CS-184: Computer Graphics

Lecture 4: 2D Transformations

Maneesh Agrawala
University of California, Berkeley

Announcements

Assignment 2: due Fri Sep 10 by 11pm
Assignment 3: due Fri Sep 17 by 11pm
Today

2D Transformations

• "Primitive" Operations
  • Scale, Rotate, Shear, Flip, Translate
• Homogenous Coordinates
• SVD
• Start thinking about rotations...

Introduction

Transformation:
An operation that changes one configuration into another

For images, shapes, etc.
Geometric transformation maps positions that define object to new positions
Linear transformation means the transformation is defined by a linear function... which is what matrices are good for:

Some Examples

Original
Some Examples

Images from Conan The Destroyer, 1984

Mapping Function

\[ f(\mathbf{p}) = \mathbf{p}' \]
Maps points in original image \( \mathbf{p} = (x, y) \)
to point in transformed image \( \mathbf{p}' = (x', y') \)

\[ \mathbf{c}(\mathbf{p}) = [195, 120, 58] \]
\[ \mathbf{c}(\mathbf{p}') = \mathbf{c}(f^{-1}(\mathbf{p}')) = \mathbf{c}(\mathbf{p}) \]
Linear -vs- Nonlinear

Linear (shear)
Nonlinear (swirl)

Geometric -vs- Color Space

Linear Geometric (flip)
Color Space Transform (edge finding)

Instancing

M.C. Escher, from Ghostscript 8.0 Distribution
Instancing

Reuse geometric descriptions
Saves memory

Linear is Linear

Composing two linear function is still linear
Transform polygon by transforming vertices

\[
\begin{align*}
  f(x) &= a + bx & g(f) &= c + df \\
  g(x) &= c + df(x) = c + ad + bdx \\
  g(x) &= d' + b'x
\end{align*}
\]
Points in Space

Represent point in space by vector in $\mathbb{R}^n$

- Relative to some origin!
- Relative to some coordinate axes!

Later we’ll add something extra...

Linear Functions in 2D

$x' = f(x, y) = c_1 + c_2x + c_3y$
$y' = f(x, y) = d_1 + d_2x + d_3y$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} + \begin{bmatrix} M_{xx} & M_{xy} \\ M_{yx} & M_{yy} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

$x' = t + M \cdot x$

Linear is Linear

Polygons defined by points
Edges defined by interpolation between two points
Interior defined by interpolation between all points

_linear interpolation_
Linear is Linear

Composing two linear function is still linear
Transform polygon by transforming vertices

Basic Transformations

Basic transforms are: rotate, scale, and translate
Shear is a composite transformation!

Rotations

\[ p' = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} p \]
Rotations

Rotations are positive counter-clockwise
Consistent w/ right-hand rule
Don’t be different...

Note:
• rotate by zero degrees give identity
• rotations are modulo 360 (or $2\pi$)

Rotations

Preserve lengths and distance to origin
Rotation matrices are orthonormal

$$\det(R) = 1 \neq -1$$

In 2D rotations commute...
• But in 3D they won’t!

Scales

Uniform/isotropic
Non-uniform/anisotropic

$$p' = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} p$$
Scales

Diagonal matrices
- Diagonal parts are scale in X and scale in Y directions
- Negative values flip
- Two negatives make a positive (180 deg. rotation)
- Really, axis-aligned scales

Shears

Shears are not really primitive transforms
Related to non-axis-aligned scales
More shortly....
Translation

This is the not-so-useful way:

\[ \begin{bmatrix} p' \end{bmatrix} = p + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \]

Translate

Note that its not like the others.

Arbitrary Matrices

For everything but translations we have:

\[ x' = A \cdot x \]

Soon, translations will be assimilated as well

What does an arbitrary matrix mean?

Singular Value Decomposition

For any matrix, \( A \), we can write SVD:

\[ A = QSR^T \]

where \( Q \) and \( R \) are orthonormal and \( S \) is diagonal

Can also write Polar Decomposition

\[ A = QRSR^T \]

where \( Q \) is still orthonormal but not the same \( Q \).
Decomposing Matrices

We can force $Q$ and $R$ to have $\det = 1$ so they are rotations.

Any matrix is now:
- Rotation\Rotation\Scale\Rotation
- See, shear is just a mix of rotations and scales

Composition

Matrix multiplication composites matrices

$p' = BAp$

“Apply $A$ to $p$ and then apply $B$ to that result.”

$p' = B(Ap) = (BA)p = Cp$

Several transformations composited to one

Translations still left out...

$p' = B(Ap + t) = xAp + Bt = Cp + u$

Composition

Transformations built up from others

SVD builds from scale and rotations

Also build other ways

i.e. 45 deg rotation built from shears
Homogeneous Coordinates

Move to one higher dimensional space
• Append a 1 at the end of the vectors
\[ \mathbf{p} = \begin{bmatrix} p_x \\ p_y \end{bmatrix} \quad \tilde{\mathbf{p}} = \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix} \]

• For directions the extra coordinate is a zero

Now everything looks the same...
Hence the term “homogenized!”

Homogeneous Translation

\[ \tilde{\mathbf{p}}' = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix} \]

\[ \tilde{\mathbf{p}}' = \tilde{\mathbf{A}}\tilde{\mathbf{p}} \]

The tildes are for clarity to distinguish homogenized from non-homogenized vectors.

Homogeneous Others

\[ \tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A} & 0 \\ 0 & 0 & 1 \end{bmatrix} \]

Now everything looks the same...
Hence the term “homogenized!”
Compositing Matrices

Rotations and scales always about the origin
How to rotate/scale about another point?

rotate vs scale

Rotate About Arb. Point

Step 1: Translate point to origin

Translate (-C)

Rotate About Arb. Point

Step 1: Translate point to origin
Step 2: Rotate as desired

Translate (-C)
Rotate (θ)
Rotate About Arb. Point

Step 1: Translate point to origin
Step 2: Rotate as desired
Step 3: Put back where it was

\[
\begin{align*}
\tilde{p}' &= \begin{pmatrix} -1 \end{pmatrix} R T \tilde{p} = A \tilde{p} \\
\text{Don't negate the 1...}
\end{align*}
\]

Scale About Arb. Axis

Diagonal matrices scale about coordinate axes only:

Not axis-aligned

Scale About Arb. Axis

Step 1: Translate axis to origin
Scale About Arb. Axis

Step 1: Translate axis to origin
Step 2: Rotate axis to align with one of the coordinate axes
Step 3: Scale as desired
Steps 4&5: Undo 2 and 1 (reverse order)
Order Matters!

The order that matrices appear in matters

\[ \mathbf{A} \cdot \mathbf{B} \neq \mathbf{B} \cdot \mathbf{A} \]

Some special cases work, but they are special

But matrices are associative

\[(\mathbf{A} \cdot \mathbf{B}) \cdot \mathbf{C} = \mathbf{A} \cdot (\mathbf{B} \cdot \mathbf{C})\]

Think about efficiency when you have many points to transform...

Matrix Inverses

In general: \( \mathbf{A}^{-1} \) undoes effect of \( \mathbf{A} \)

Special cases:

- Translation: negate \( t_x \) and \( t_y \)
- Rotation: transpose
- Scale: invert diagonal (axis-aligned scales)

Others:

- Invert matrix
- Invert SVD matrices

Point Vectors / Direction

Points in space have a 1 for the “w” coordinate

What should we have for \( \mathbf{a} - \mathbf{b} \) ?

- \( w = 0 \)
- Directions not the same as positions
- Difference of positions is a direction
- Position + direction is a position
- Direction + direction is a direction
- Position + position is nonsense
Some Things Require Care

For example, normals transform abnormally

\[ n^T t = 0 \quad t_M = M t \]

find \( N \) such that \( n_N^T t_M = 0 \)

\[ n^T t = n^T t t = n^T M^{-1} M t = 0 \]
Some Things Require Care

For example normals transform abnormally

\[ n^T t = 0 \quad t_M = Mt \quad \text{find } N \text{ such that } n^T_N t_M = 0 \]

\[
\begin{align*}
  n^T t &= n^T t_M = n^T M^{-1} Mt = 0 \\
  (n^T M^{-1})^T t_M &= 0 \\
  n^T_N &= n^T M^{-1} \\
  n_N &= (n^T M^{-1})^T \\
  N &= (M^{-1})^T 
\end{align*}
\]

See book for details

Suggested Reading

Fundamentals of Computer Graphics by Pete Shirley

- Chapter 6
- And re-read chapter 5 if your linear algebra is rusty!