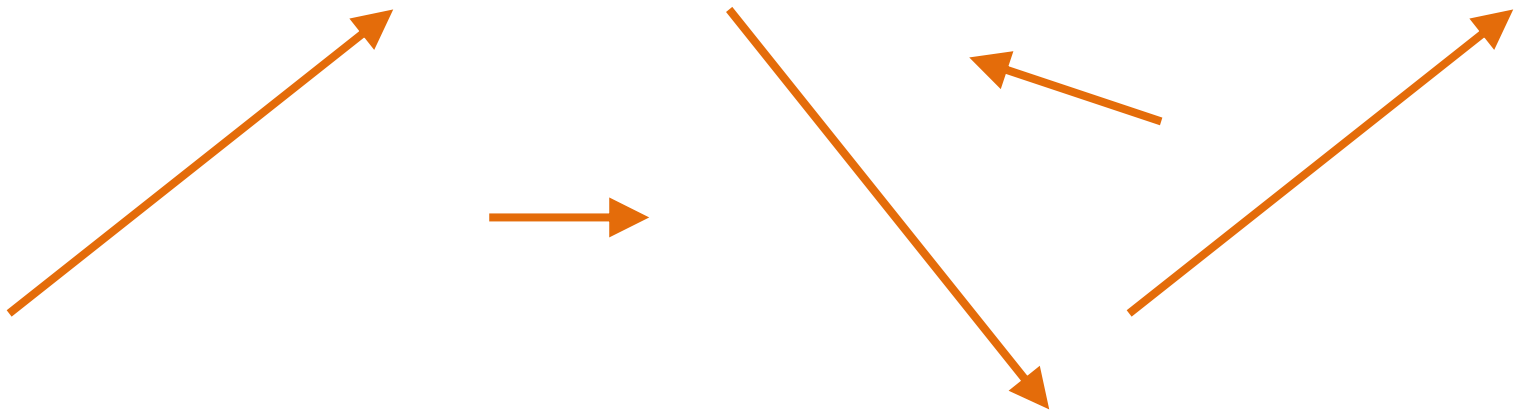
Vectors & Matrices

Marq Singer (marq@essentialmath.com)

What Is a Vector?

- Geometric object with two properties
  - direction
  - length (if length is 1, is *unit vector*)
- Graphically represented by



Algebraic Vectors

- Any entity that meets certain rules (lies in *vector space*) can be called ‘vector’
- Ex: Matrices, quaternions, fixed length polynomials
- Mostly mean geometric vectors, however

Vector Space

- Set of vectors related by +,·
- Meet rules
  - $\mathbf{v + w = w + v}$ **(commutative +)**
  - $\mathbf{(v + w) + u = v + (w + u)}$ **(associative +)**
  - $\mathbf{v + 0 = v}$ **(identity +)**
  - $\mathbf{v + (-v) = 0}$ **(inverse +)**
  - $\mathbf{(\alpha\beta)\, v = \alpha\, (\beta v)}$ **(associative ·)**
  - $\mathbf{(\alpha+\beta)v = \alpha v + \beta v}$ **(distributive ·)**
  - $\mathbf{\alpha(v + w) = \alpha v + \alpha w}$ **(distributive ·)**

Real Vector Spaces

- Usually only work in these
- $\boldsymbol{R}^n$ is an *n*-dimensional system of real numbers
  - Represented as ordered list of real numbers $(a_1,\ldots,a_n)$
- $\boldsymbol{R}^3$ is the 3D world, $\boldsymbol{R}^2$ is the 2D world

Linear Combination

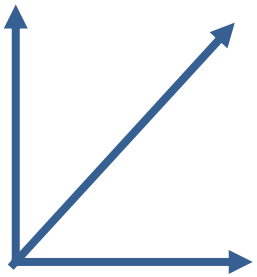
- Combine set of *n* vectors using addition and scalar multiplication
  - $\mathbf{v} = \alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2 + \ldots + \alpha_n\mathbf{v}_n$
- Collection of all possible linear combinations for given $\mathbf{v}_1 \ldots \mathbf{v}_n$ is called a *span*
- Linear combination of 2 perpendicular vectors span a plane

Linear Dependence

- A *system* of vectors $\mathbf{v_1}, \ldots, \mathbf{v_n}$ is called *linearly dependant* if for at least one $\mathbf{v_i}$
  - $\mathbf{v_i} = \alpha_1\mathbf{v_1} + \ldots + \alpha_{i-1}\mathbf{v_{i-1}} + \alpha_{i+1}\mathbf{v_{i+1}} + \ldots + \alpha_n\mathbf{v_n}$
- Otherwise, *linearly independent*
- Two linearly dependant vectors are said to be *collinear*
  - I.e. $\mathbf{w} = \alpha \cdot \mathbf{v}$
  - I.e. they point the “same” direction

Linear Dependence

- Example



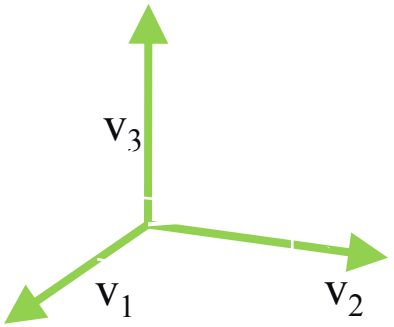
- Center vector can be constructed from outer vectors

Vector Basis

- Ordered set of *n* lin. ind. vectors
  - $\beta = \{ \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n \}$
- Span *n*-dimensional space
- Represent any vector as linear combo
  - $\mathbf{v} = \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \ldots + \alpha_n \mathbf{v}_n$
- Or just components
  - $\mathbf{v} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$

Vector Representation

- 3D vector $\mathbf{v}$ represented by (x, y, z)
  - Use standard basis $\{ \mathbf{i}, \mathbf{j}, \mathbf{k} \}$
  - Unit length, perpendicular (orthonormal)
  - $\mathbf{v} = x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$
- Number of units in each axis direction



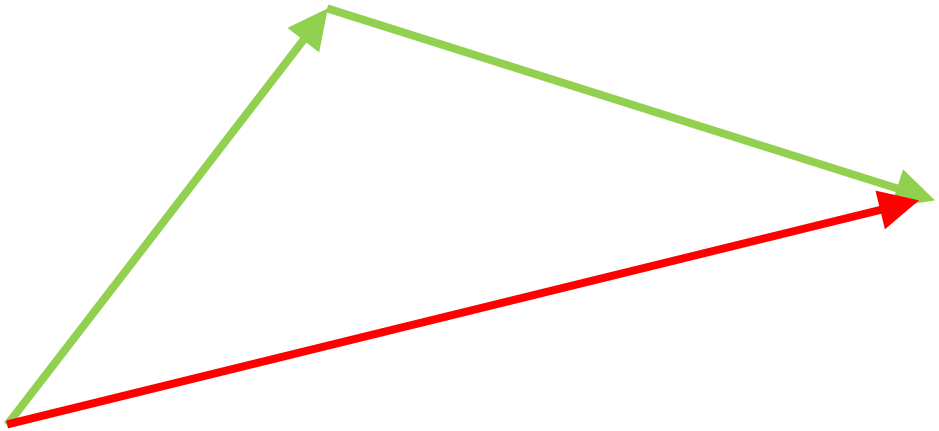
Vector Operations

- Addition: +,-
- Scale: $\cdot$
- Length: $||\mathbf{v}||$
- Normalize: $\hat{\mathbf{v}}$

Addition

- Add **a** to **b**

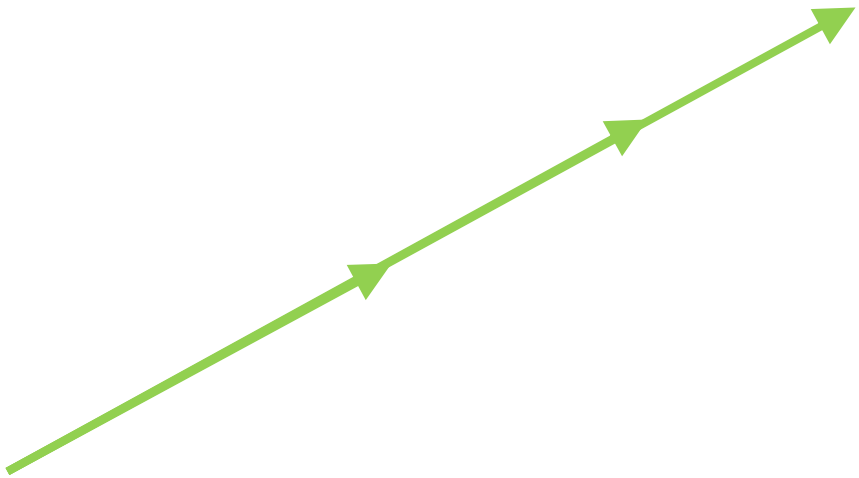
$$\mathbf{a}+\mathbf{b}=\left(a_1+b_1,a_2+b_2,a_3+b_3\right)$$



Scalar Multiplication

- change length of vector $\mathbf{v}$ by α

$$\alpha \cdot \mathbf{v} = (\alpha \cdot v_1, \alpha \cdot v_2, \alpha \cdot v_3)$$



Length

- Length
  - ||v|| gives length (or Euclidean norm) of v

$$\|\mathbf{v}\| = \sqrt{v_1^{\,2} + v_2^{\,2} + v_3^{\,2}}$$

  - if ||v|| is 1, v is called unit vector
  - usually compare length squared
- Normalize
  - v scaled by 1/||v|| gives unit vector $\hat{\mathbf{v}}$

Vector Operations

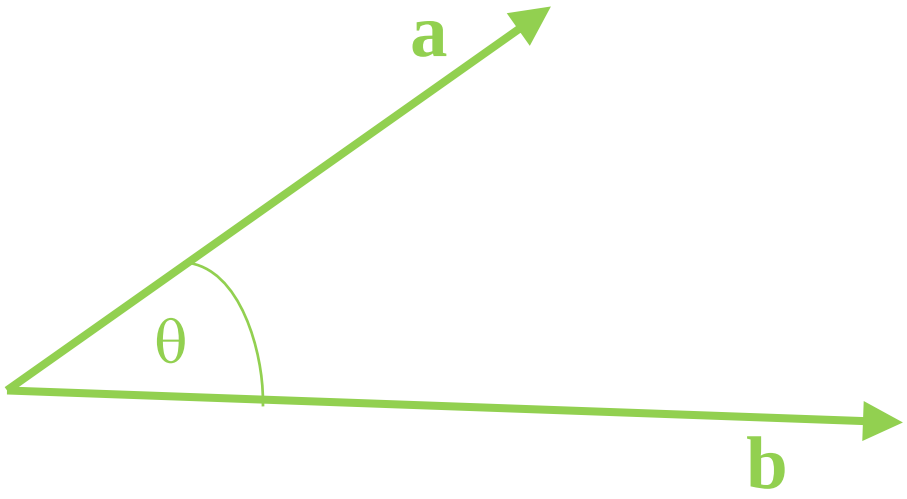
- Games tend to use most of the common vector operations
  - Addition, Subtraction
  - Scalar multiplication
- Two others are *extremely* common:
  - Dot product
  - Cross product

Dot product

- Also called inner product, scalar product

$$\mathbf{a} \bullet \mathbf{b} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

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

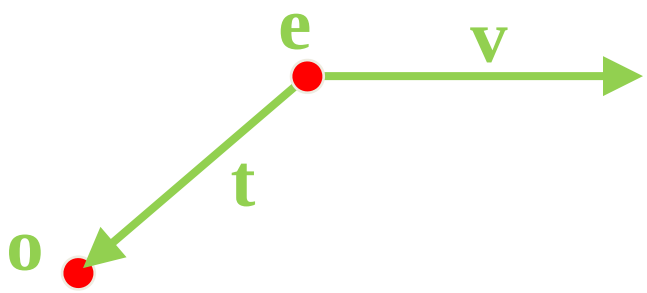


Dot Product: Uses

- $\mathbf{a} \bullet \mathbf{a}$ equals $\|\mathbf{a}\|^2$
- can test for collinear vectors
  - if $\mathbf{a}$ and $\mathbf{b}$ collinear & unit length, $|\mathbf{a} \bullet \mathbf{b}| \sim 1$
  - Problems w/floating point, though
- can test angle/visibility
  - $\mathbf{a} \bullet \mathbf{b} > 0$ if angle < 90°
  - $\mathbf{a} \bullet \mathbf{b} = 0$ if angle = 90° (orthogonal)
  - $\mathbf{a} \bullet \mathbf{b} < 0$ if angle > 90°

Dot Product: Example

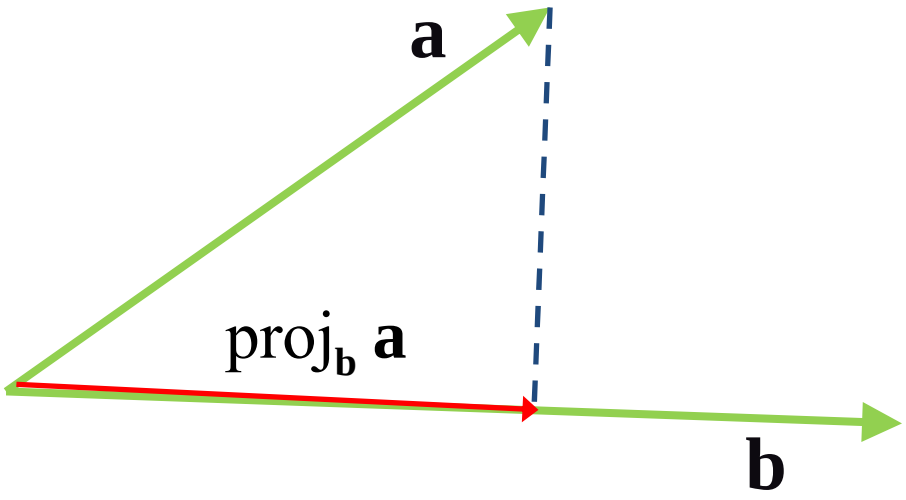
- Suppose have view vector $\mathbf{v}$ and vector $\mathbf{t}$ to object in scene ($\mathbf{t} = \mathbf{o} - \mathbf{e}$)
- If $\mathbf{v} \bullet \mathbf{t} < 0$, object behind us, don't draw



Dot Product: Uses

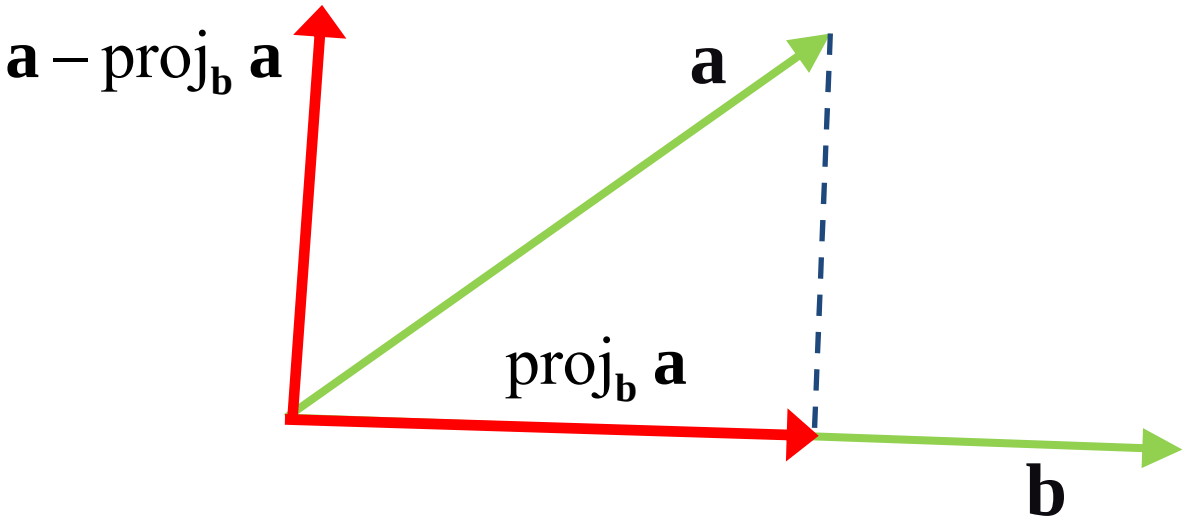
- *Projection* of **a** onto **b** is

$$\text{proj}_{\mathbf{b}}\,\mathbf{a} = \frac{\mathbf{a} \bullet \mathbf{b}}{\mathbf{b} \bullet \mathbf{b}}\mathbf{b}$$



Dot Product: Uses

- Example: break **a** into components collinear and perpendicular to **b**

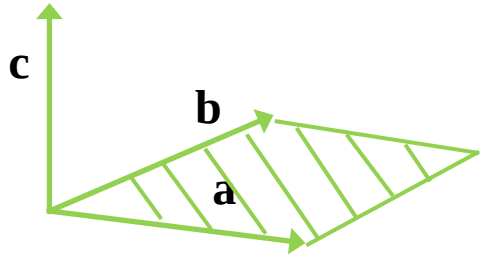


Cross Product

- Cross product: definition

$$\mathbf{a} \times \mathbf{b} = (a_2 b_3 - a_3 b_2, a_3 b_1 - a_1 b_3, a_1 b_2 - a_2 b_1)$$

  - returns vector perpendicular to **a** and **b**
  - right hand rule
  - length = area of parallelogram



Cross Product: Uses

- gives a vector perpendicular to the other two!
- $||\mathbf{a} \times \mathbf{b}|| = ||\mathbf{a}||\ ||\mathbf{b}||\ \sin(\theta)$
- can test collinearity
  - $||\mathbf{a} \times \mathbf{b}|| = 0$ if $\mathbf{a}$ and $\mathbf{b}$ are collinear
  - Better than dot – don't have to be normalized

Other Operations

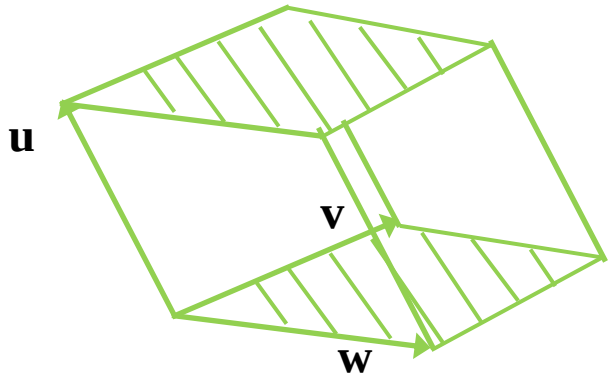
- Several other vector operations used in games may be new to you:
  - Scalar Triple Product
  - Vector Triple Product
- These are often used directly or indirectly in game code, as we'll see

Scalar Triple Product

- Dot product/cross product combo

$$\mathbf{u} \bullet (\mathbf{v} \times \mathbf{w})$$

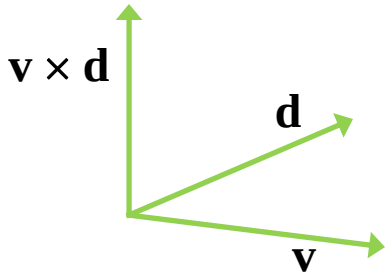
- Volume of parallelpiped

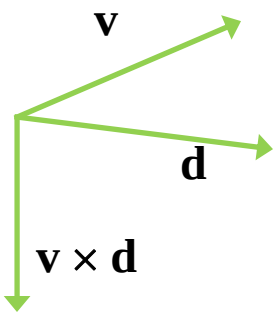


- Test rotation direction
  - Check sign

Triple Scalar Product: Example

- Current velocity $\mathbf{v}$, desired direction $\mathbf{d}$ on $\mathbf{xy}$ plane
- Take $\mathbf{z} \bullet (\mathbf{v} \times \mathbf{d})$
- If > 0, turn left, if < 0, turn right





Vector Triple Product

- Two cross products

$$\mathbf{u} \times (\mathbf{v} \times \mathbf{w})$$

- Useful for building orthonormal basis
  - Compute and normalize:

$$\mathbf{u}$$

$$\mathbf{v} \times \mathbf{u}$$

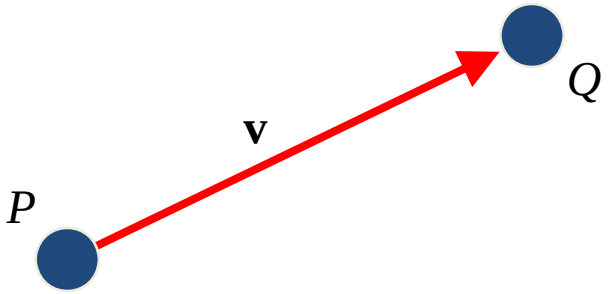
$$\mathbf{u} \times (\mathbf{v} \times \mathbf{u})$$

Points

- Points are positions in space — anchored to origin of coordinate system
- Vectors just direction and length — free-floating in space
- Can’t do all vector operations on points
- But generally use one class in library

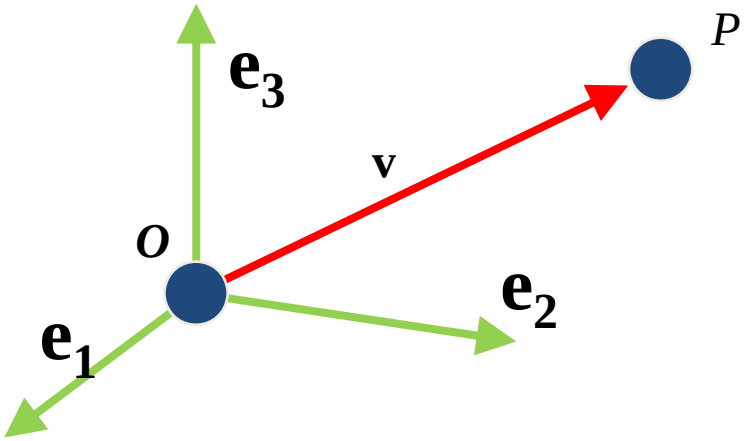
Point-Vector Relations

- Two points related by a vector
  - $(Q - P) = \mathbf{v}$
  - $P + \mathbf{v} = Q$



Affine Space

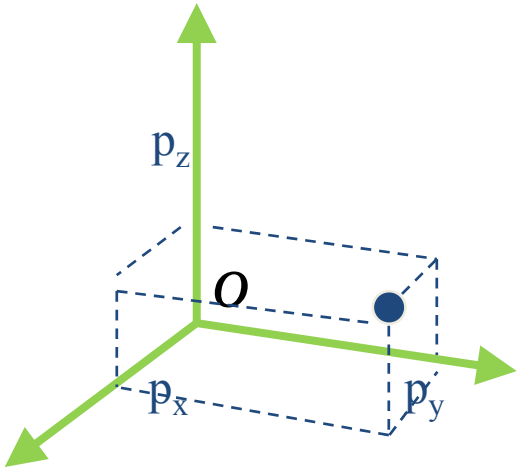
- Vector, point related by *origin*
  - $(P - O) = \mathbf{v}$
  - $O + \mathbf{v} = P$



- Vector space, origin, relation between them make an *affine* space

Cartesian Frame

- Basis vectors $\{\mathbf{i}, \mathbf{j}, \mathbf{k}\}$, origin $(0,0,0)$
- 3D point P represented by (p_x, p_y, p_z)
- Number of units in each axis direction relative to origin

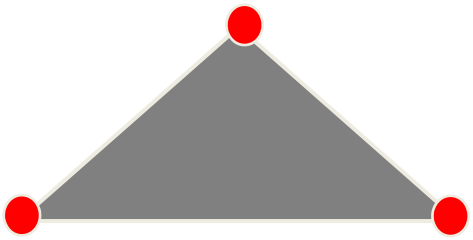


Affine Combination

- Like linear combination, but with points
  - $P = a_1P_1 + a_2P_2 + \ldots + a_nP_n$
  - $a_1,\ldots,a_n$ barycentric coord., add to 1
- Same as point + linear combination
  - $P = P_1 + a_2\,(P_2\text{-}P_1) + \ldots + a_n\,(P_n\text{-}P_1)$
- If vectors $(P_2\text{-}P_1)$, …, $(P_n\text{-}P_1)$ are linearly independent, $\{P_1, \ldots, P_n\}$ called a *simplex* (think of as affine basis)

Convex Combination

- Affine combination with $a_1,\ldots,a_n$ between 0 and 1
- Spans smallest convex shape surrounding points – convex hull
- Example: triangle

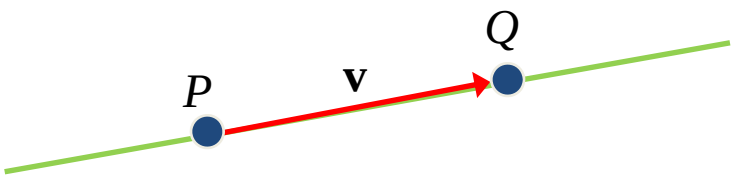


Points, Vectors in Games

- Points used for models, position
  - vertices of a triangle
- Vectors used for velocity, acceleration
  - indicate difference between points, vectors

Parameterized Lines

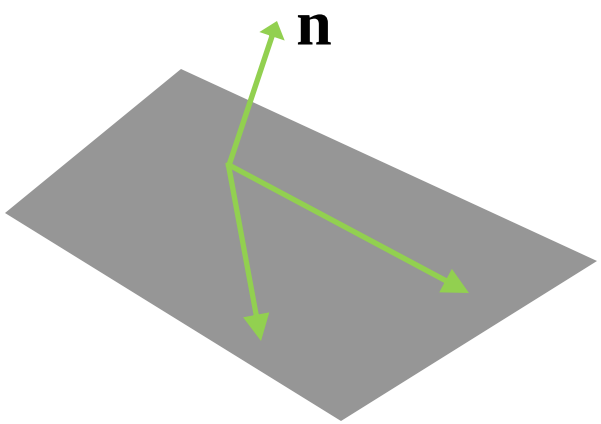
- Can represent line with point and vector
  - $P + t\mathbf{v}$



- Can also represent an *interpolation* from P to Q
  - $P + t(Q\text{-}P)$
  - Also written as $(1\text{-}t)P + tQ$

Planes

- 2 non-collinear vectors *span* a plane
- Cross product is *normal* $\mathbf{n}$ to plane

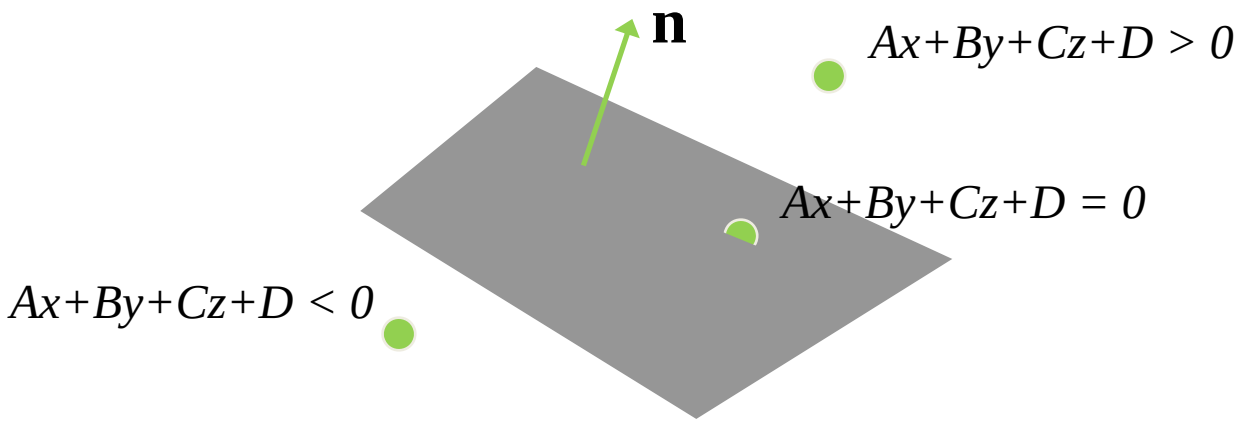


Planes

- Defined by
  - normal $\mathbf{n} = (A, B, C)$
  - point on plane $P0$
- Plane equation
  - $Ax+By+Cz+D = 0$
  - $D=-(A \cdot P0_x + B \cdot P0_y + C \cdot P0_z)$

Planes

- Can use plane equation to test locality of point



- If $\mathbf{n}$ is normalized, gives distance to plane

Transformation

- Have some geometric data
- How to apply functions to it?
- Also desired: combine multiple steps into single operation
- For vectors: linear transformations

Transformations

- A **transformation** $T:V\rightarrow W$ is a function that maps elements from vector space V to W
- The function

$$f(x, y) = x^2 + 2y$$

is a transformation because it maps $\boldsymbol{R}^2$ into $\boldsymbol{R}$

Linear Transformation

- Two basic properties:
  - $T(x + y) = T(x) + T(y)$
  - $T(ax) = aT(x)$
- Follows that
  - $T(0) = 0$
  - $T(ax+y) = aT(x) + T(y)$

Linear Transformations

- Basis vectors span vector space
- Know where basis goes, know where rest goes
- So we can do the following:
  - Transform basis
  - Store as columns in a matrix
  - Use matrix to perform linear transforms

Linear Transforms

- Example:

$$T(x, y) = (x + 2y, 2x + y)$$

- (1,0) maps to (1,2)
- (0,1) maps to (2,1)
- Matrix is

$$\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$$

What is a Matrix?

- Rectangular m x n array of numbers
- M *rows* by n *columns*

$$\begin{pmatrix} 1.3 & 2.4 & 45.3 \\ 2.1 & 0 & 98 \\ 12 & 69 & -20 \end{pmatrix}$$

- If n=m, matrix is *square*

Matrix Concepts

- Number at row i and column j of matrix $\mathbf{A}$ is *element* $\mathbf{A_{ij}}$
- Elements in row i make row vector
- Elems in column j make column vector
- If at least one $\mathbf{A_{ii}}$ (diagonal from upper left to lower right) are non-zero and all others are zero, is *diagonal* matrix

Transpose

- Represented by $\mathbf{A}^{\mathrm{T}}$
- Swap rows and columns along diagonal

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}^{T} = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

- $\mathbf{A^{T}_{ij} = A_{ji}}$
- Diagonal is invariant

Transpose

- Transpose swaps transformed basis vectors from columns to rows
- Useful identity

$$(\mathbf{AB})^T = \mathbf{B}^T \mathbf{A}^T$$

Transforming Vectors

- Represent vector as matrix with one column
- # of components = columns in matrix
- Take dot product of vector w/each row
- Store results in new vector

$$\mathbf{Ax} = \mathbf{b}$$

Transforming Vectors

- Example: 2D vector

$$\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} a_1 b_{11} + a_2 b_{12} \\ a_1 b_{21} + a_2 b_{22} \end{pmatrix}$$

- Example: 3D vector to 2D vector

$$\begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} a_1 b_{11} + a_2 b_{12} + a_3 b_{13} \\ a_1 b_{21} + a_2 b_{22} + a_3 b_{23} \end{pmatrix}$$

Row Vectors

- Can also use row vectors
- Transformed basis stored as rows
- Dot product with columns
- Pre-multiply instead of post-multiply

$$\begin{pmatrix} a_1 & a_2 \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_1 b_{11} + a_2 b_{21} \\ a_1 b_{12} + a_2 b_{22} \end{pmatrix}$$

- If column default, represent row vector by $\mathbf{v}^T$

Row vs. Column

- Using column vectors, others use row vectors
  - Keep your order straight!

$\mathbf{M}_3 \cdot \mathbf{M}_2 \cdot \mathbf{M}_1 \cdot \mathbf{v}$ Column vector order (us, OpenGL)

$\mathbf{v}^T \cdot \mathbf{M}_1 \cdot \mathbf{M}_2 \cdot \mathbf{M}_3$ Row vector order (DirectX)

- Transpose to convert from row to column (and vice versa)

Matrix Product

- Want to combine transforms

$$(T \circ S)(\mathbf{x}) = T(S(\mathbf{x}))$$

- What matrix represents $(T \circ S)$?

- Idea:
  - Columns of matrix for S are xformed basis
  - Transform again by T

Matrix Product

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11} \cdot b_{11} + a_{12} \cdot b_{21} & a_{11} \cdot b_{12} + a_{12} \cdot b_{22} \\ a_{21} \cdot b_{11} + a_{22} \cdot b_{21} & a_{21} \cdot b_{12} + a_{22} \cdot b_{22} \end{pmatrix}$$

or

$$\begin{pmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \end{pmatrix} \cdot \begin{pmatrix} \mathbf{b}_1 & \mathbf{b}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{a}_1 \bullet \mathbf{b}_1 & \mathbf{a}_1 \bullet \mathbf{b}_2 \\ \mathbf{a}_2 \bullet \mathbf{b}_1 & \mathbf{a}_2 \bullet \mathbf{b}_2 \end{pmatrix}$$

- In general, element $\mathbf{AB}_{\mathbf{ij}}$ is dot product of row i from $\mathbf{A}$ and column j from $\mathbf{B}$

Matrix product (cont'd)

- Number of rows in $\mathbf{A}$ must equal number of columns in $\mathbf{B}$
- Generally not commutative

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

- Is associative

$$\mathbf{A}(\mathbf{BC}) = (\mathbf{AB})\mathbf{C}$$

Block Matrices

- Can represent matrix with submatrices

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix}$$

- Product of block matrix contains sums of products of submatrices

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{E} & \mathbf{F} \\ \mathbf{G} & \mathbf{H} \end{pmatrix} = \begin{pmatrix} \mathbf{AE} + \mathbf{BG} & \mathbf{AF} + \mathbf{BH} \\ \mathbf{CE} + \mathbf{DG} & \mathbf{CF} + \mathbf{DH} \end{pmatrix}$$

Identity

- Identity matrix $\mathbf{I}$ is square matrix with main diagonal of all 1s

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

- Multiplying by $\mathbf{I}$ has no effect
  - $\mathbf{A} \cdot \mathbf{I} = \mathbf{A}$

Inverse

- $\mathbf{A}^{-1}$ is inverse of matrix $\mathbf{A}$ such that

$$\mathbf{A} \cdot \mathbf{A}^{-1} = \mathbf{I}$$

- $\mathbf{A}^{-1}$ reverses what $\mathbf{A}$ does
- $\mathbf{A}$ is *orthogonal* if $\mathbf{A}^{\mathrm{T}} = \mathbf{A}^{-1}$
  - Component vectors are at right angles and unit length
  - I.e. orthonormal basis

Computing Inverse

- Only square matrices have inverse
- Inverse doesn’t always exist
- Zero row, column means no inverse
- Use Gaussian elimination or Cramer’s rule (see references)

Computing Inverses

- Most interactive apps avoid ever computing a general inverse
- Properties of the matrices used in most apps can simplify inverse
- If you know the underlying structure of the matrix, you can use the following:

Computing Inverse

- If orthogonal, $\mathbf{A}^{-1} = \mathbf{A}^{T}$
- Inverse of diagonal matrix is diagonal matrix with $\mathbf{A}^{-1}_{ii} = 1/\mathbf{A}_{ii}$
- If know underlying structure can use

$$(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$$

- We'll use this to avoid explicit inverses

Storage Format

- Row major
  - Stored in order of rows

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{pmatrix}$$

  - Used by DirectX

Storage Format (cont'd)

- Column Major Order
  - Stored in order of columns

$$\begin{pmatrix} 0 & 4 & 8 & 12 \\ 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \end{pmatrix}$$

  - Used by OpenGL, and us

Storage Format (cont'd)

- Note: storage format not the same as multiplying by row vector
- Same memory footprint:
  - Matrix for multiplying column vectors in column major format
  - Matrix for multiplying row vectors in row major format
- I.e. two transposes return same matrix

System of Linear Equations

- Define system of *m* linear equations with *n* unknowns

$$b_1 = a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n$$
$$b_2 = a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n$$
$$\ldots$$
$$b_m = a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n$$

References

- Anton, Howard and Chris Rorres, *Elementary Linear Algebra,* 7th Ed, Wiley & Sons, 1994.
- Axler, Sheldon, *Linear Algebra Done Right,* Springer Verlag, 1997.
- Blinn, Jim, *Notation, Notation, Notation*, Morgan Kaufmann, 2002.
- Van Verth, James M. and Lars M. Bishop, *Essential Mathematics For Games & Interactive Applications*, Morgan Kaufmann, San Francisco, 2004.