Linear Algebra Review

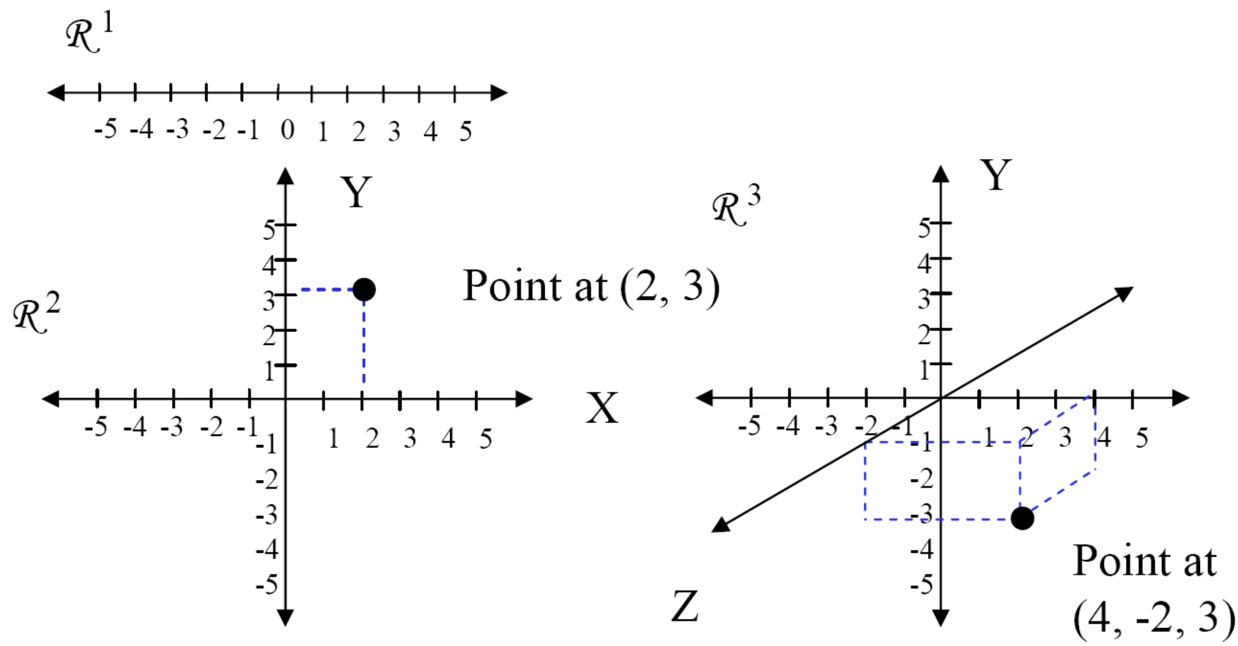
Credits: Brown University's
cs123 staff

What We Should Know About Linear Algebra

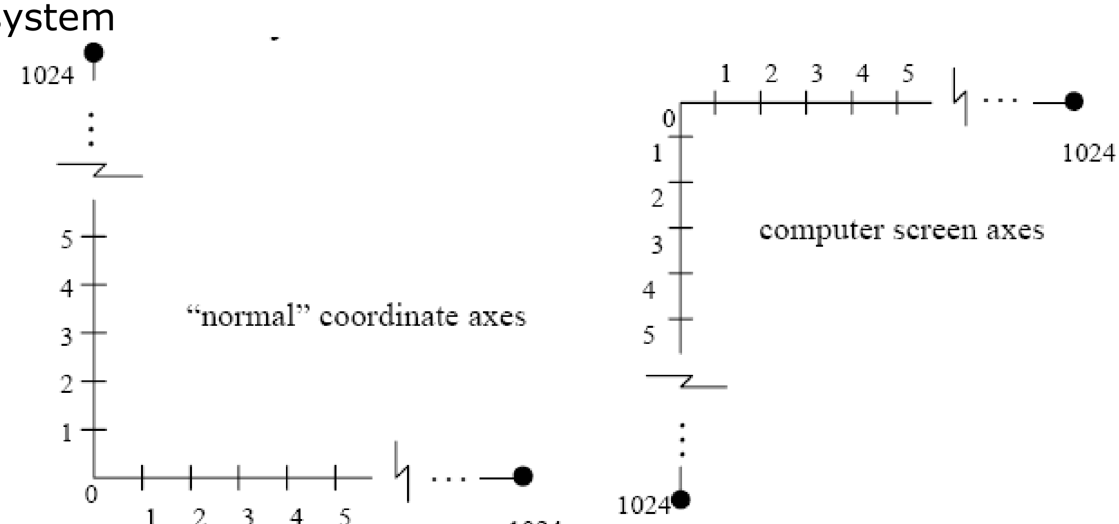
- 3D Coordinate geometry
- Vectors in 2 space and 3 space
- Dot product and cross product – definitions and uses
- Vector and matrix notation and algebra
- Properties (matrix associativity but **NOT** matrix commutativity)
- Matrix transpose and inverse – definition, use, and calculation
- Homogenous coordinates

Cartesian Coordinate Space

- Examples: one, two and three dimensional real coordinate spaces

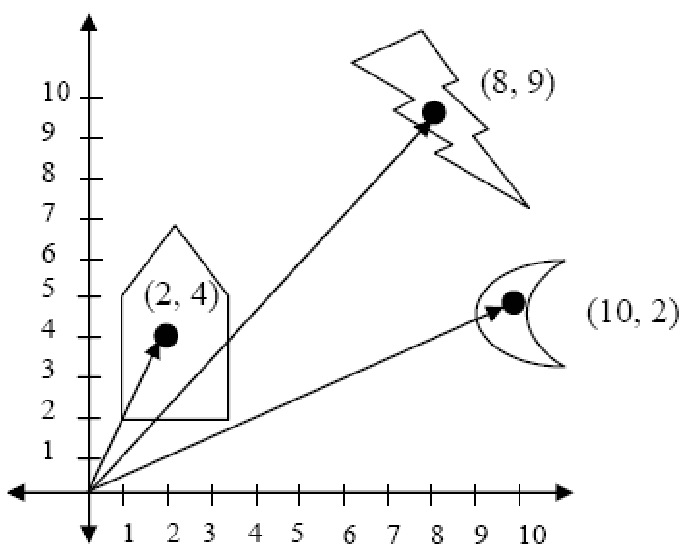


- Real numbers: between any two real numbers on an axis there exists another real number
- Compare with the computer screen, a positive integer coordinate system



Vectors & Vector Space (1/2)

- Consider all locations in relationship to one central reference point, called origin



- A vector tells us which direction to go with respect to the origin, and also the length of the trip
  - A vector does **not** specify where the trip begins, to continue this analogy

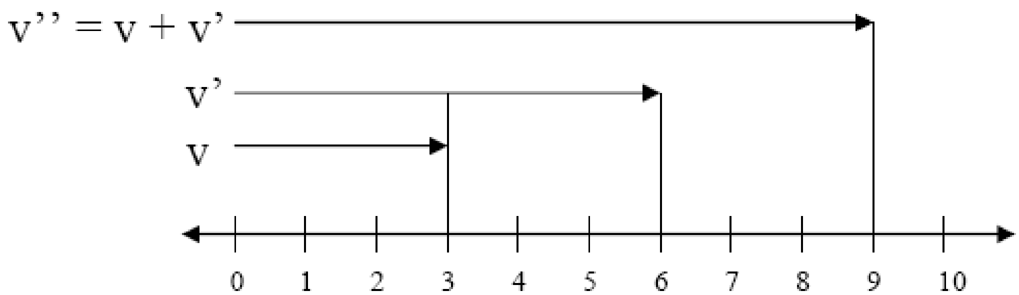
Vectors & Vector Space (2/2)

- Vectors are used extensively in computers to
  - represent positions of vertices of objects
  - determine orientation of a surface in space ("surface normal")
  - represent relative distances and orientations of lights, objects, and viewers in a 3D scene, so the rendering algorithms can create the impression of light interacting with solid and transparent objects (e.g., vectors from light sources to surfaces)

Vector Addition

Vector addition in $\Re^1$

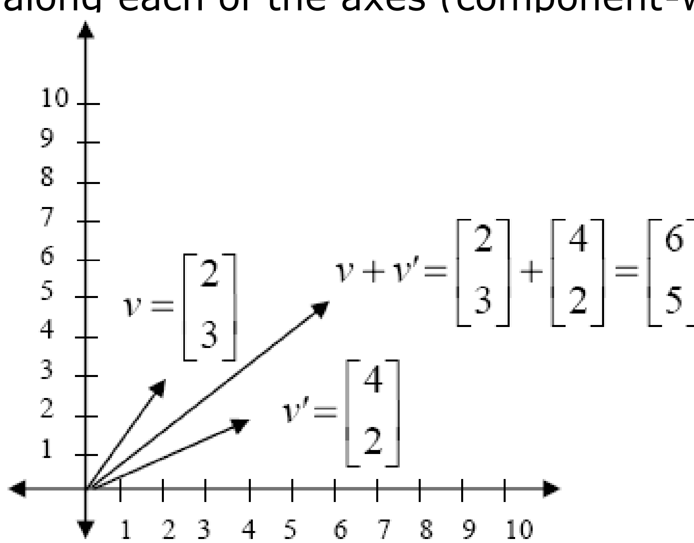
- Familiar addition of real numbers



v = [3], v′ = [6], v′′ = [9]

Vector addition in $\Re^2$

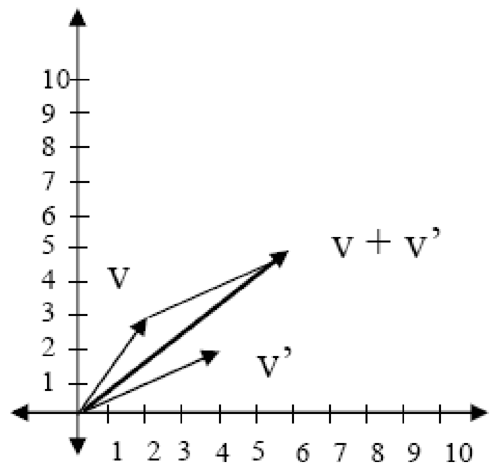
- The x and y parts of vectors can be added using addition of real numbers along each of the axes (component-wise addition)



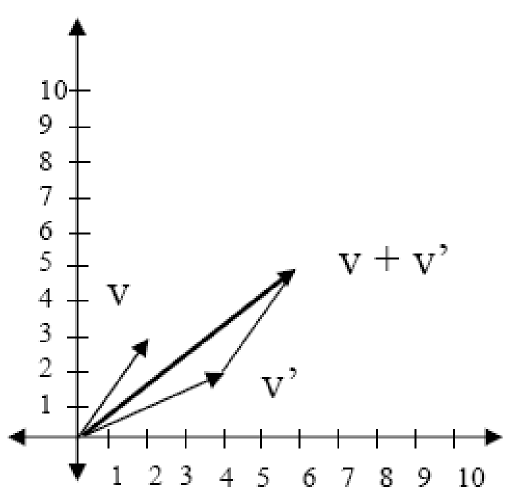
Result, v + v′, plotted in $\Re^2$ is the new vector

Adding Vectors Visually

- v’ added to v, using the parallelogram rule: take vector from the origin to v’; reposition it so that its tail is at the head of vector v; define v+v’ as the head of the new vector

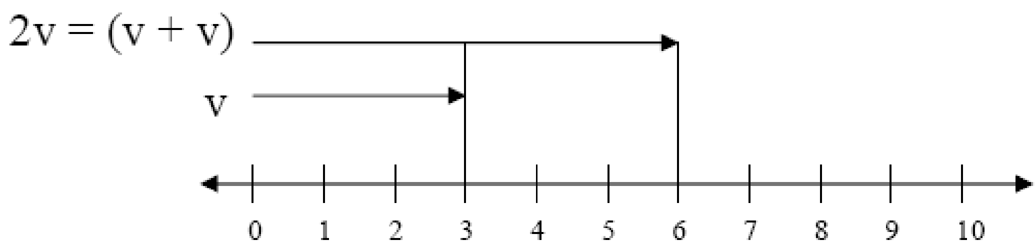


- or, equivalently, add v’ to v

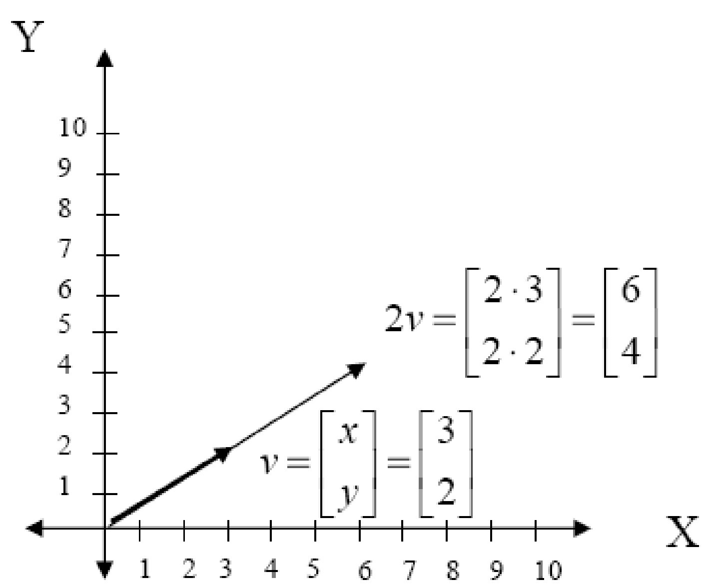


Scalar Multiplication (1/2)

On R1, familiar multiplication rules

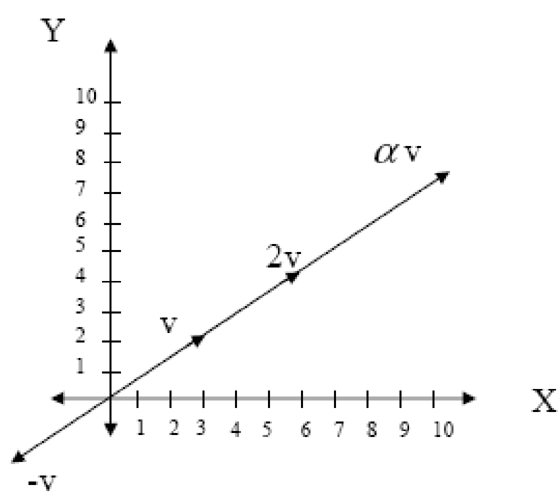


On R2 also



Scalar Multiplication (2/2) : Linear Dependence

- Set of all scalar multiples of a vector is a line through the origin



- Two vectors are linearly dependent when one is a multiple of the other

Basis Vectors of the Plane

- The unit vectors (i.e., whose length is one) on the x and y-axes are called the standard basis vectors of the plane
- The collection of all scalar multiples of $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ gives the first coordinate axis
- The collection of all scalar multiples of $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ gives the second coordinate axis
- Then any vector $\begin{bmatrix} x \\ y \end{bmatrix}$ can be expressed as the sum of

  scalar multiples of the unit vectors:

$$\begin{bmatrix} x \\ y \end{bmatrix} = x\begin{bmatrix} 1 \\ 0 \end{bmatrix} + y\begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

- We call these two vectors **basis** vectors for because any other vector can be expressed in terms of them
  - **This is an important concept. Make sure that you understand it.**

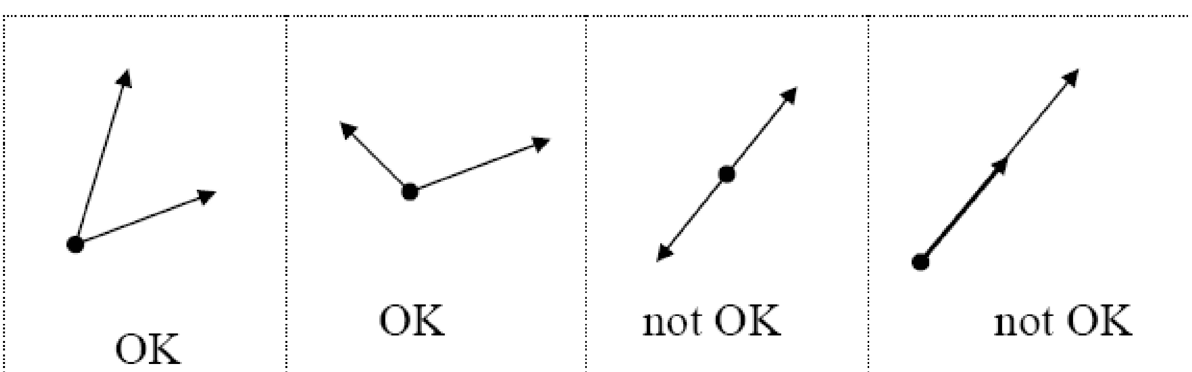
Non-orthogonal Basis Vectors

Question: Must the vectors be perpendicular in order to form a basis?

Answer: No.

$$\begin{bmatrix} n \\ m \end{bmatrix} = \alpha \begin{bmatrix} a \\ b \end{bmatrix} + \beta \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} \alpha a + \beta c \\ \alpha b + \beta d \end{bmatrix}$$

But: The vectors cannot be linearly dependent.



Rule for Dot Product

- Also known as scalar product, or inner product. The result is a scalar (i.e., a number, not a vector)

- Defined as the sum of the pairwise multiplications

- Example:

$$\begin{bmatrix} a & b & c & d \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = ax + by + cz + dw$$

- Note, the result is not a vector of the component-wise multiplications, it is a scalar value

Uses of the Dot Product

- Define length or magnitude of a vector
- Normalize vectors (generate vectors whose length is 1, called unit vectors)
- Measure angles between vectors
- Determine if two vectors are perpendicular
- Find the length of a vector projected onto a coordinate axis
- There are applets for the dot product under *Applets -> Linear Algebra*

Finding the Length of a Vector

- The dot product of a vector with itself, (v • v), is the square of the length of the vector

- We define the norm of a vector (i.e., its length) to be

$$\|v\| = \sqrt{v \bullet v}$$

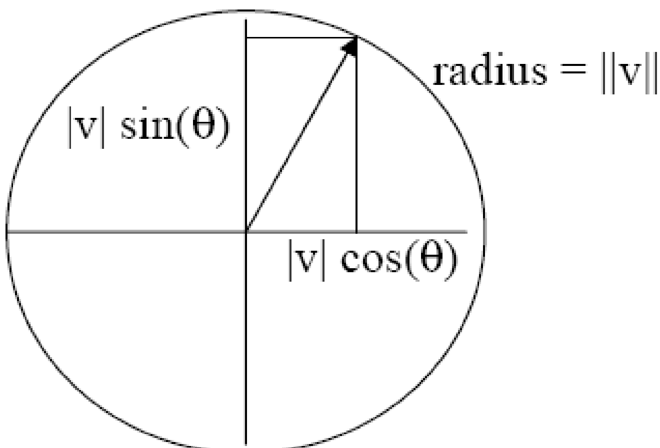
- V is called a unit vector if $\|v\| = \sqrt{v \bullet v}$ = 1

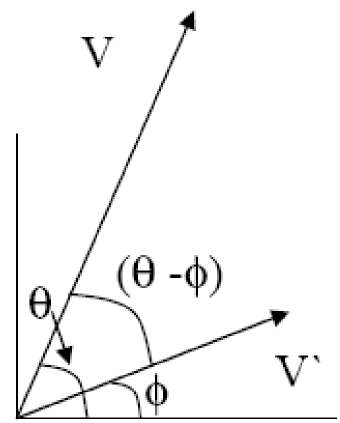
- To make an arbitrary vector v into a unit vector, i.e. to "normalize" it, divide by the length (norm) of v, which is denoted ||v||. Note that if v = 0, then its unit vector is undefined. So in general (with the 0 exception) we have

$$\hat{v} = \frac{1}{\|v\|} v$$

Finding the Angle Between Two Vectors

- The dot product of two non-zero vectors is the product of their lengths and the cosine of the angle between them: $v \cdot v' = \|v\|\|v'\|\cos(\theta - \phi)$





Proof:

$$v = [v_x v_y] = \|v\|[\cos\theta \sin\theta]$$
$$v' = [v'_x v'_y] = \|v'\|[\cos\phi \sin\phi]$$
$$v \bullet v' = \left(\|v\| \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \bullet \|v'\| \begin{bmatrix} \cos\phi \\ \sin\phi \end{bmatrix} \right) = \|v\|\|v'\| \left(\begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \bullet \begin{bmatrix} \cos\phi \\ \sin\phi \end{bmatrix} \right)$$
$$= \|v\|\|v'\|(\cos\theta\cos\phi + \sin\theta\sin\varphi)$$

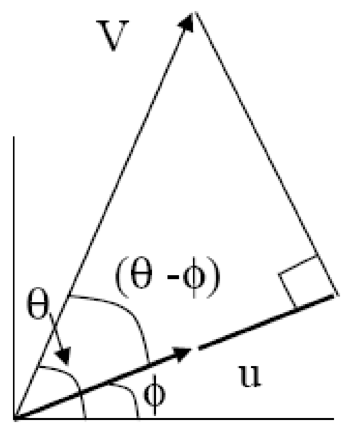
and, by a basic trigonometric identity

$$\cos\theta\cos\phi + \sin\theta\sin\phi = \cos(\theta - \phi)$$
$$\text{so, } v \bullet v' = \|v\|\|v'\|\cos(\theta - \phi)$$

More Uses of the Dot Product

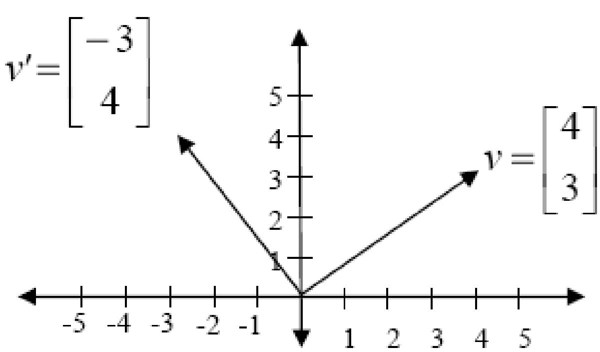
Finding the length of a projection

- If u is a unit vector, then $v \bullet u$ is the length of the projection of v onto the line containing u



Determining right angles

- Perpendicular vectors always have a dot product of 0 because the cosine of 90º is 0
- Example: $v = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$ and $v' = \begin{bmatrix} -3 \\ 4 \end{bmatrix}$



$$v \bullet v' = xx' + yy' = (4 \cdot -3) + (3 \cdot 4) = 0$$

$$v \bullet v' = \|v\| \|v'\| \cos\left(\frac{\pi}{2}\right) = \|v\| \|v'\| \bullet 0 = 0$$

Cross Product

- Cross product of $\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$ and $\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$, written $\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$ is

  defined as: $\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{bmatrix}$
- The resulting vector is perpendicular to both original vectors. That is, it is normal to the plane containing the two vectors.
- Its length is equal to the area of the parallelogram formed by the two vectors
- Thus, we can write:

  $\|v_1 \times v_2\| = \|v_1\| \|v_2\| \sin\theta$, where θ is the angle between v_1 and v_2

  Note that the cross-product does not generate a normalized vector (a vector of unit length).
- An easier way to represent the math for the cross product:
- $$\vec{a} \times \vec{b} = \det\left(\begin{bmatrix} \hat{i} & \hat{j} & \hat{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{bmatrix}\right)$$

  (This is for those who know how to get the determinant of a 3x3 matrix. i, j, k are the unit basis vectors.)
- Cross product follows right-hand rule, so switching order of vectors gives vector in opposite direction.

$$\vec{a} \times \vec{b} \neq \vec{b} \times \vec{a}, \text{ in fact } (\vec{a} \times \vec{b}) = -(\vec{b} \times \vec{a})$$

Computing the Surface Normals

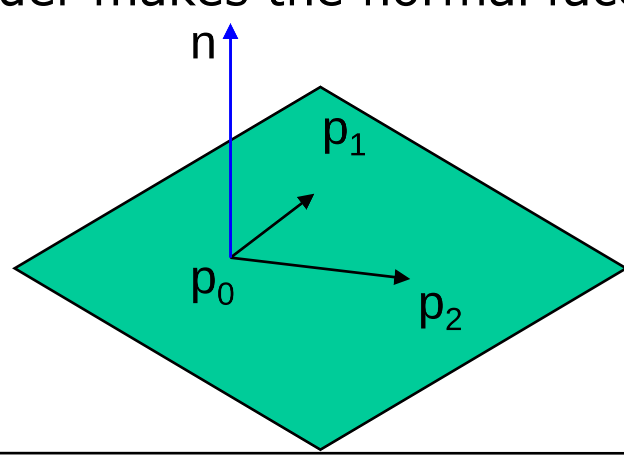
- All photorealistic lighting methods need to know normal to the polygon/face
- Each face/polygon has at least 3 vertices/points

- Three non-collinear points p_0, p_1, p_2 determine a plane
- The normal is $\perp$ to the plane: use cross-product

$$n = (p_2 - p_0) \times (p_1 - p_0)$$

(to get unit normal need to normalize)

- Counter-clockwise order makes the normal face outwards (good)



Computing the Vertex Normals

- Vertex normals are used in addition to surface normals to produce smooth shading

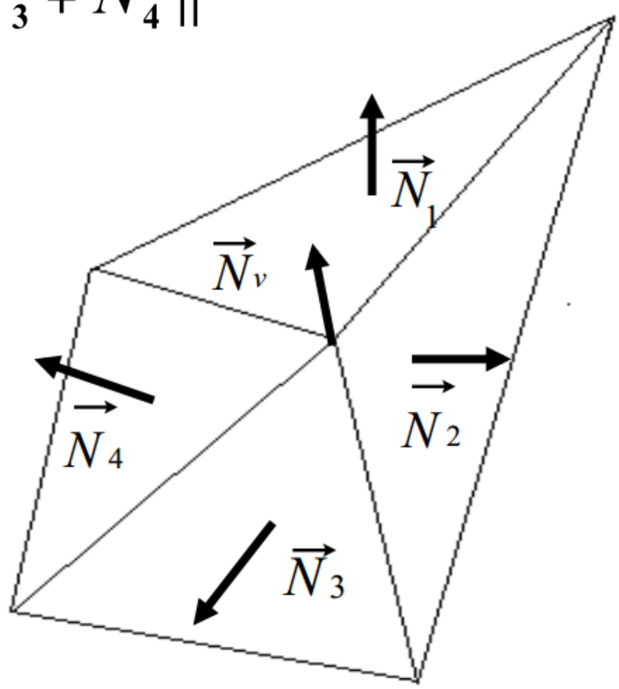
- They are calculated by averaging the surrounding polygons' normals:

$$\vec{N}_v = \frac{\vec{N}_1 + \vec{N}_2 + \vec{N}_3 + \vec{N}_4}{\| \vec{N}_1 + \vec{N}_2 + \vec{N}_3 + \vec{N}_4 \|}$$

- More generally:

$$\vec{N}_v = \frac{\sum_{i=1}^{n} \vec{N}_i}{\left\| \sum_{i=1}^{n} \vec{N}_i \right\|}$$

where n=3 or 4 usually



Algebraic Properties of Vectors

- Commutative (vector) $v + v' = v' + v$
- Associative (vector) $(v + v') + u = v + (v' + u)$
- Additive identity There is a vector 0, such that for all v, $(v + 0) = v = (0 + v)$
- Additive inverse For any v there is a vector –v such that v + (–v) = 0
- Distributive (vector) $r(v + v') = rv + rv'$
- Distributive (scalar) $(r + s)v = rv + sv$
- Associative (scalar) $r(sv) = (rs)v$
- Multiplicative identity For any v, $1 \in \Re$, $1 \cdot v = v$

Digression: Types of Transformations

- Projective $\supset$ affine $\supset$ linear
- Linear: preserves parallel lines, acts on a line to yield either another line or point. The vector [0, 0] is always transformed to [0, 0]. Examples: Scale and Rotate.
- Affine: preserves parallel lines, acts on a line to yield either another line or point. The vector [0, 0] is <u>not</u> always transformed to [0, 0]. Examples: Translate, as well as Rotate, and Scale (since all linear transformations are also affine).
- Projective: parallel lines <u>not necessarily</u> preserved, but acts on a line to yield either another line or point (not curves). Examples: you'll see a transformation in our camera model which is not Linear or Affine, but is projective. Translate, Rotate, and Scale are all Projective, since this is even more general than Affine.
- All of these transformations send lines to lines, therefore we only need to store endpoints
  - Note that we're just talking about transformations here, and not matrices specifically (yet)

Matrix-Vector Multiplication (1/2)

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

$$\text{where } M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, v = \begin{bmatrix} x \\ y \end{bmatrix}, v' = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

- The new vector is the dot product of each row of the matrix with the original column vector. Thus, the kth entry of the transformed vector is the dot product of the kth row of the matrix with the original vector.
- Example: scaling the vector $\begin{bmatrix} 3 \\ 6 \end{bmatrix}$ by 7 in the x direction

  and 0.5 in the y direction

$$\begin{bmatrix} 7 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 3 \\ 6 \end{bmatrix} = \begin{bmatrix} (7 \cdot 3) + (0 \cdot 6) \\ (0 \cdot 3) + (0.5 \cdot 6) \end{bmatrix} = \begin{bmatrix} 21 + 0 \\ 0 + 3 \end{bmatrix} = \begin{bmatrix} 21 \\ 3 \end{bmatrix}$$

Matrix-Vector Multiplication (2/2)

- In general:

$$\begin{bmatrix} a_1 & a_2 & a_3 & \cdots & a_n \\ b_1 & b_2 & b_3 & \cdots & b_n \\ c_1 & c_2 & c_3 & \cdots & c_n \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ m_1 & m_2 & m_3 & \cdots & m_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_n \end{bmatrix} = \begin{bmatrix} (a_1x_1)+(a_2x_2)+(a_3x_3)+\cdots+(a_nx_n) \\ (b_1x_1)+(b_2x_2)+(b_3x_3)+\cdots+(b_nx_n) \\ (c_1x_1)+(c_2x_2)+(c_3x_3)+\cdots+(c_nx_n) \\ \cdots \\ (m_1x_1)+(m_2x_2)+(m_3x_3)+\cdots+(m_nx_n) \end{bmatrix}$$

- Can also be expressed as:

$$MX = \begin{bmatrix} \sum_{i=1}^{n} a_i x_i \\ \sum_{i=1}^{n} a_i x_i \\ \cdots \\ \sum_{i=1}^{n} m_i x_i \end{bmatrix} = \begin{bmatrix} a \bullet x \\ b \bullet x \\ \cdots \\ m \bullet x \end{bmatrix}$$

Matrix-Matrix Multiplication

- Generally, how do we compute the product MN from matrices M and N?
- One way to think of matrix multiplication is in terms of row and column vectors: MN_{ij} = the dot product of the i^{th} row of M and the j^{th} column of N
- It is important to note that the rows of M and the columns of N must be the same size in order to compute their dot product
- So for M, an *m x n* matrix, and N, an *n x k* matrix:

$$MN = \begin{bmatrix} row_{M,1} \bullet col_{N,1} & \cdots & \cdots & row_{M,m} \bullet col_{N,k} \\ \vdots & row_{M,i} \bullet col_{N,j} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ row_{M,m} \bullet col_{N,1} & \cdots & \cdots & row_{M,m} \bullet col_{N,k} \end{bmatrix}$$

- Where rowA,x means the xth row of A, and similarly, colA,x means the xth column of A

Correct or Incorrect?

Can we multiply (mind the order):

- A 4x4 matrix with a 4x1 matrix (i.e., column vector)?

- A 4x4 matrix with a 1x4 matrix (i.e., row vector)?

- A 1x4 matrix (i.e., row vector) with a 4x4 matrix?

- A 4x1 matrix (i.e., column vector) with a 4x4 matrix?

Algebraic Properties of Matrices

Properties of matrix addition

- Commutative — $A+B=B+A$
- Associative — $(A+B)+C=A+(B+C)$
- Identity — There is a matrix 0, such that, for all A, $(A+0)=A=(0+A)$
- Inverse — For any A, there is a –A such that $A+(-A)=0$
- Distributive (matrix) — $r(A+B)=rA+rB$
- Distributive (scalar) — $(r+s)A=rA+sA$

Properties of matrix multiplication

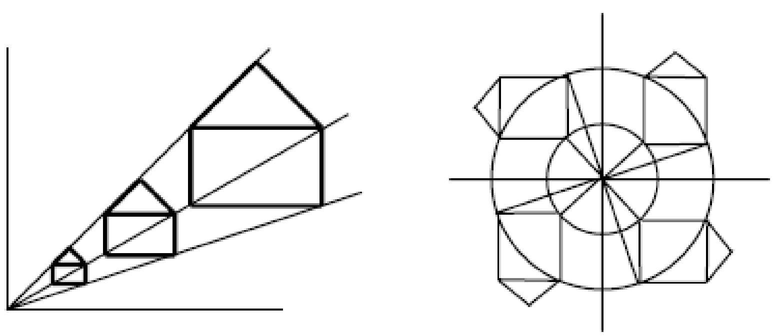
- NOT commutative — $AB \neq BA$
- Associative (matrix) — $(AB)C=A(BC)$
- Associative (scalar) — $(rs)A=r(sA)$
- Distributive (vector) — $A(v+v')=Av+Av'$
- Identity — There is a matrix I, such that, for all A, $AI=A=IA$
- Inverse (more later) — For some A, there is a matrix A^{-1} such that: $AA^{-1}=I=A^{-1}A$

Combining Transformations

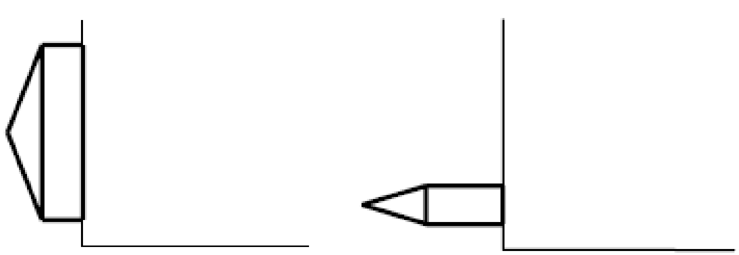
Matrices representing transformations performed in a sequence can be composed into a single matrix

- However, there are problems with combining matrices
  - Matrix multiplication is not commutative.

- Rotating or scaling object not centered at the origin introduces unwanted translation



Translation induced by scaling and rotation



Rotation followed by non-uniform scale; next same scale followed by same rotation

Commutative and Non-Commutative Combinations of Transformations – Be Careful!

In 2D

- Commutative
  - Translate, translate
  - Scale, scale
  - Rotate, rotate
  - Scale uniformly, rotate
- Non-commutative
  - Non-uniform scale, rotate
  - Translate, scale
  - Rotate, translate

In 3D

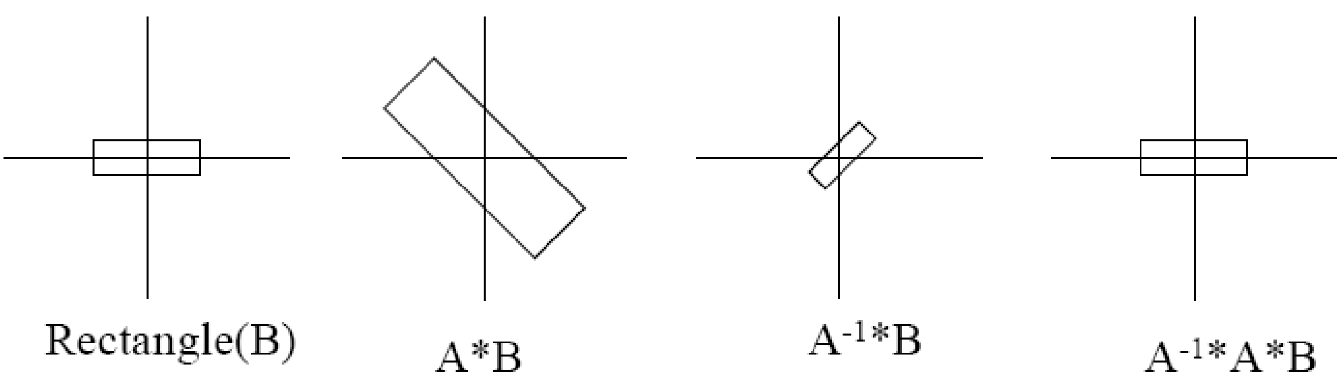
- Commutative
  - Translate, translate
  - Scale, scale
  - Scale uniformly, rotate
- Non-commutative
  - Non-uniform scale, rotate
  - Translate, scale
  - Rotate, translate
  - **Rotate, rotate**

What Does an Inverse Do?

- The inverse A^{-1} of transformation A will "undo" the result of transforming by A

- For example: if A scales by a factor of two and rotates 135^o, then A^{-1} will rotate by -135^o and then scale by one half

- Computing the inverse of a matrix: *Elementary Linear Algebra, by Howard Anton,* which can be found at the library



Correct or Incorrect?

$$\begin{bmatrix} 1 & 0 & 7 \\ 0 & 1 & 9 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 7 \\ 0 & 1 & 9 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 7 \\ 0 & 1 & 9 \\ 0 & 0 & 1 \end{bmatrix}$$

Addendum – Matrix Notation

- The application of matrices in the row vector notation is executed in the reverse order of applications in the column vector notation:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \leftrightarrow \begin{bmatrix} x & y & z \end{bmatrix}$$

- Column format: vector follows transformation matrix.

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

- Row format: vector precedes matrix and is pos-multiplied by it.

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

- By convention, we always use column vectors.

But, There's a Problem…

- Notice that

$$\begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} ax+dy+gz & bx+ey+hz & cx+fy+iz \end{bmatrix}$$

- while

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

Solution to Notational Problem

- In order for both types of notations to yield the same result, a matrix in the row system must be the transpose of the matrix in the column system
- Transpose is defined such that each entry at (i,j) in M, is mapped to (j,i) in its transpose (which is denoted M^T). You can visualize M^T as rotating M around its main diagonal

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

- Again, the two types of notation are equivalent:

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

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

- Different texts and graphics packages use different notations. Be careful!

Matrix Notation and Composition

- Application of matrices in row-major notation is reverse of application in column-major notation:

$$TRSv$$

$$\xleftarrow[\text{Matrices applied right to left}]{\hspace{6cm}}$$

$$vS^TR^TT^T$$

$$\xrightarrow[\text{Matrices applied left to right}]{\hspace{6cm}}$$

- In cs1566 we use the column-major notation

Are OpenGL Matrices Column-major or Row-major?

- For programming purposes, OpenGL matrices are 16-value arrays with base vectors laid out contiguously in memory. The translation components occupy the 13th, 14th, and 15th elements of the 16-element matrix.
- Column-major versus row-major is purely a notational convention. Note that post-multiplying with column-major matrices produces the same result as pre-multiplying with row-major matrices.
- The OpenGL Specification and the OpenGL Reference Manual both use column-major notation. One can use any notation, as long as it's clearly stated.