CSE 167:
Introduction to Computer Graphics
Lecture #4: Linear Algebra

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2020

Announcements

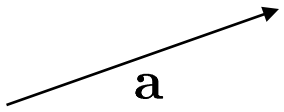
- Homework Project 1 due October 25
- Discussion Project 1: Wednesday 1pm

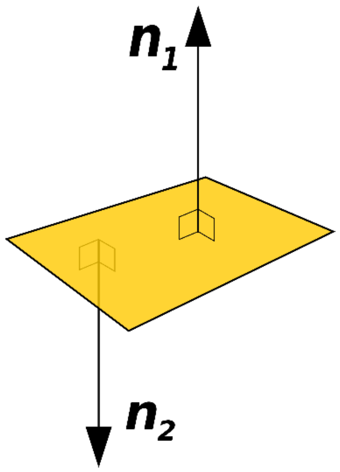
Overview

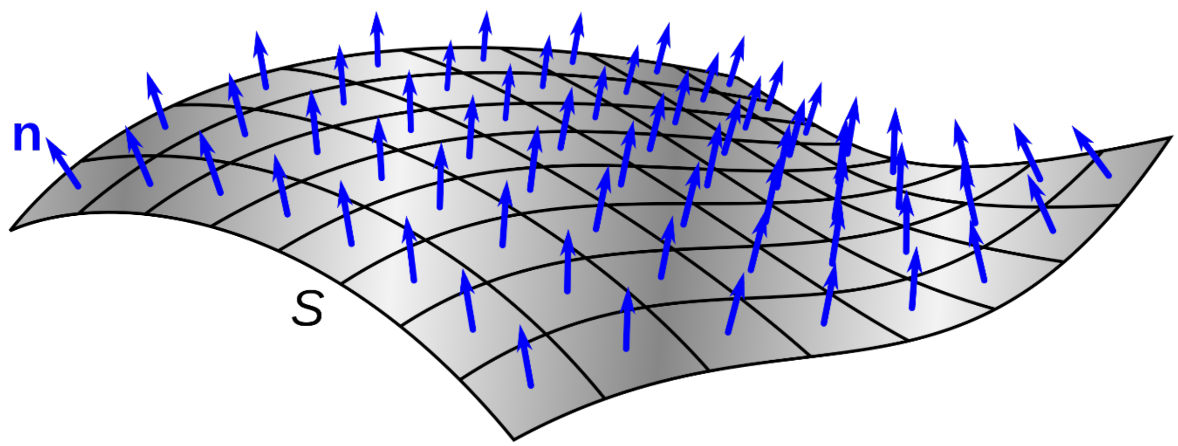
- Vectors and matrices
- Affine transformations
- Homogeneous coordinates

Vectors

- Give direction and length in 3D
- Vectors can describe
  - Difference between two 3D points
  - Speed of an object
  - Surface normals (vectors perpendicular to surfaces)







Vector arithmetic using coordinates

$$\mathbf{a} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix}$$

$$\mathbf{a} + \mathbf{b} = \begin{bmatrix} a_x + b_x \\ a_y + b_y \\ a_z + b_z \end{bmatrix} \qquad \mathbf{a} - \mathbf{b} = \begin{bmatrix} a_x - b_x \\ a_y - b_y \\ a_z - b_z \end{bmatrix}$$

$$-\mathbf{a} = \begin{bmatrix} -a_x \\ -a_y \\ -a_z \end{bmatrix} \qquad s\,\mathbf{a} = \begin{bmatrix} sa_x \\ sa_y \\ sa_z \end{bmatrix}$$

where s is a scalar

Vector Magnitude

- The magnitude (length) of a vector is:

$$|\mathbf{v}|^2 = v_x^2 + v_y^2 + v_z^2$$

$$|\mathbf{v}| = \sqrt{v_x^2 + v_y^2 + v_z^2}$$

- A vector with length of 1.0 is called *unit vector*
- We can also *normalize* a vector to make it a unit vector

$$\frac{\mathbf{v}}{|\mathbf{v}|}$$

- Unit vectors are often used as **surface normals**

Dot Product

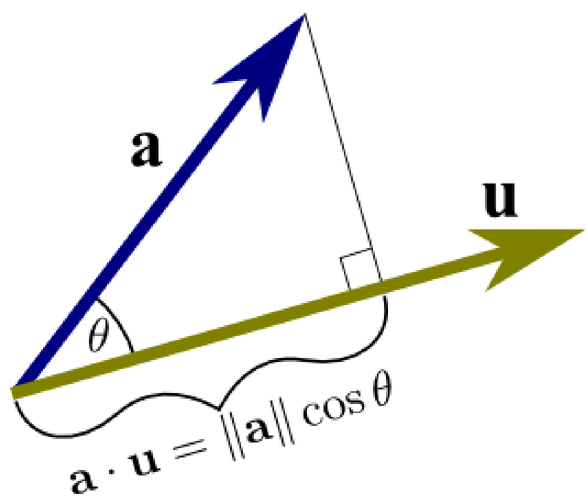
$$\mathbf{a} \cdot \mathbf{b} = \sum a_i b_i$$

$$\mathbf{a} \cdot \mathbf{b} = a_x b_x + a_y b_y + a_z b_z$$

$$\mathbf{a} \cdot \mathbf{b} = |a||b| \cos\theta$$

Dot Product with Unit Vector

- The dot product of **a** with unit vector **u**, denoted **a** ·**u**, is defined to be the projection of **a** in the direction of **u**, or the amount that **a** is pointing in the same direction as unit vector **u**.

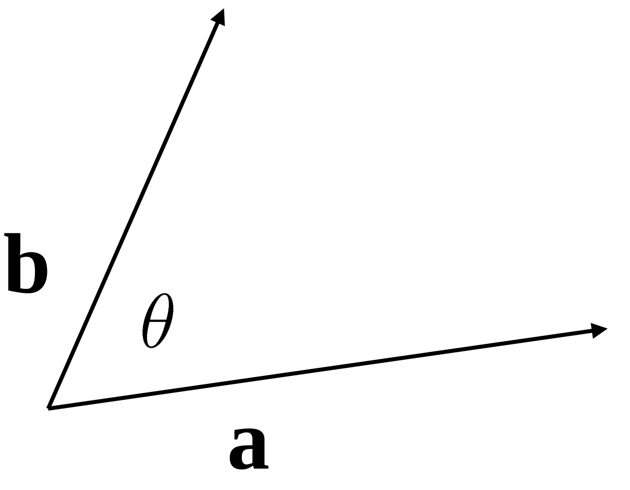


Angle Between Two Vectors

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}|\cos\theta$$

$$\cos\theta = \left(\frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}||\mathbf{b}|} \right)$$

$$\theta = \cos^{-1}\left(\frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}||\mathbf{b}|} \right)$$



Dot Product: Interpretation

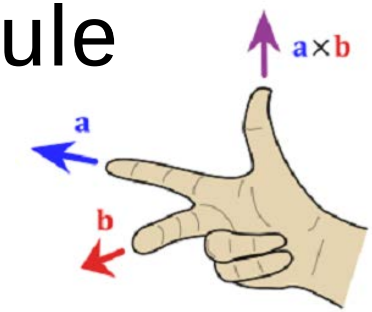
- If **a** and **b** are perpendicular, the result of the dot product will be zero.

- If the angle between **a** and **b** is less than 90 degrees, the dot product will be positive (greater than zero).

- If the angle between **a** and **b** is greater than 90 degrees, the dot product will be negative (less than zero)

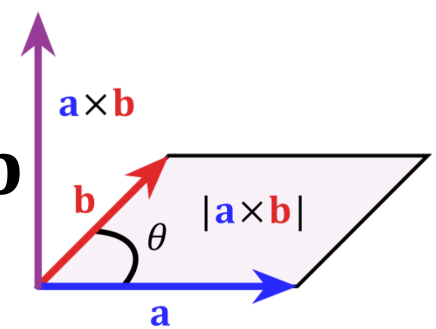
Cross Product

$\mathbf{a} \times \mathbf{b}$ is a vector *perpendicular* to both $\mathbf{a}$ and $\mathbf{b}$, in the direction defined by the right hand rule



$$\left|\mathbf{a} \times \mathbf{b}\right| = \left|\mathbf{a}\right|\left|\mathbf{b}\right| \sin\theta$$

$$\left|\mathbf{a} \times \mathbf{b}\right| = \text{area of parallelogram } \mathbf{ab}$$



$$\left|\mathbf{a} \times \mathbf{b}\right| = 0 \text{ if } \mathbf{a} \text{ and } \mathbf{b} \text{ are parallel}$$

(or one or both degenerate)

Cross Product

$$a \times b = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \times \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = \begin{bmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{bmatrix}$$

Cross Product Calculation

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \times \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = \begin{bmatrix} \boxed{a_y b_z - a_z b_y} \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{bmatrix}$$

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \times \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = \begin{bmatrix} a_y b_z - a_z b_y \\ \boxed{a_z b_x - a_x b_z} \\ a_x b_y - a_y b_x \end{bmatrix}$$

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \times \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = \begin{bmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ \boxed{a_x b_y - a_y b_x} \end{bmatrix}$$

Matrices

- Rectangular array of numbers

$$\mathbf{M} = \begin{bmatrix} m_{1,1} & m_{1,2} & \dots & m_{1,n} \\ m_{2,1} & m_{2,2} & \dots & m_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{m,1} & m_{2,2} & \dots & m_{m,n} \end{bmatrix} \in \mathbf{R}^{m \times n}$$

- Square matrix if **m = n**
- In graphics almost always: **m = n = 3; m = n = 4**

Matrix Addition

$$\mathbf{A} + \mathbf{B} = \begin{bmatrix} a_{1,1} + b_{1,1} & a_{1,2} + b_{1,2} & \dots & a_{1,n} + b_{1,n} \\ a_{2,1} + b_{2,1} & a_{2,2} + b_{2,2} & \dots & a_{2,n} + b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} + b_{m,1} & a_{2,2} + b_{2,2} & \dots & a_{m,n} + b_{m,n} \end{bmatrix}$$

$$\mathbf{A}, \mathbf{B} \in \mathbf{R}^{m \times n}$$

Multiplication With Scalar

$$s\mathbf{M} = \mathbf{M}s = \begin{bmatrix} sm_{1,1} & sm_{1,2} & \dots & sm_{1,n} \\ sm_{2,1} & sm_{2,2} & \dots & sm_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ sm_{m,1} & sm_{2,2} & \dots & sm_{m,n} \end{bmatrix}$$

Matrix Multiplication

$$\mathbf{AB} = \mathbf{C}, \quad \mathbf{A} \in \mathbf{R}^{p,q}, \mathbf{B} \in \mathbf{R}^{q,r}, \mathbf{C} \in \mathbf{R}^{p,r}$$

$$(\mathbf{AB})_{i,j} = \mathbf{C}_{i,j} = \sum_{k=1}^{q} a_{i,k} b_{k,j}, \quad i \in 1..p, j \in 1..r$$

Matrix-Vector Multiplication

$$\begin{bmatrix} \mathrm{a} & b \\ \mathrm{c} & d \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + c \\ dx + ey + f \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} ax + by + cz \\ dx + ey + fz \\ gx + hy + iz \end{bmatrix}$$

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + cz + d \\ ex + fy + gz + h \\ ix + jy + kz + l \\ 1 \end{bmatrix}$$

Identity Matrix

$$I_1 = [1],\ I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},\ I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},\ \cdots,\ I_n = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

$$\mathbf{MI} = \mathbf{IM} = \mathbf{M}, \quad \text{for any } \mathbf{M} \in \mathbf{R}^{n \times n}$$

Matrix Inverse

If a square matrix **M** is non-singular, there exists a unique inverse $\mathbf{M}^{-1}$ such that

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

$$(\mathbf{MPQ})^{-1} = \mathbf{Q}^{-1}\mathbf{P}^{-1}\mathbf{M}^{-1}$$

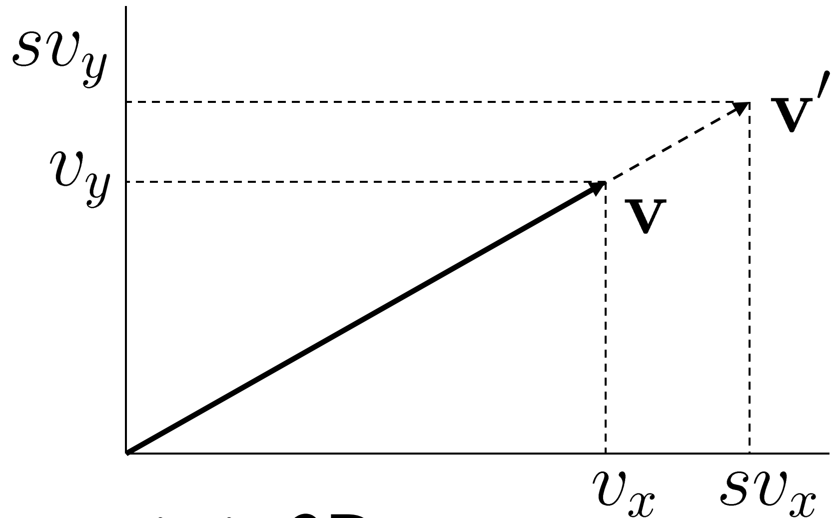
Overview

- Vectors and matrices
- Affine transformations
- Homogeneous coordinates

Affine Transformations

- Most important for graphics:
  - rotation, translation, scaling
- Wolfram MathWorld:
  - An **affine transformation** is any **transformation** that preserves collinearity (i.e., all points lying on a line initially still lie on a line after **transformation**) and ratios of distances (e.g., the midpoint of a line segment remains the midpoint after **transformation**).
- Implemented using matrix multiplications

Uniform Scale

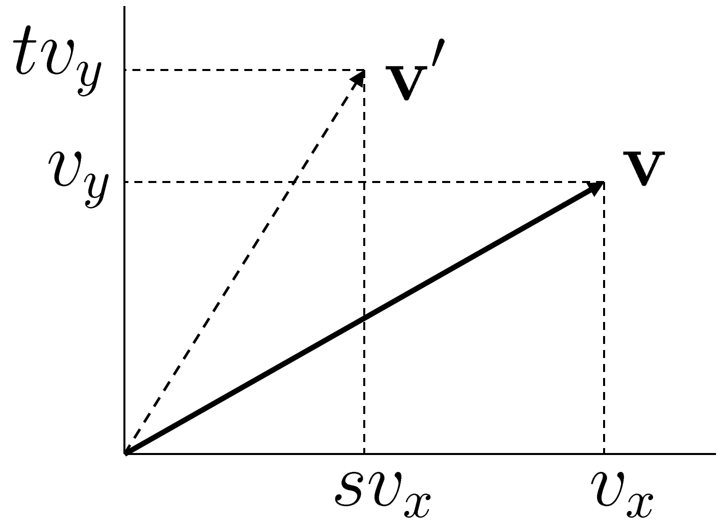


- Uniform scale matrix in 2D

$$\begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \mathbf{v} = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} v'_x \\ v'_y \end{bmatrix} = \mathbf{v}'$$

- Analogous in 3D: $\begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s \end{bmatrix}$

Non-Uniform Scale



- Nonuniform scaling matrix in 2D

$$\begin{bmatrix} s & 0 \\ 0 & t \end{bmatrix} \mathbf{v} = \begin{bmatrix} s & 0 \\ 0 & t \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} v'_x \\ v'_y \end{bmatrix} = \mathbf{v}'$$

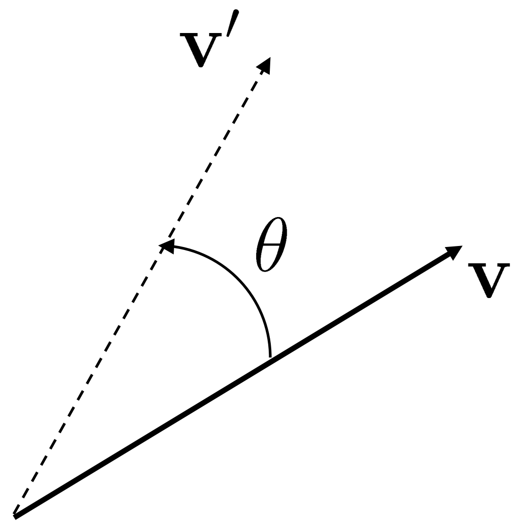
Non-Uniform Scale in 3D

- Scale in 2D: $$\begin{bmatrix} s & 0 \\ 0 & t \end{bmatrix}$$
- Analogous in 3D: $$\begin{bmatrix} s & 0 & 0 \\ 0 & t & 0 \\ 0 & 0 & u \end{bmatrix}$$

Rotation in 2D

- Convention: positive angle rotates counterclockwise
- Rotation matrix

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



$$\mathbf{v}' = \mathbf{R}(\theta)\mathbf{v}$$

Rotation in 3D

Rotation around coordinate axes

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

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

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

Rotation in 3D

- Concatenation of rotations around x, y, z axes

$$\mathbf{R}_{x,y,z}(\theta_x, \theta_y, \theta_z) = \mathbf{R}_x(\theta_x)\mathbf{R}_y(\theta_y)\mathbf{R}_z(\theta_z)$$

- $\theta_x, \theta_y, \theta_z$ are called Euler angles
- Result depends on matrix order!

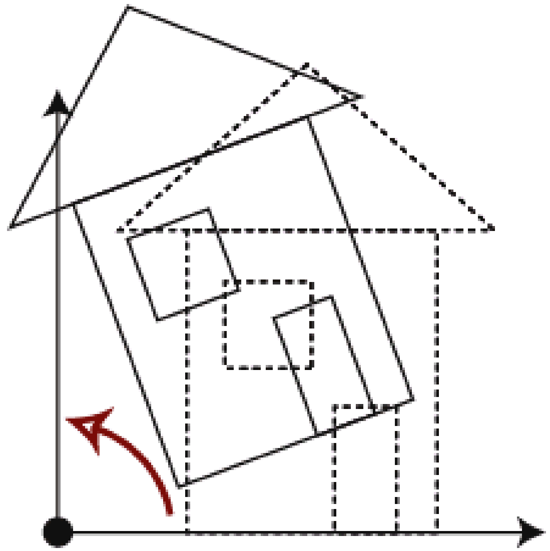
$$\mathbf{R}_x(\theta_x)\mathbf{R}_y(\theta_y)\mathbf{R}_z(\theta_z) \neq \mathbf{R}_z(\theta_z)\mathbf{R}_y(\theta_y)\mathbf{R}_x(\theta_x)$$

Rotation about an Arbitrary Axis

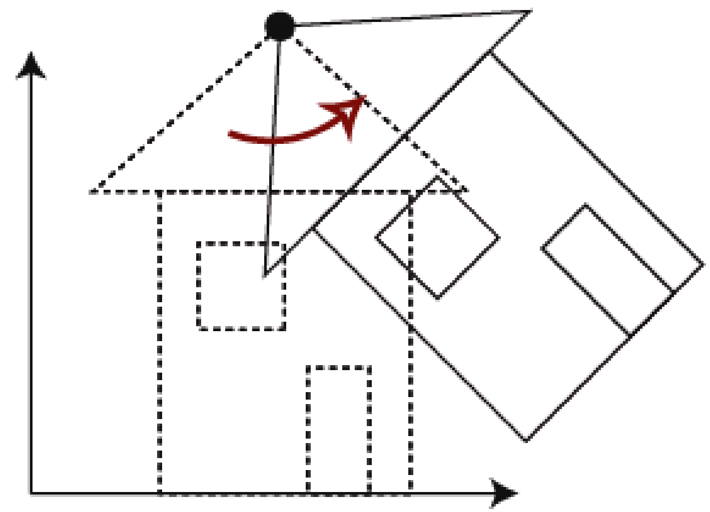
- Complicated!
- Rotate point [x,y,z] about axis [u,v,w] by angle θ:

$$\begin{bmatrix} \dfrac{u(ux+vy+wz)(1-\cos\theta)+(u^2+v^2+w^2)x\cos\theta+\sqrt{u^2+v^2+w^2}(-wy+vz)\sin\theta}{u^2+v^2+w^2} \\ \dfrac{v(ux+vy+wz)(1-\cos\theta)+(u^2+v^2+w^2)y\cos\theta+\sqrt{u^2+v^2+w^2}(wx-uz)\sin\theta}{u^2+v^2+w^2} \\ \dfrac{w(ux+vy+wz)(1-\cos\theta)+(u^2+v^2+w^2)z\cos\theta+\sqrt{u^2+v^2+w^2}(-vx+uy)\sin\theta}{u^2+v^2+w^2} \end{bmatrix}$$

How to rotate around a Pivot Point?

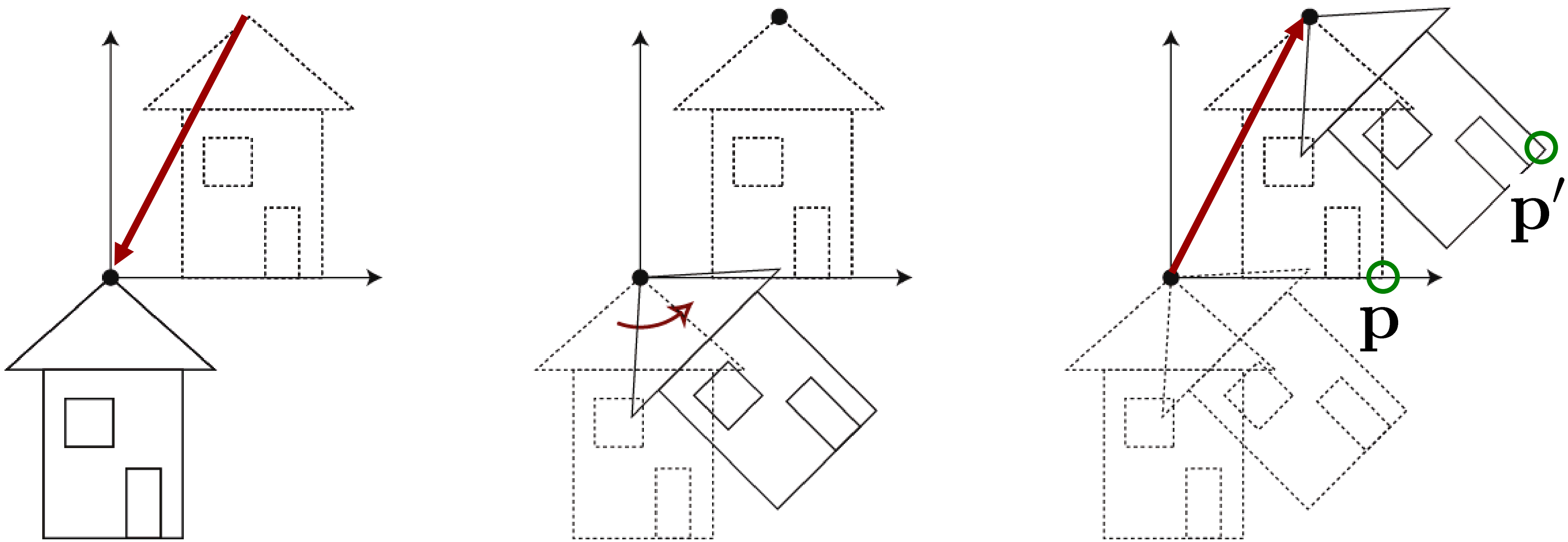


Rotation around origin:
p’ = **R p**



Rotation around pivot point:
p’ = ?

Rotating point p around a pivot point



1. Translation T
2. Rotation R
3. Translation T^{-1}

$$\mathbf{p}' = \mathbf{T}^{-1}\,\mathbf{R}\,\mathbf{T}\,\mathbf{p}$$

Concatenating transformations

- Given a sequence of transformations $\mathbf{M}_3\mathbf{M}_2\mathbf{M}_1$

$$\mathbf{p}' = \mathbf{M}_3\mathbf{M}_2\mathbf{M}_1\mathbf{p}$$

$$\mathbf{M}_{total} = \mathbf{M}_3\mathbf{M}_2\mathbf{M}_1$$

$$\mathbf{p}' = \mathbf{M}_{total}\mathbf{p}$$

- Note: associativity applies

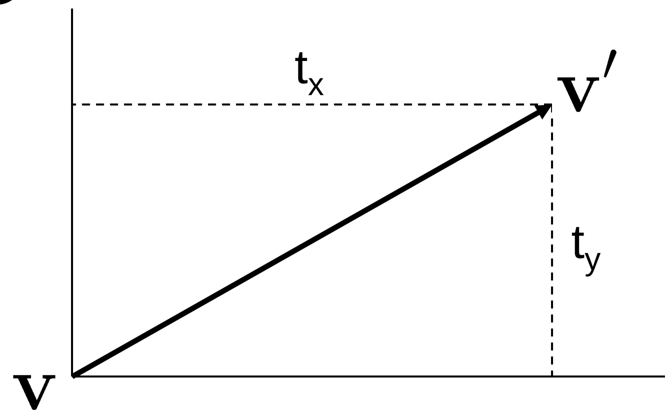
$$\mathbf{M}_{total} = (\mathbf{M}_3\mathbf{M}_2)\mathbf{M}_1 = \mathbf{M}_3(\mathbf{M}_2\mathbf{M}_1)$$

Overview

- Vectors and matrices
- Affine transformations
- Homogeneous coordinates

Translation

- Translation in 2D



- Translation matrix T=?

$$v' = \begin{bmatrix} v_x \\ v_y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = Tv = T\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix}\begin{bmatrix} v_x \\ v_y \end{bmatrix}$$

Translation

- Translation in 2D: 3x3 matrix

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Analogous in 3D: 4x4 matrix

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ \boldsymbol{z}' \\ \boldsymbol{w} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \boldsymbol{t}_x \\ 0 & 1 & 0 & \boldsymbol{t}_y \\ 0 & 0 & 1 & \boldsymbol{t}_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{z} \\ \boldsymbol{w} \end{bmatrix}$$

Homogeneous Coordinates

- Basic: a trick to unify/simplify computations.

- Deeper: projective geometry
  - Interesting mathematical properties
  - Good to know, but less immediately practical
  - We will use some aspect of this when we do perspective projection

Homogeneous Coordinates

- Allows us to unify affine transformation calculations.
- Add an extra component. 1 for a point, 0 for a vector:

$$\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \qquad \vec{\mathbf{v}} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix}$$

- Combine **M** and **d** into single 4x4 matrix:

$$\begin{bmatrix} m_{xx} & m_{xy} & m_{xz} & d_x \\ m_{yx} & m_{yy} & m_{yz} & d_y \\ m_{zx} & m_{zy} & m_{zz} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Let's see what happens when we multiply now…

Homogeneous Point Transform

- Transform a point:

$$\begin{bmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{bmatrix} = \begin{bmatrix} m_{xx} & m_{xy} & m_{xz} & d_x \\ m_{yx} & m_{yy} & m_{yz} & d_y \\ m_{zx} & m_{zy} & m_{zz} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} m_{xx}p_x + m_{xy}p_y + m_{xz}p_z + d_x \\ m_{yx}p_x + m_{yy}p_y + m_{yz}p_z + d_y \\ m_{zx}p_x + m_{zy}p_y + m_{zz}p_z + d_z \\ 0+0+0+1 \end{bmatrix}$$

$$\mathbf{M} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} + \vec{\mathbf{d}}$$

  - Top three rows are the affine transform!
  - Bottom row stays 1

Homogeneous Vector Transform

- Transform a vector:

$$\begin{bmatrix} v'_x \\ v'_y \\ v'_z \\ 0 \end{bmatrix} = \begin{bmatrix} m_{xx} & m_{xy} & m_{xz} & d_x \\ m_{yx} & m_{yy} & m_{yz} & d_y \\ m_{zx} & m_{zy} & m_{zz} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix} = \begin{bmatrix} m_{xx}v_x + m_{xy}v_y + m_{xz}v_z + 0 \\ m_{yx}v_x + m_{yy}v_y + m_{yz}v_z + 0 \\ m_{zx}v_x + m_{zy}v_y + m_{zz}v_z + 0 \\ 0 + 0 + 0 + 0 \end{bmatrix}$$

$$\mathbf{M} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

- Top three rows are the linear transform
  - Displacement **d** is properly ignored
- Bottom row stays 0

Homogeneous Arithmetic

- Correct operations always end in 0 or 1

$$\text{vector+vector:} \quad \begin{bmatrix} \vdots \\ 0 \end{bmatrix} + \begin{bmatrix} \vdots \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} \vdots \\ 0 \end{bmatrix}$$

$$\text{vector-vector:} \quad \begin{bmatrix} \vdots \\ 0 \end{bmatrix} - \begin{bmatrix} \vdots \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} \vdots \\ 0 \end{bmatrix}$$

$$\text{scalar*vector:} \quad s\begin{bmatrix} \vdots \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} \vdots \\ 0 \end{bmatrix}$$

$$\text{point+vector:} \quad \begin{bmatrix} \vdots \\ 1 \end{bmatrix} + \begin{bmatrix} \vdots \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} \vdots \\ 1 \end{bmatrix}$$

$$\text{point-point:} \quad \begin{bmatrix} \vdots \\ 1 \end{bmatrix} - \begin{bmatrix} \vdots \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} \vdots \\ 0 \end{bmatrix}$$

$$\text{point+point:} \quad \begin{bmatrix} \vdots \\ 1 \end{bmatrix} + \begin{bmatrix} \vdots \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} \vdots \\ 2 \end{bmatrix}$$

$$\text{scalar*point:} \quad s\begin{bmatrix} \vdots \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} \vdots \\ s \end{bmatrix}$$

$$\left\{ \begin{matrix} \text{weighted average} \\ \text{affine combination} \end{matrix} \right\} \text{of points:} \quad \frac{1}{3}\begin{bmatrix} \vdots \\ 1 \end{bmatrix} + \frac{2}{3}\begin{bmatrix} \vdots \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} \vdots \\ 1 \end{bmatrix}$$

Homogeneous Transforms

- Rotation, Scale, and Translation of points and vectors unified in a single matrix transformation:

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

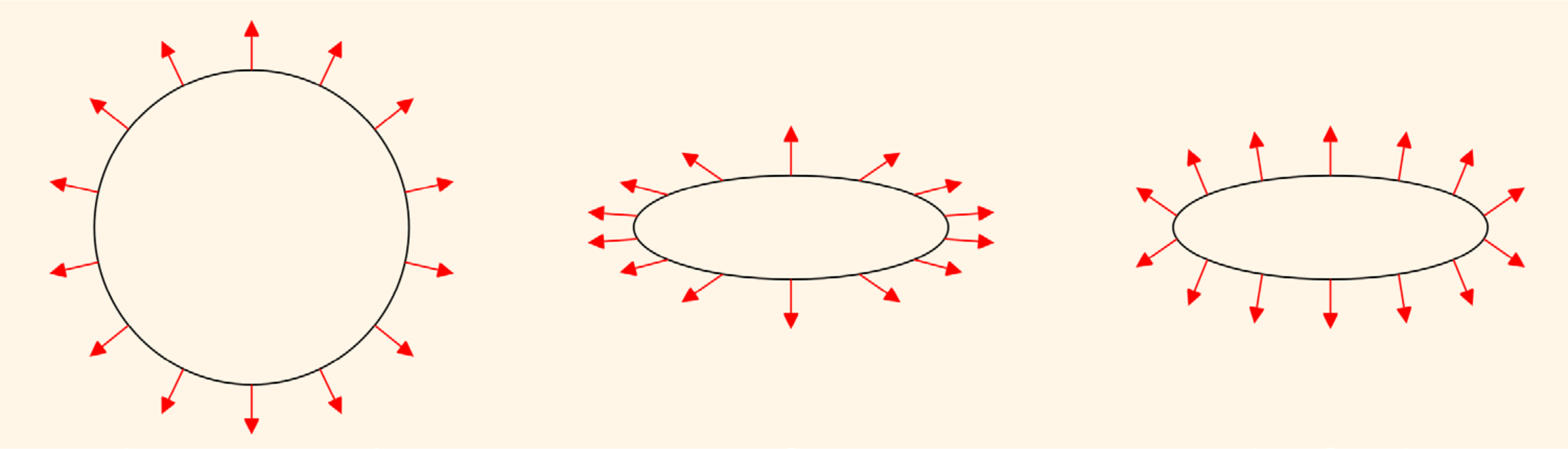
- Matrix has the form:
  - Last row always 0,0,0,1

$$\begin{bmatrix} m_{xx} & m_{xy} & m_{xz} & d_x \\ m_{yx} & m_{yy} & m_{yz} & d_y \\ m_{zx} & m_{zy} & m_{zz} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Transforms can be composed by matrix multiplication
  - Same caveat: order of operations is important
  - Same note: transforms operate right-to-left

Normal Transformation

- Why don't normal vectors always get transformed correctly with geometry?



- Middle image: normal scaled like geometry gives wrong result

  https://paroj.github.io/gltut/Illumination/Tut09%20Normal%20Transformation.html

4x4 Scale Matrix

- Generic form: $$\begin{bmatrix} s & 0 & 0 & 0 \\ 0 & t & 0 & 0 \\ 0 & 0 & u & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
- Inverse: $$\begin{bmatrix} \frac{1}{s} & 0 & 0 & 0 \\ 0 & \frac{1}{t} & 0 & 0 \\ 0 & 0 & \frac{1}{u} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4x4 Rotation Matrix

- Generic form:

$$\begin{bmatrix} r_1 & r_2 & r_3 & 0 \\ r_4 & r_5 & r_6 & 0 \\ r_7 & r_8 & r_9 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Inverse:

$$\begin{bmatrix} r_1 & r_4 & r_7 & 0 \\ r_2 & r_5 & r_8 & 0 \\ r_3 & r_6 & r_9 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4x4 Translation Matrix

- Generic form:

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

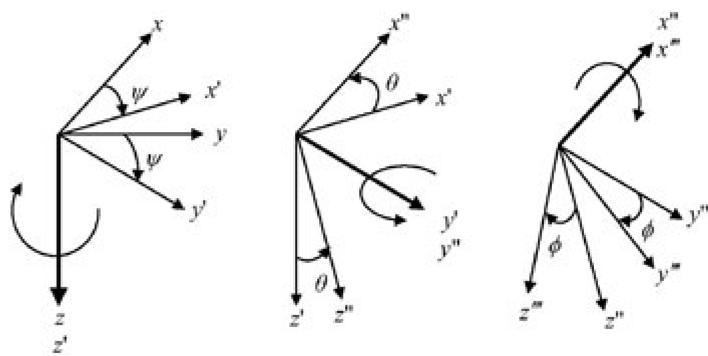
- Inverse:

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

Quaternions

Rotation Calculations

- Intuitive approach: Euler Angles
  - Simplest way to calculate rotations
  - Defines rotation by 3 sequential rotations about coordinate axes
- Example for rotation order Z-Y-X:



1[st], the body frame yaws right an angle ψ about the z-axis

2[nd], the body frame pitches up an angle θ about the y'-axis

3[rd], the body frame rolls CW an angle ϕ about the x''-axis

http://www.globalspec.com/reference/49379/203279/3-3-euler-angles

Problems With Euler Angles

- Problems with Euler angles:
  - No standard for order of rotations
  - Gimbal Lock, occurs in certain object orientations
    - Video: https://www.youtube.com/watch?v=rrUCBOlJdt4
- Better: rotation about arbitrary axis (no Gimbal lock)
  - Can be done with 4x4 matrix
- But: smoothly interpolating between two orientations is difficult
- → Quaternions

Quaternion Definition

- Given angle and axis of rotation:
  - a: rotation angle
  - {nx,ny,nz}: normalized rotation axis
- Calculation of quaternion coefficients w, x, y, z:
  - w = cos(a /2)
  - x = sin(a /2) * nx
  - y = sin(a /2) * ny
  - z = sin(a /2) * nz

Useful Quaternions

w	x	y	z	Description
1	0	0	0	Identity quaternion, no rotation
0	1	0	0	180° turn around X axis
0	0	1	0	180° turn around Y axis
0	0	0	1	180° turn around Z axis
sqrt(0.5)	sqrt(0.5)	0	0	90° rotation around X axis
sqrt(0.5)	0	sqrt(0.5)	0	90° rotation around Y axis
sqrt(0.5)	0	0	sqrt(0.5)	90° rotation around Z axis
sqrt(0.5)	-sqrt(0.5)	0	0	-90° rotation around X axis
sqrt(0.5)	0	-sqrt(0.5)	0	-90° rotation around Y axis
sqrt(0.5)	0	0	-sqrt(0.5)	-90° rotation around Z axis

Quaternions in GLM

- Create a quaternion for a 90 degree rotation about the y axis:
  - glm::quat rot = glm::angleAxis(glm::radians(90.f), glm::vec3(0.f, 1.f, 0.f));
- Cast the quaternion into a 4x4 matrix:
  - glm::mat4 rotate = glm::mat4_cast(rot);

Quaternions: Further Reading

- Rotating Objects Using Quaternions:
  - http://www.gamasutra.com/view/feature/131686/rotating_objects_using_quaternions.php
- Quaternions in GLM:
  - http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-17-quaternions/
- Quaternions in Unity 3D:
  - https://docs.unity3d.com/ScriptReference/Quaternion.html
- Quaternions in OpenSceneGraph :
  - http://www.openscenegraph.org/index.php/documentation/knowledge-base/40-quaternion-maths