CS-184: Computer Graphics

Lecture #4: 2D Transformations

Prof. James O'Brien
University of California, Berkeley

V2013-F-04-1.0

1

Today

- 2D Transformations
  - "Primitive" Operations
    - Scale, Rotate, Shear, Flip, Translate
  - Homogenous Coordinates
  - SVD
  - Start thinking about rotations...

2

2

Introduction

- Transformation:
  An operation that changes one configuration into another
- For images, shapes, *etc.*
  A geometric transformation maps positions that define the object to other positions
  Linear transformation means the transformation is defined by a linear function... which is what matrices are good for.

3

3

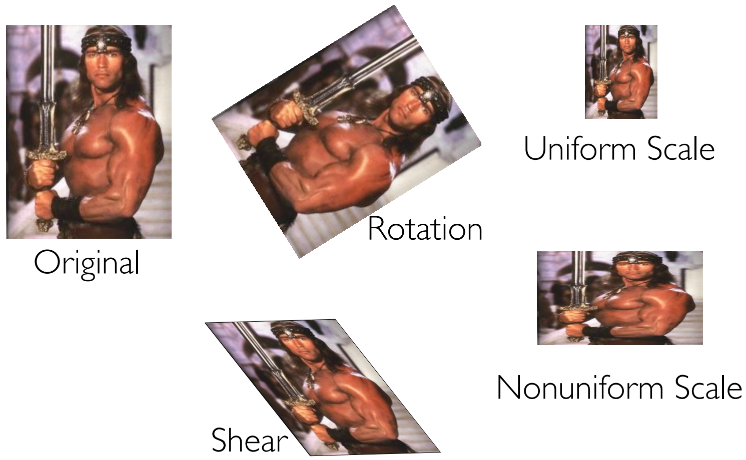
Some Examples

Original

4

4

Some Examples



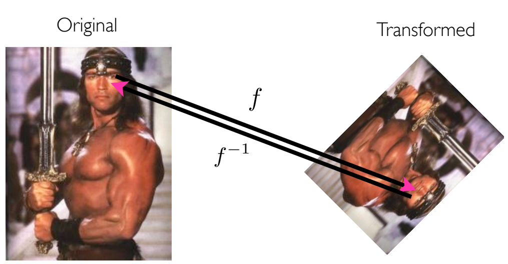
Images from ***Conan The Destroyer***, 1984

5

5

Mapping Function

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

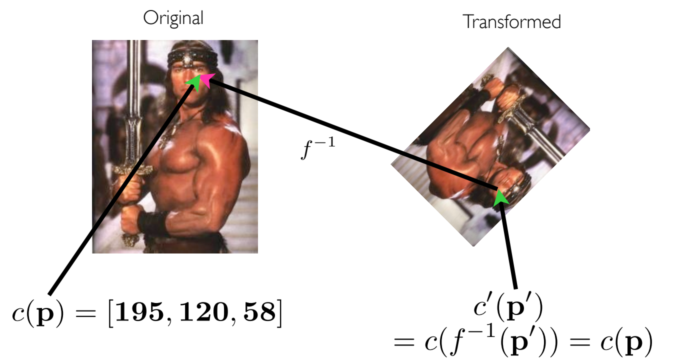


6

6

Mapping Function

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



7

7

Linear -vs- Nonlinear

Nonlinear (swirl)

Linear (shear)

8

8

Geometric -vs- Color Space

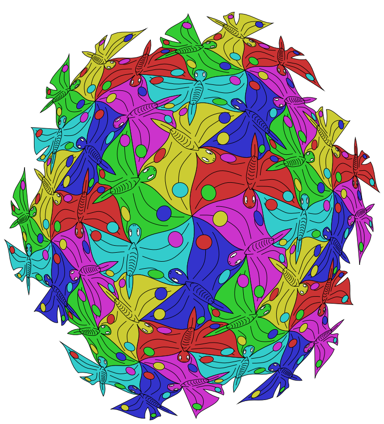
Color Space Transform
(edge finding)

Linear Geometric
(flip)

9

9

Instancing



M.C. Escher, from Ghostscript 8.0 Distribution

10

10

Instancing

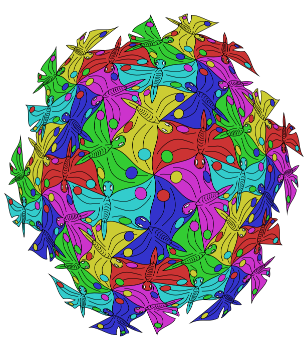
Carlo Sequin

11

11

Instancing

- Reuse geometric descriptions
- Saves memory

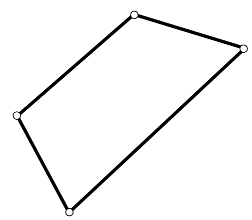


12

12

Linear is Linear

- Polygons defined by points
- Edges defined by interpolation between two points
- Interior defined by interpolation between all points
- ***Linear* interpolation**

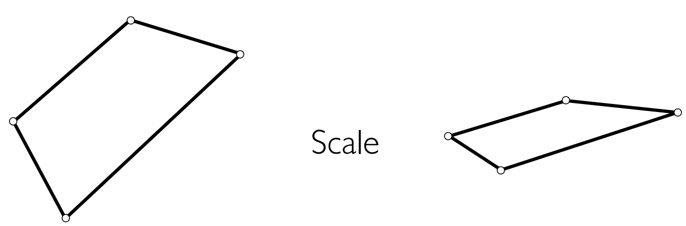


13

13

Linear is Linear

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



14

14

Linear is Linear

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

$$f(x) = a + bx \qquad g(f) = c + df$$

$$g(x) = c + df(x) = c + ad + bdx$$

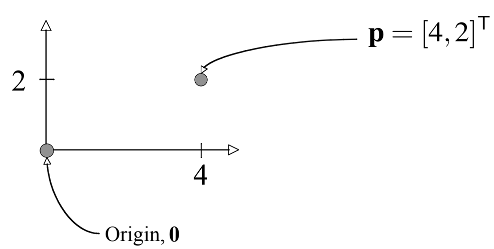
$$g(x) = a' + b'x$$

15

15

Points in Space

- Represent point in space by vector in R^n
  - Relative to some origin!
  - Relative to some coordinate axes!
  - The choice of coordinate system is arbitrary and should be *convenient*.
- Later we'll add something extra...

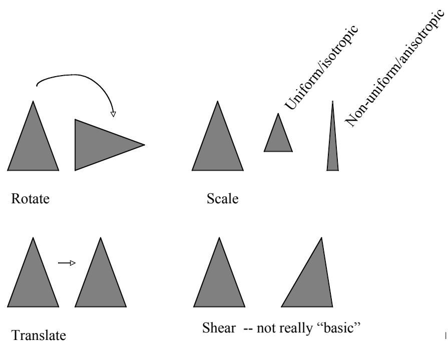


16

16

Basic Transformations

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



17

17

Linear Functions in 2D

$$x' = f(x,y) = c_1 + c_2 x + c_3 y$$
$$y' = f(x,y) = d_1 + d_2 x + d_3 y$$

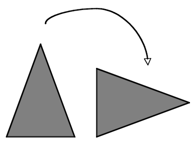
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} + \begin{bmatrix} M_{xx} & M_{xy} \\ M_{yx} & M_{yy} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\mathbf{x}' = \mathbf{t} + \mathbf{M} \cdot \mathbf{x}$$

18

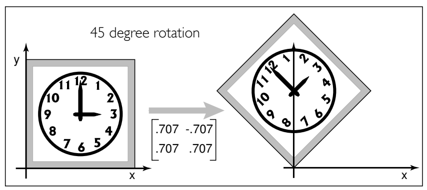
18

Rotations



Rotate

$$\mathbf{p}' = \begin{bmatrix} \mathrm{Cos}(\theta) & -Sin(\theta) \\ Sin(\theta) & \mathrm{Cos}(\theta) \end{bmatrix} \mathbf{p}$$



19

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 2π)

20

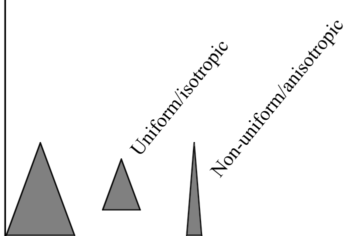
Rotations

- Preserve lengths and distance to origin
- Rotation matrices are orthonormal
- $\mathrm{Det}(\mathbf{R}) = 1 \neq -1$
- In 2D rotations commute...
  - But in 3D they won't!

21

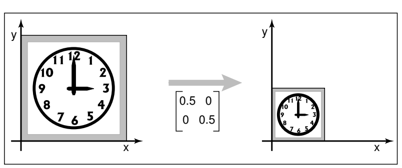
21

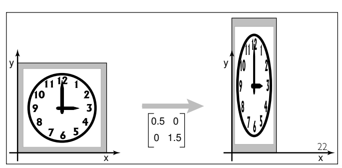
Scales



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

Scale



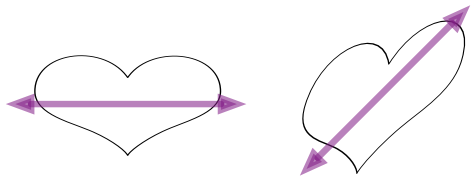


22

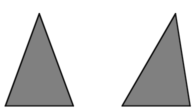
22

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

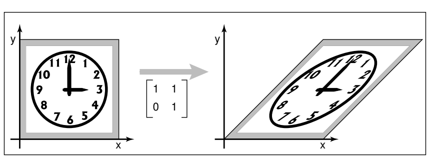


Shears



Shear

$$\mathbf{p}' = \begin{bmatrix} 1 & H_{yx} \\ H_{xy} & 1 \end{bmatrix} \mathbf{p}$$



Shears

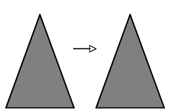
- Shears are not really primitive transforms
- Related to non-axis-aligned scales
- More shortly.....

25

25

Translation

- This is the not-so-useful way:



Translate

$$\mathbf{p}' = \mathbf{p} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Note that its not like the others.

26

26

Arbitrary Matrices

- For everything but translations we have:

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

- Soon, translations will be assimilated as well
- What does an arbitrary matrix mean?

27

27

Singular Value Decomposition

- For any matrix, $\mathbf{A}$, we can write SVD:

$$\mathbf{A} = \mathbf{QSR}^{\mathsf{T}}$$

where $\mathbf{Q}$ and $\mathbf{R}$ are orthonormal and $\mathbf{S}$ is diagonal

- Can also write Polar Decomposition

$$\mathbf{A} = \mathbf{PRSR}^{\mathsf{T}}$$

where $\mathbf{P}$ is also orthonormal $\quad \mathbf{P} = \mathbf{QR}^{\mathsf{T}}$

28

28

Decomposing Matrices

- We can force **P** 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

$$\mathbf{p}' = \mathbf{BAp}$$

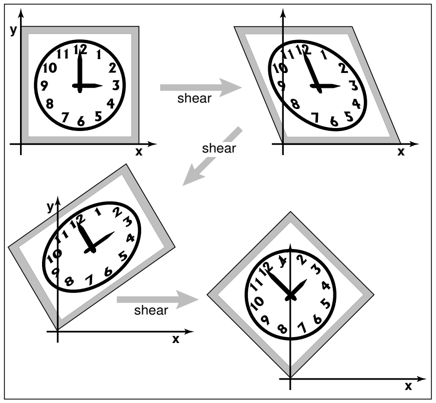
"Apply **A** to **p** and then apply **B** to that result."

$$\mathbf{p}' = \mathbf{B}(\mathbf{Ap}) = (\mathbf{BA})\mathbf{p} = \mathbf{Cp}$$

- Several translations composted to one
- Translations still left out...

$$\mathbf{p}' = \mathbf{B}(\mathbf{Ap} + \mathbf{t}) = \mathbf{BAp} + \mathbf{Bt} = \mathbf{Cp} + \mathbf{u}$$

Composition



Transformations built up from others

SVD builds from scale and rotations

Also build other ways

i.e. 45 deg rotation built from shears

31

31

Homogeneous Coordinates

- Move to one higher dimensional space
  - Append a 1 at the end of the vectors

$$\mathbf{p} = \begin{bmatrix} p_x \\ p_y \end{bmatrix} \qquad \tilde{\mathbf{p}} = \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix}$$

  - For *directions* the extra coordinate is a zero

32

32

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.

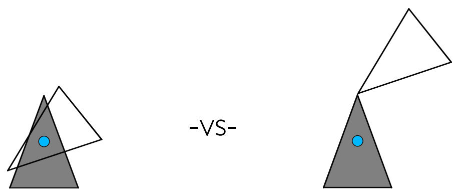
Homogeneous Others

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A} & & \begin{matrix} 0 \\ 0 \end{matrix} \\ 0 & 0 & 1 \end{bmatrix}$$

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?



35

Rotate About Arb. Point

- Step 1: Translate point to origin

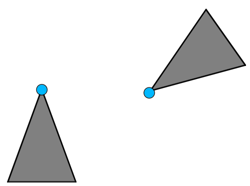
36

Rotate About Arb. Point

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

Translate (-**C**)

Rotate (θ)



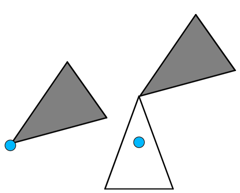
Rotate About Arb. Point

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

Translate (-**C**)

Rotate (θ)

Translate (**C**)



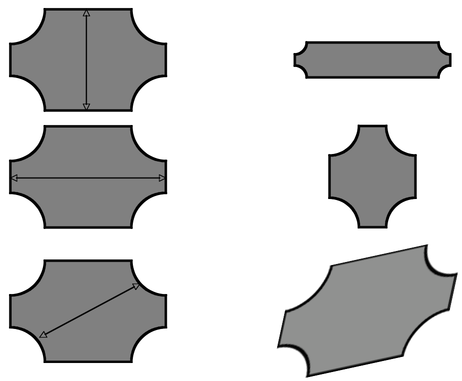
$$\tilde{\mathbf{p}}' = (-\mathbf{T})\mathbf{R}\mathbf{T}\tilde{\mathbf{p}} = \mathbf{A}\tilde{\mathbf{p}}$$

Don't negate the 1...

Scale About Arb. Axis

• Diagonal matrices scale about coordinate axes only:

Not axis-aligned

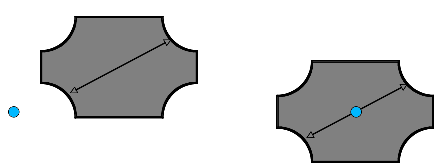


39

39

Scale About Arb. Axis

• Step 1: Translate axis to origin

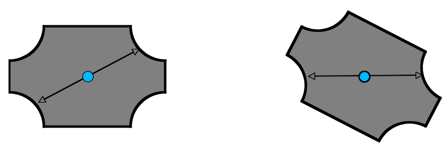


40

40

Scale About Arb. Axis

- Step 1: Translate axis to origin
- Step 2: Rotate axis to align with one of the coordinate axes

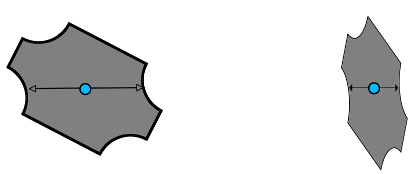


41

41

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

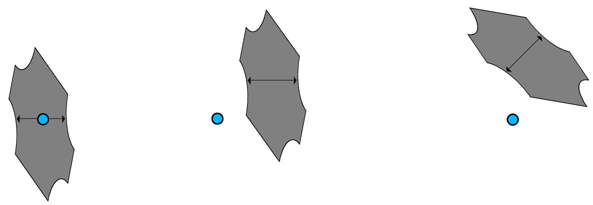


42

42

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

$$\mathbf{A} \cdot \mathbf{B} \neq \mathbf{BA}$$

- 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...

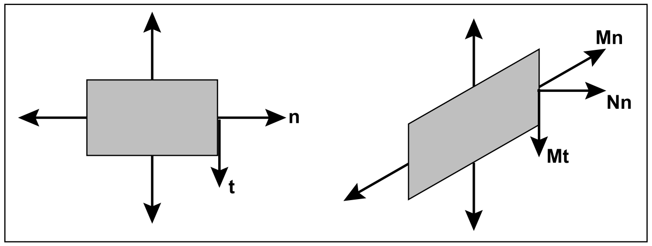
Matrix Inverses

- 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 Vectors

- 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

Somethings Require Care

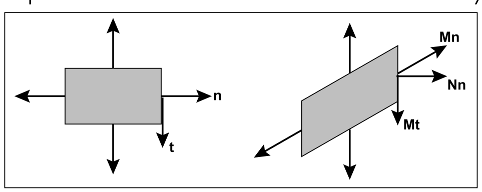


For example normals do not transform normally

$$\mathbf{M}(\mathbf{a} \times \mathbf{b}) \neq (\mathbf{Ma}) \times (\mathbf{Mb})$$

Some Things Require Care

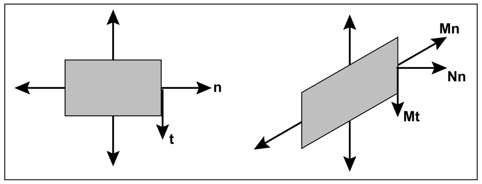
For example normals transform abnormally



$\mathbf{n^T t} = 0$ $\mathbf{t_M} = \mathbf{Mt}$ find $\mathbf{N}$ such that $\mathbf{n_N^T t_M} = 0$

Some Things Require Care

For example normals transform abnormally



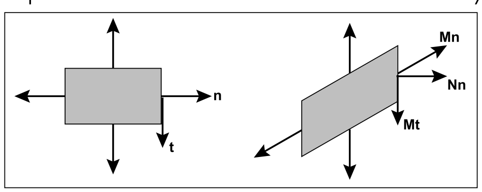
$\mathbf{n^T t} = 0 \quad \mathbf{t_M} = \mathbf{Mt}$ find $\mathbf{N}$ such that $\mathbf{n_N^T t_M} = 0$

$$\mathbf{n^T t} = \mathbf{n^T I t} = \mathbf{n^T M^{-1} M t} = 0$$

49

Some Things Require Care

For example normals transform abnormally



$\mathbf{n^T t} = 0 \quad \mathbf{t_M} = \mathbf{Mt}$ find $\mathbf{N}$ such that $\mathbf{n_N^T t_M} = 0$

$$\mathbf{n^T t} = \mathbf{n^T I t} = \mathbf{n^T M^{-1} M t} = 0$$

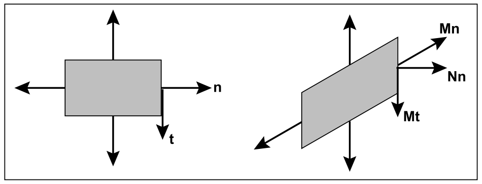
$$(\mathbf{n^T M^{-1}})\mathbf{t_M} = 0$$

$$\mathbf{n_N^T} = \mathbf{n^T M^{-1}}$$

50

Some Things Require Care

For example normals transform abnormally



$\mathbf{n^T t} = 0 \quad \mathbf{t_M} = \mathbf{Mt}$ find $\mathbf{N}$ such that $\mathbf{n_N^T t_M} = 0$

$$\mathbf{n^T t} = \mathbf{n^T I t} = \mathbf{n^T M^{-1} M t} = 0$$
$$(\mathbf{n^T M^{-1}})\mathbf{t_M} = 0$$
$$\mathbf{n_N^T} = \mathbf{n^T M^{-1}}$$
$$\mathbf{n_N} = (\mathbf{n^T M^{-1}})^\mathbf{T}$$
$$\mathbf{N} = (\mathbf{M^{-1}})^\mathbf{T}$$

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!