

# CS174A Lecture 5

# Announcements & Reminders

---

- *10/16/22: A2 due; will be discussed during this week's TA session*
- *10/27/22: Midterm Exam: 6:00 – 7:30 PM PST, in person, in class*
- *11/08/22: Team project proposals due, initial version*
- *11/09/22: A3 due*
- *11/10/22: Midway demo, online zoom*
- *12/06/22: Final Exam: 6:30 – 9:30 PM PST, in person, in class*
- *Updated syllabus on Canvas*

# Team Project Proposal Template

---

- *Approximately 1 page*
- *What is the theme (or story) of your animation?*
- *What topics learnt in the course is used and how?*
- *What interactivity will you use?*
- *What advanced features you'll be implementing?*
- *List of team members*
- *Rough sketch of your 3D digital environment*

# Last Lecture Recap

---

- *Interpolations: linear, affine, convex & their properties*
- *Homogeneous representation of points & vectors*
- *Shapes: lines, circles, polygons (**triangles**)*
- *3D Polyhedrons: indexed data structure*

# Next Up

---

- ***Transformations:***
  - translations, rotations, scaling, shear
  - Matrix representations
  - Inverse of these transformations
- ***Spaces:***
  - Model space
  - Object/world space
  - Eye/camera space
  - Screen space

# SIGGRAPH trailers from 2015

Going backwards,

<https://www.youtube.com/watch?v=RvXtjANeuJA>

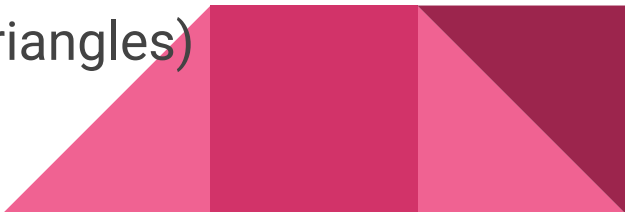
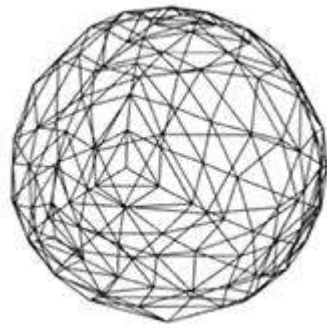
And

<https://www.youtube.com/watch?v=XrYkEhs2FdA>



# Polygons

- An ordered sequence of vertices
- Simple vs. Complex
  - Issue with Complex: hard to deal with self-intersecting polygons
- Convex vs. Concave
  - Issue with Concave: point-inside-polygon test will fail in some cases
  - Issue with Concave: normal will point in wrong direction at concave vertices
- Planar vs. Non-Planar
  - Issue with Non-Planar: normal are different at different locations on the non-planar face
- Tessellation (any poly) vs. Triangulation (only triangles)



# Why Triangles?

- Ordered list of 3 non-collinear vertices
- They are simple
- They are planar
- They are convex
- Modern GPUs: 100M triangles/sec
- Shading, lighting, texture mapping all in GPU
- WebGL uses triangles only for rendering





# What do we do with our point lists?

- **Immediate mode rendering:** Send them one at a time to GPU to draw every point of every triangle. Bad
- **Retained mode:** Store them all in a data structure
  - Now we can store one matrix to represent the whole shape's movement/positioning, instead of repeating ourselves on the point-by-point level
  - Can draw the same shape in many places this way
- **Better yet:** Load the data onto the GPU once and re-use it there
  - Fewer slow GPU communications



# Matrix Intro

Types of matrices found in graphics

# What does a matrix mean?

- A system of (first order) equations
- Our mathematical tool to tweak 3D functions or point clouds with
- Remember that a matrix is shorthand:

$$\begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \end{bmatrix} = \begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} \Rightarrow \begin{aligned} x_{new} &= 2x_{old} + 3y_{old} \\ y_{new} &= 4x_{old} + 5y_{old} \end{aligned}$$



# Reminder: Matrix Multiplication

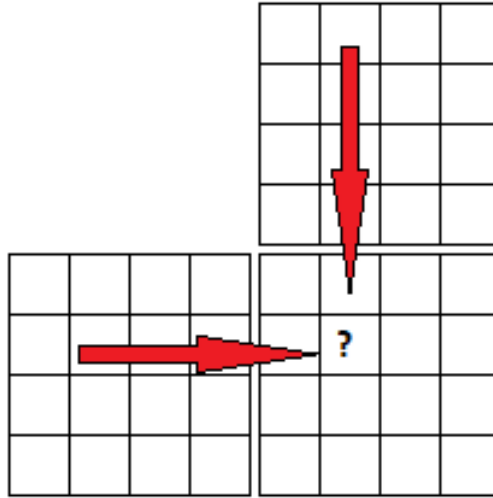
- $A * B = M$
- Each cell of  $M$  is going to be a dot product of one row of  $A$  and one column of  $B$ .
- Remember the rule:
  - Row size comes from  $A$ , column size comes from  $B$
- If  $B$  is just a vector (1 column), that's fine,  $M$  will be too



# Reminder: Matrix Multiplication

- It's helpful to line up A and B offset when you write them down

$$\begin{array}{c} [B] \\ \downarrow \\ [A] \Rightarrow [?] \end{array}$$



# Matrix Concepts

- In Graphics: Guiding a shape into place in a scene
- A few common graphics matrices perform movement, or “warp” space
- What kind? Recall elementary row operations



## Elementary Matrices

An elementary matrix  $E$  is an  $n \times n$  matrix that can be obtained from the identity matrix  $I_n$  by one elementary row operation.

$$E = e(I)$$

where  $e$  is an elementary row operation.

- All elementary matrices are invertible, an inverse exists.
- The inverse of an elementary matrix is also an elementary matrix.

Recall identity matrices

$$[1] \quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \dots$$

Row Replacement:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & 0 & 1 \end{bmatrix} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g-4a & h-4b & i-4c \end{bmatrix}$$

Interchange of Rows:

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

Multiplication by a constant:

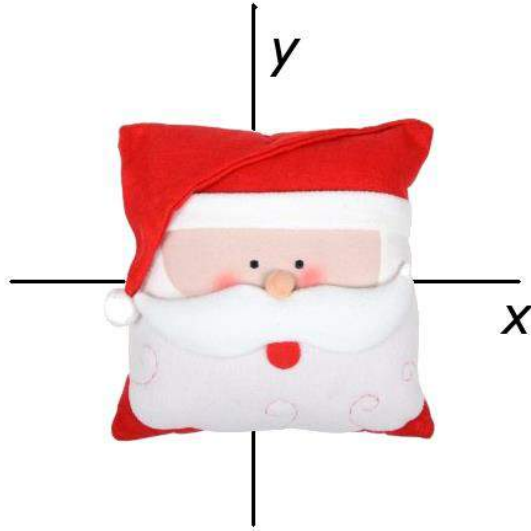
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 5 \end{bmatrix} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 5g & 5h & 5i \end{bmatrix}$$



What if we apply  $e_1$  to all the points of an image?

Interchange of Rows:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = ?$$



# What if we apply $e_1$ to all the points of an image?

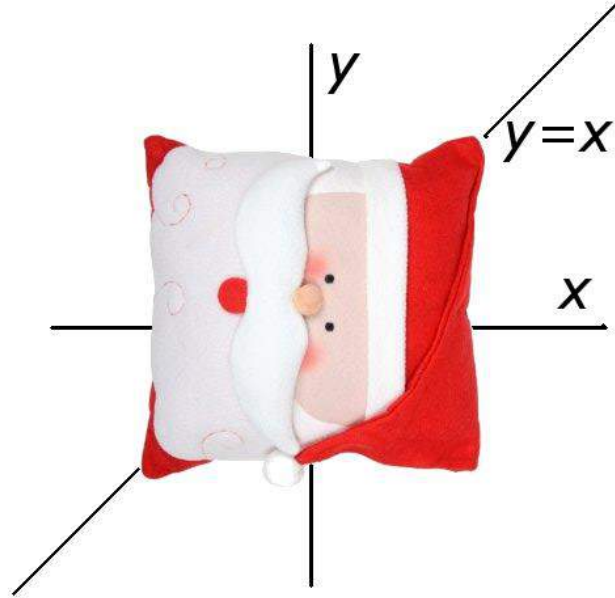
Our axes swap

$\text{new}_x = y, \text{new}_y = x$

Matrix form of that:

Interchange of Rows:

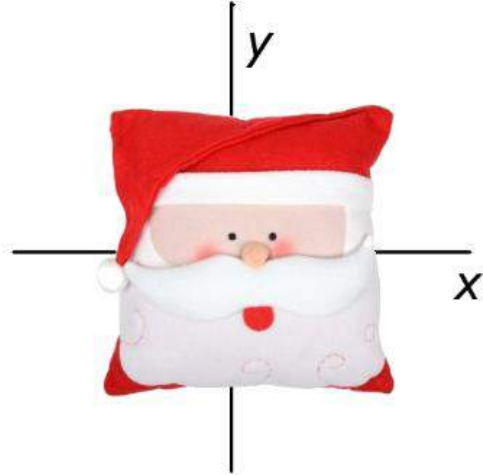
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = ?$$



# What if we apply $e_2$ to all the points of an image?

Multiplication by a constant:

$$\begin{bmatrix} 5 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = ?$$



# What if we apply $e_2$ to all the points of an image?

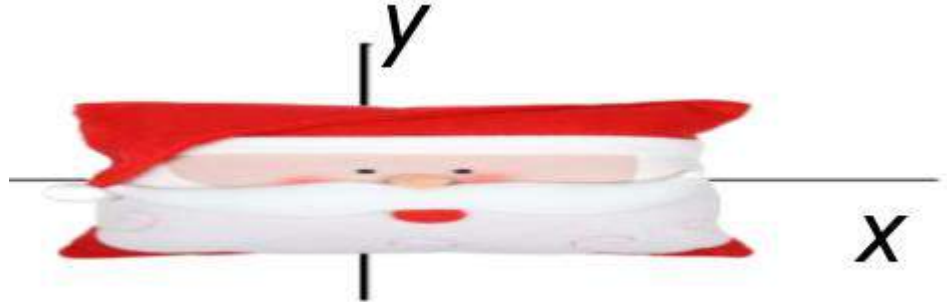
We get a scale.

$$\text{new}_x = 5x$$

Matrix form of that:

Multiplication by a constant:

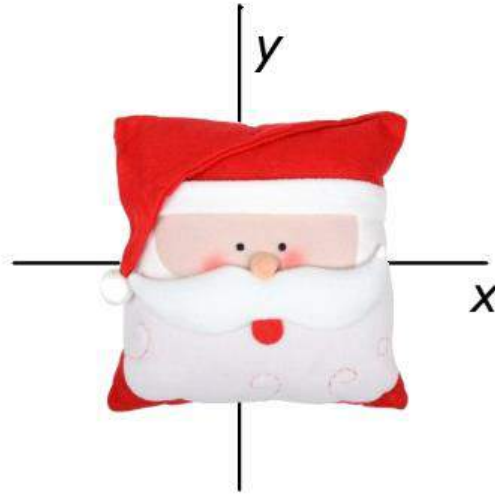
$$\begin{bmatrix} 5 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = ?$$



What if we apply  $e_3$  to all the points of an image?

Row Replacement:

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = ?$$



# What if we apply $e_3$ to all the points of an image?

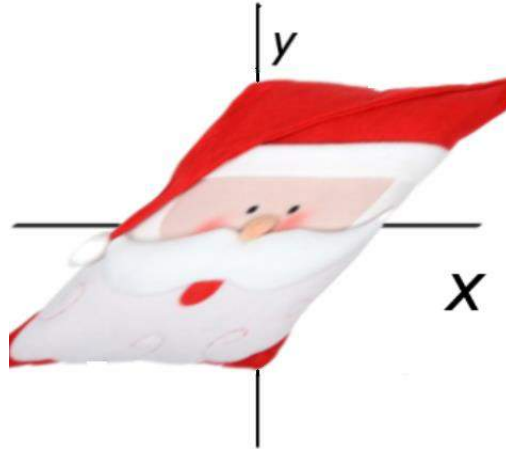
We get a “shear.”

$$\text{new}_x = x + y$$

Matrix form of that:

Row Replacement:

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = ?$$

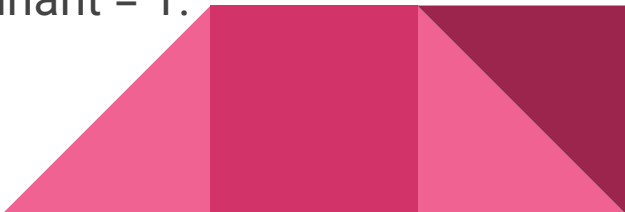


# Rotation Matrices

A rotation is two shears.

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

But certain properties have to hold. This one is not a pure rotation matrix (it has some scaling effect too). Rotations have to have determinant = 1.

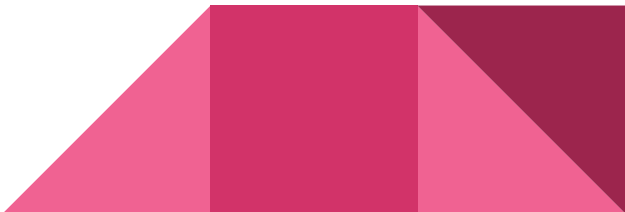


# Rotation Matrices

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

A rotation also has to have orthogonal columns. This one passes that!

Our simpler shear matrix from earlier did not (check the columns):

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




# Rotation Matrices

$$\begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \\ & & 1 \end{bmatrix}$$

This one is more like it. **Trig** identities ensure this matrix always has determinant=1. The two columns are also always perpendicular. Suppose theta is 45 degrees:



# Rotation Matrices

$$\begin{bmatrix} \cos\Theta & -\sin\Theta & \\ \sin\Theta & \cos\Theta & \\ & & 1 \end{bmatrix}$$

This one is more like it. **Trig** identities ensure this matrix always has determinant=1. The two columns are also always perpendicular. Suppose theta is 45 degrees:

$$\begin{bmatrix} \cos 45^\circ & -\sin 45^\circ & \\ \sin 45^\circ & \cos 45^\circ & \\ & & 1 \end{bmatrix} = \begin{bmatrix} .5\sqrt{2} & -.5\sqrt{2} & \\ .5\sqrt{2} & .5\sqrt{2} & \\ & & 1 \end{bmatrix}$$

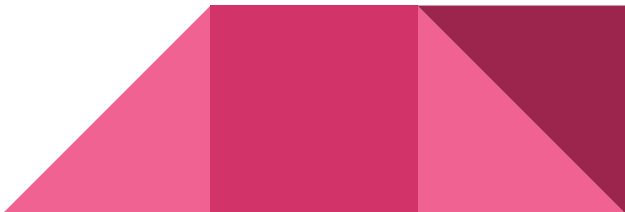


# Rotation Matrices

Let's try our rotation matrix, but we don't want to multiply ugly radicals so let's combine it with a scale matrix for testing.

$$\begin{bmatrix} .5\sqrt{2} & -.5\sqrt{2} & \\ .5\sqrt{2} & .5\sqrt{2} & \\ & & 1 \end{bmatrix} * \begin{bmatrix} \sqrt{2} & & \\ & \sqrt{2} & \\ & & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & \\ 1 & 1 & \\ & & 1 \end{bmatrix}$$

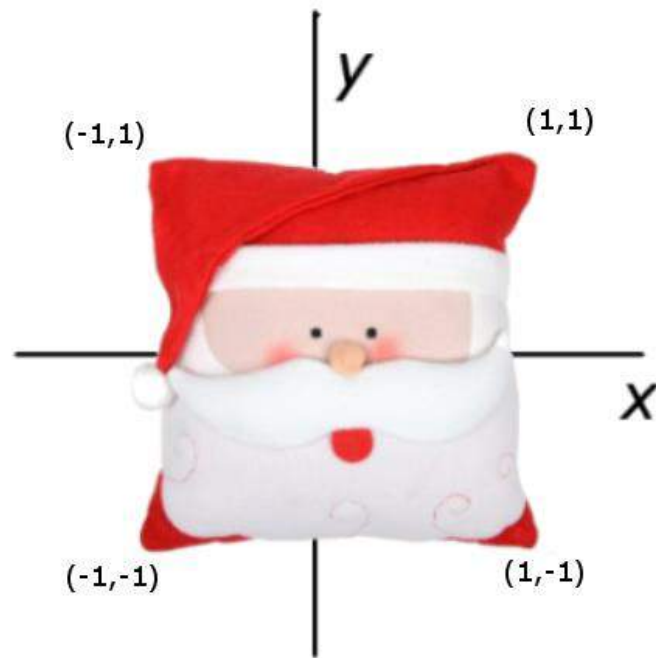
There, much better to work with. (This was our matrix from earlier that had a scaling influence).



# Rotation Matrices

What it means:

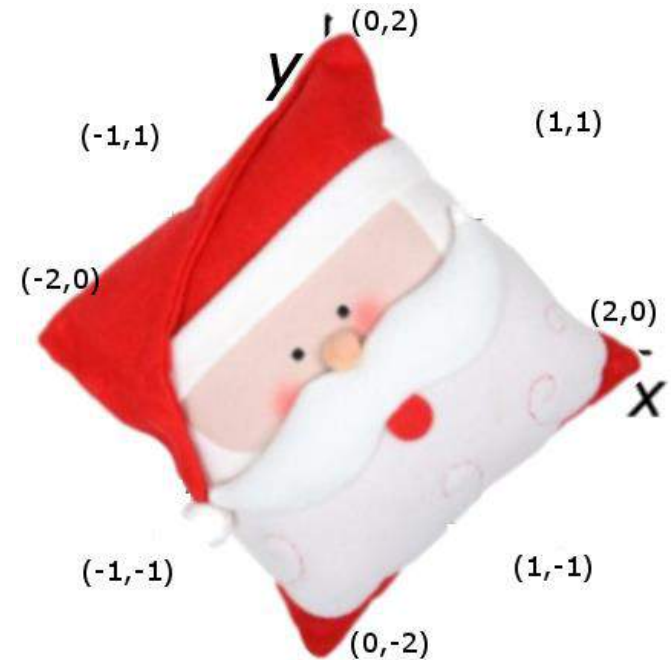
$$\begin{bmatrix} 1 & -1 \\ 1 & 1 \\ & & 1 \end{bmatrix} \Rightarrow \begin{aligned} x_{new} &= x_{old} - y_{old} \\ y_{new} &= x_{old} + y_{old} \end{aligned}$$



# Rotation Matrices

What it means:

$$\begin{bmatrix} 1 & -1 \\ 1 & 1 \\ & & 1 \end{bmatrix} \Rightarrow \begin{aligned} x_{new} &= x_{old} - y_{old} \\ y_{new} &= x_{old} + y_{old} \end{aligned}$$



# All this time...

SANTA'S NOSE HAS NEVER LEFT THE ORIGIN (!!!)

None of these operations have resulted in **translation** (sliding all of the points at once somewhere else)

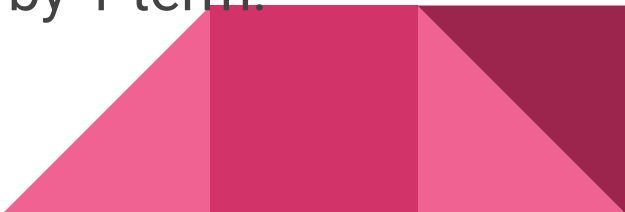


# Translation Matrices

- Every matrix we've discovered using the elementary ones only has a linear effect, stretching and warping the image around the origin
- Why can't we accomplish translation of Santa to other places with our current machinery?



# Translation Matrices

- Every point always takes a contribution of  $x_{\text{old}}$ ,  $y_{\text{old}}$ , and  $z_{\text{old}}$  to get its value
  - The origin point (0,0,0) will always contribute zero to each axis for the new point, mapping onto itself.
  - We need to be able to take a component of some other quantity besides  $x_{\text{old}}$ ,  $y_{\text{old}}$ , and  $z_{\text{old}}$
  - Our vector we're multiplying needs to grow by 1 term.
  - Let's supply an extra constant. Use 1.
- 

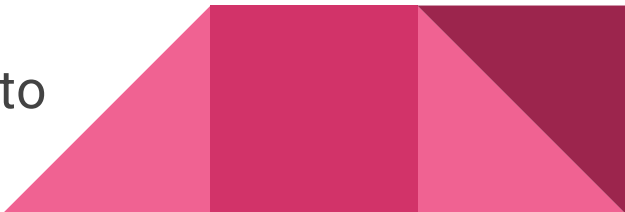


# Translation Matrices

Solution:

$$\begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \\ z_{old} \\ 1 \end{bmatrix} = \begin{bmatrix} x_{new} \\ y_{new} \\ z_{new} \\ 1 \end{bmatrix}$$

One of the components that contributes to the answer is now the constant 1, which we can pull from even when all others are zero. This allows us to move the picture.

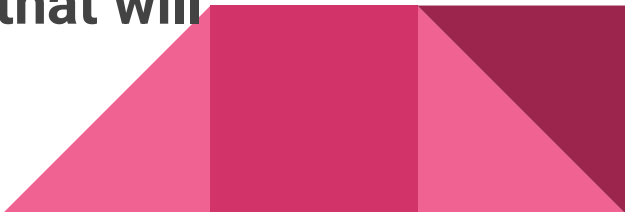


# Translation Matrices

$$\begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \\ z_{old} \\ 1 \end{bmatrix} = \begin{bmatrix} x_{new} \\ y_{new} \\ z_{new} \\ 1 \end{bmatrix}$$

We'd like our new vector to have “1” in the 4<sup>th</sup> coordinate as well, so we can use matrices on it right away.

**What does our matrix have to be like to ensure that will happen?**



# Translation Matrices

Solution:

$$\begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \\ z_{old} \\ 1 \end{bmatrix} = \begin{bmatrix} x_{new} \\ y_{new} \\ z_{new} \\ 1 \end{bmatrix}$$

This 4th row is an equation that literally says “ $1_{new}=1_{old}$ ”.



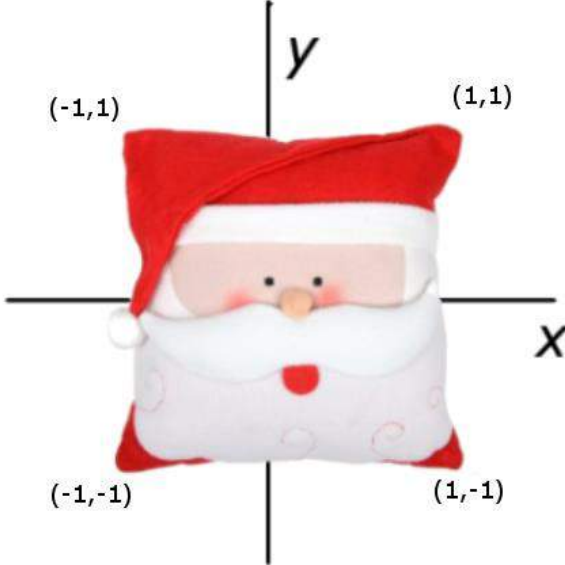
# Linear Vs. Affine Transformations

- A linear transformation only takes linear combinations of  $x$ ,  $y$ , and  $z$ .
  - Santa's nose stays on the origin forever.
- An *\*affine\** transformation is more general and more powerful.
  - It's called an affine transformation because it **preserves** affine combinations (line segment interpolations don't get messed up before vs. after).

# Translation Matrices

3D translations are the reason we use 4x4 matrices instead of 3x3.

Let's do one:

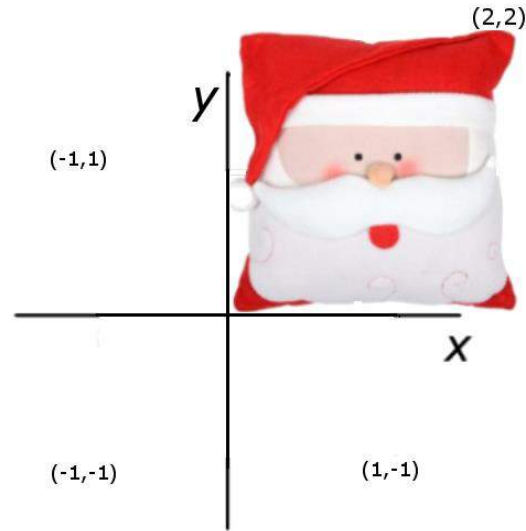
$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \\ z_{old} \\ 1 \end{bmatrix} = \begin{bmatrix} x_{new} \\ y_{new} \\ z_{new} \\ 1 \end{bmatrix}$$


# Translation Matrices

3D translations are the reason we use 4x4 matrices instead of 3x3.

Let's do one:

$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \\ z_{old} \\ 1 \end{bmatrix} = \begin{bmatrix} x_{new} \\ y_{new} \\ z_{new} \\ 1 \end{bmatrix}$$



# Summary: Affine is Useful

- Affine transformations are the main modeling tool in graphics
  - They are applied as matrix multiplications
  - Any affine transformation can be performed as a series of elementary affine transformations
  - We can now do object placement
    - Model entire scenes



# Affine Transformation in 3D

- Translation

$$\begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Rotate

$$\begin{pmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Scale

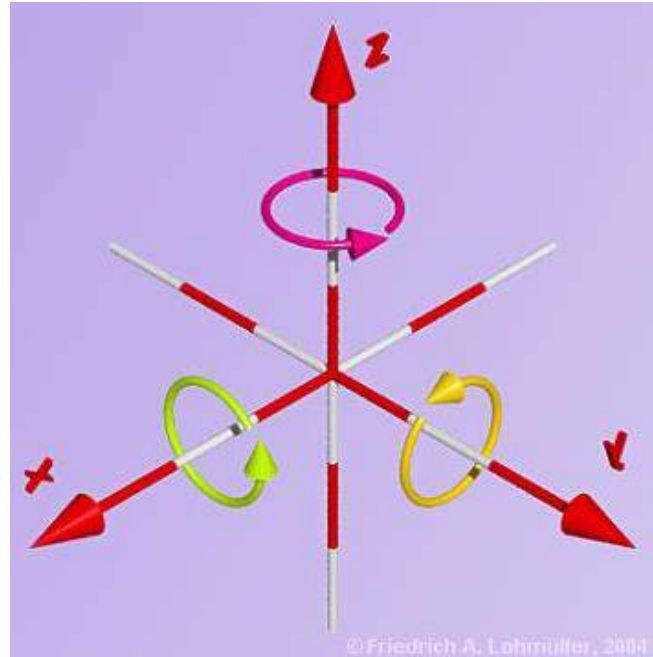
$$\begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Shear

$$\begin{pmatrix} 1 & 0 & SH_x & 0 \\ 0 & 1 & SH_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



# Three Axes to Rotate Around



# Rotation Matrices

## 3D: Rotation matrix

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{X Rotation}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{Y Rotation}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{Z Rotation}$$



# 3D Rotation

*Various representations possible*

*Decomposition into axis rotations*

- x-roll, y-roll, z-roll

*Counterclockwise positive angle assumption*

# Reminder: 2D Rotation

$$Q_x = \cos \theta P_x - \sin \theta P_y$$

$$Q_y = \sin \theta P_x + \cos \theta P_y$$

In matrix form:

$$\begin{bmatrix} Q_x \\ Q_y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix}$$

Or:

$$Q = \mathbf{R}(\theta)P$$

# Z-Roll

$$Q_x = \cos \theta P_x - \sin \theta P_y$$

$$Q_y = \sin \theta P_x + \cos \theta P_y$$

$$Q_z = P_z$$

$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# X-Roll

*Cyclic indexing*

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ z \\ x \\ y \end{bmatrix}$$

$$x \rightarrow y \rightarrow z \rightarrow x \rightarrow y$$

$$Q_y = \cos \theta P_y - \sin \theta P_z$$

$$Q_z = \sin \theta P_y + \cos \theta P_z$$

$$Q_x = P_x$$

$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Y-Roll

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ z \\ x \\ y \end{bmatrix}$$

$$Q_z = \cos \theta P_z - \sin \theta P_x$$

$$Q_x = \sin \theta P_z + \cos \theta P_x$$

$$Q_y = P_y$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Elementary 3D Affine Transformations

## *Translation*

$$\begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix}$$



# Scaling Around the Origin

$$\begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix}$$

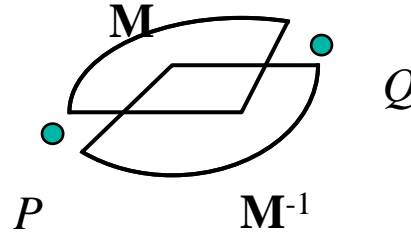
# Shear Around the Origin

*Along x-axis*

$$\begin{bmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & a & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix}$$

# Inverse of a Transformation

*Inverse transformation:  $Q = MP$ ,  $P = M^{-1}Q$*



*We can use Cramer's rule to invert  $M$ , or we can be smarter about it*

# Inverse of Translation

$$Q = T(t)P \rightarrow P = T(-t)Q$$

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix}$$

# Inverse of Scaling

$$Q = \mathcal{S}(s_x, s_y)P \rightarrow P = \mathcal{S}\left(\frac{1}{s_x}, \frac{1}{s_y}\right)Q$$

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

# Inverse of a Shear in $x$

$$Q = Sh_x(a)P \rightarrow P = Sh_x(-a)Q$$

$$\begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & -a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

# Inverse of Rotation

$$Q = \mathcal{R}(\theta)P \rightarrow P = \mathcal{R}(-\theta)Q$$

$$\begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

# Inversion of Transformations

***Translation:***  $\mathbf{T}^{-1}(t_x, t_y, t_z) = \mathbf{T}(-t_x, -t_y, -t_z)$

***Rotation:***  $\mathbf{R}_{axis}^{-1}(\theta) = \mathbf{R}_{axis}(-\theta)$

***Scaling:***  $\mathbf{S}^{-1}(s_x, s_y, s_z) = \mathbf{S}(1/s_x, 1/s_y, 1/s_z)$

***Shearing:***  $\mathbf{S}_h^{-1}(a) = \mathbf{S}_h(-a)$



# Inverse of Rotations

*Pure rotation only, no scaling or shear*

$$\mathbf{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}$$

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

*Since the rotation matrix  $\mathbf{M}$  is an orthonormal matrix*

# Composition of 3D Affine Transformations

*The composition of affine transformations is an affine transformation*

*Any 3D affine transformation can be performed as a series of elementary affine transformations*