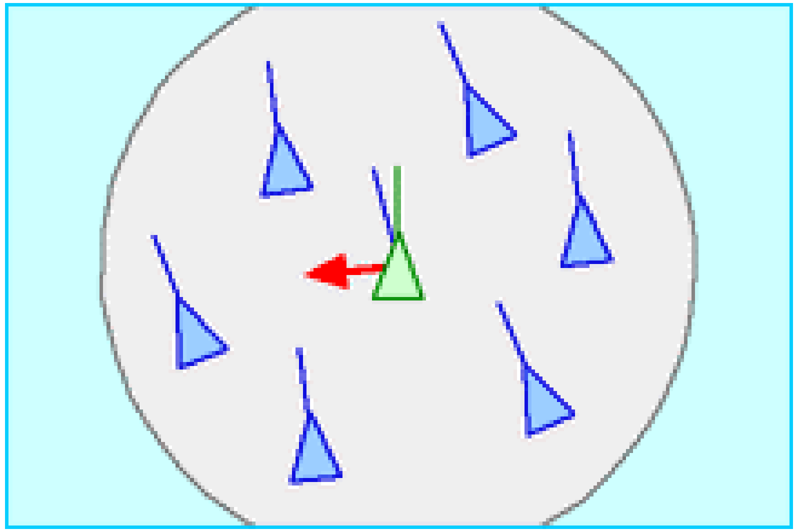
We only turn - we don't change speed!

Local models (flocking)

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

Be careful when doing math on angles (wraparound)

Some examples



Alignment

- find average of neighbor's direction
- turn towards that direction

Notice:

- need to decide who is a neighbor (parameter)
- distance fall-off
- how much to steer towards average

Some Examples

Chase

A "preditor" knows another "prey"

- turn in the direction of prey

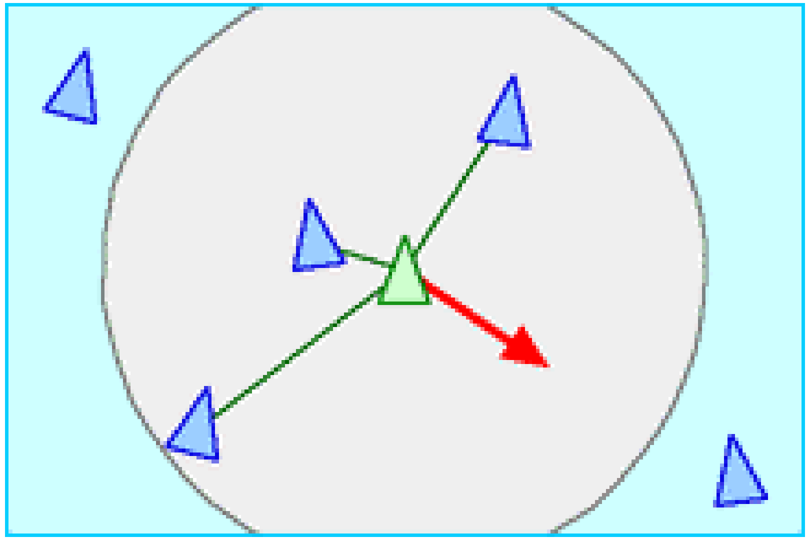
Mouse

When the mouse is clicked, turn towards it

Some Examples

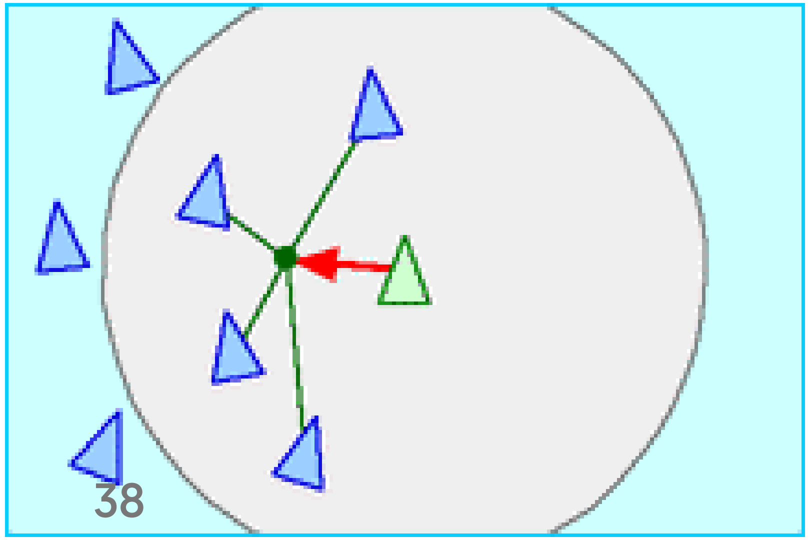
Separation

Find the "center" of the neighbors (average of their positions) - turn away from that



Cohesion

Find the "center" of the neighbors (average of their positions) - turn towards that



JavaScript Tips

Traditional object oriented programming...

```
class Rectangle {
    constructor(x, y, height, width) {
        this.x = x;
        this.y = y;
        this.height = height;
        this.width = width;
    }

    draw(context) {
        context.fillRect(this.x, this.y, this.height, this.width);
    }
}
```

JavaScript Tip of the Day

Beware of **this**!

`this` is a **keyword** not a **variable**

it does not behave like a variable - it is **not** lexically scoped

it has different meaning depending on context

W3 schools lists **6** different meanings of this!

This in methods

In a constructor:

`this` refers to the new (initially empty) object

In a method:

`this` refers to the object the method was called on

Except: Somethings redefine `this`

- Inner functions and event handlers
- special functions (call, apply, maybe others)

Summary: Transformation Math

- 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)