Lecture 7: Transformations Math

Review of Last Time

- Transformations
- Hierarchies, Chains, Trees, DAGs, ...
- Scene-Graph APIs (SVG)

Today

- Transformations as Functions
- Coordinate Systems as Matrices
- Linear Algebra Review
- Transformations as Linear Algebra
- Homogeneous Coordinates

After Today

- Transformation Math
- Curves
- 3D

What does a transformation do to points?

Transformations are functions that apply to points

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

Point (position) goes in

Point (position) comes out

Changes coordinate systems:

- going in (local, original coordinates)
- going out (less-local, new coordinates)

What do transformations do to points?

Transformations are functions that apply to points

$$h\Bigg(\ g\Big(\ f(\mathbf{x})\ \Big)\ \Bigg)$$

We can combine the functions (composition):

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

This says what happens to points

What happens to coordinate systems?

What is a Coordinate System?

Three things:

1. origin (where is 0,0)
2. x "step"
3. y "step"

A piece of "graph paper" that tells us how to interpret coordinates.

The axes do not have to be orthogonal.

Linear combinations

Combine:

- origin
- x steps
- y steps

Interpeted in the "current coordinate system"

Can store these in a Matrix

What is an x step

What is a y step

Where does the origin go (gets added no matter what)

Math you need to know...

Linear algebra in a few minutes

(not really)

Just the parts we'll use

Quickly today... practice later

Why?

Transformations are conveniently expressed as matrices

Vectors and Points (and Tuples)

Both are "arrays"

A **Point** is a place

A **Vector** is a movement

A **Tuple** is a list (of numbers)

A point is the interpretation of a vector in a coordinate system

Vectors (and points)

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

Row Vectors vs. Column Vectors

$$\begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} \qquad \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

They are **matrices** of different shapes

They have the same content (4 numbers)

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 Transpose

Rows become columns (or columns become rows)

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}^T$$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}^T$$

(right) Multiply Matrix by Vector

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \circ \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

(left) Multiply Matrix by Vector

$$\begin{bmatrix} x & y & z \end{bmatrix} \circ \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

Matrix multiply

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \circ \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Matrix Properties

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

What does this have to do with Transformations?

1. Coordinate systems are matrices
    - so changes in systems are matrices as well
2. The most important transformations are linear operations
    - so focus on them

Linear Transformations

Linear combinations of the inputs

$$x' = ax + by$$

$$y' = cx + dy$$

Why do we care?

- Most of what we did has this form
  - rotate, scale, skew - and combinations
- Good for analysis
- Easy to implement
- Guaranteed properties (more later)
- Allows us to use matrices

Change in notation

$$x' = ax + by$$

$$y' = cx + dy$$

rewrite as...

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Matrix Notation

$$\mathbf{x}' = \mathbf{A}\ \mathbf{x}$$

Right multiply convention

Vectors $\mathbf{x} = [x, y]^T$ and $\mathbf{x}' = [x', y']^T$

Matrix $\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$

Warning: Right Multiply

Many old books prefer left multiply

Some APIs are left multiply

Most (modern) descriptions prefer right multiply

Transformation as a Linear Operator

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

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

Composition

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

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

Composition is Matrix Multiply

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

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

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

Properties of Linear Transformations

- Composition by Matrix Multiply
- Lines remain lines
- Ratios are preserved
- Set is closed under composition

- Zero is preserved

What about Translation?

Translation in 2D is not a linear operation in 2D

But, translation is important!

Affine Transformations

Linear transformation plus a translation

Change of center

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

or

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

Affine transformations

How do we compose them?

$$f(\mathbf{x}) = \mathbf{F}\mathbf{x} + \mathbf{t}$$

$$g(\mathbf{x}) = \mathbf{G}\mathbf{x} + \mathbf{u}$$

$$g(f(\mathbf{x})) = g(\mathbf{F}\,\mathbf{x} + \mathbf{t}) = \mathbf{G}\,\mathbf{F}\,\mathbf{x} + \mathbf{G}\,\mathbf{t} + \mathbf{u}$$

Encoding Transforms in Matrices

Affine transforms (in nD) are not linear (in nD)

So work in higher dimensions...

Affine transforms (in nD) are linear (in n+1 D) in **homogeneous coordinates**

1D Example

1D Linear

$f = s\,x$

1D Affine

$f(x) = s\,x + t$

Place the 1D space in 2D

let the "1D space" be y=1

Our 1D "points" x are now [x,1] in 2D

Translation in 1D is Shear in 2D

$$x' = x + t$$

$$\begin{bmatrix} x' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix}$$

Homogeneous Coordinates

Embed an *n* dimensional space in an *n+1* dimensional space

We call the extra dimension *w*

Project back to the original space

Divide by w

What does this matrix do (1)

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

What does this matrix do (2)

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

What does this matrix do (3)

$$\begin{bmatrix} 2 & 0 & 1 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Is the bottom row always [0,0,1]?

Is w always 1?

Is the bottom row always [0,0,1]?

If we limit ourselves to affine, we don't ***need*** anything else

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

Note the order

What does this matrix do? (4)

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

What does this matrix do? (5)

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

Better in the book...

The actual matrices for your favorite transformations

Non-Affine Transformations

Projective Transformations

Useful in 2D (for computer vision)

Useful in 3D (just wait)

Focus on affine for now

Matrices and Coordinate Systems

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

Three Columns: where does the...

- (local) X axis go
- (local) y axis go
- (local) origin go

Matrices move from one coordinate system to another

Works in either direction

Implementation in APIs

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

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

Using a Matrix (without seeing it)

Canvas Coordinates	Transform	Object Coordinates
	I	
	I	

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

Using a Matrix (without seeing it)

Canvas Coordinates	Transform	Object Coordinates
	I T	
	I	

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

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.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
	I T R S	
	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();
```