6.837 Linear Algebra Review

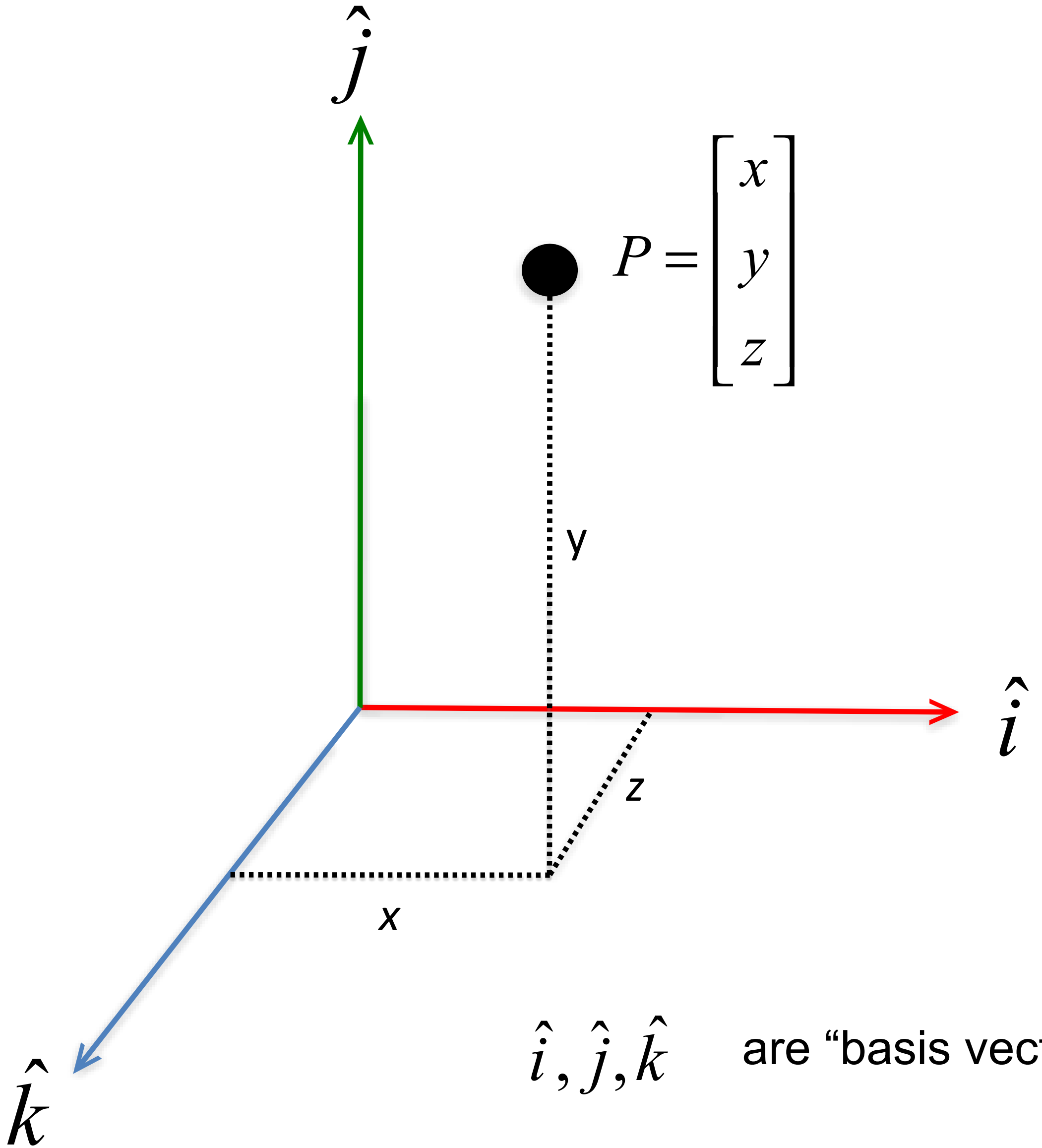
Jiawen “Kevin” Chen

jiawen@mit.edu

Overview

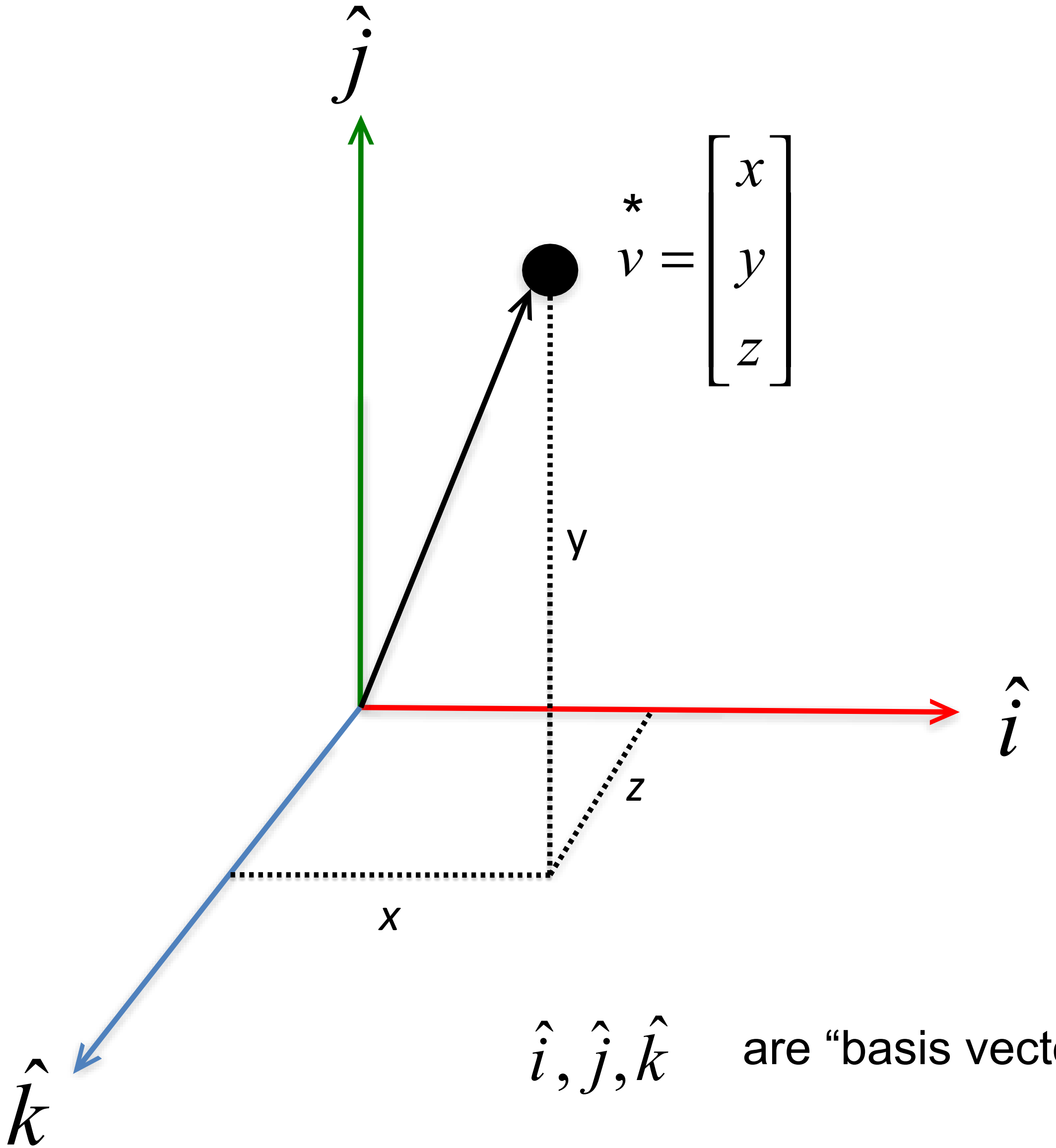
- Linear algebra
  - Points, Vectors in R^3
  - Operations (norm, dot product, cross-product)
- Geometry
  - lines, planes
- Matrices
  - Transformations

A point



$\hat{i}, \hat{j}, \hat{k}$ are "basis vectors"

A vector

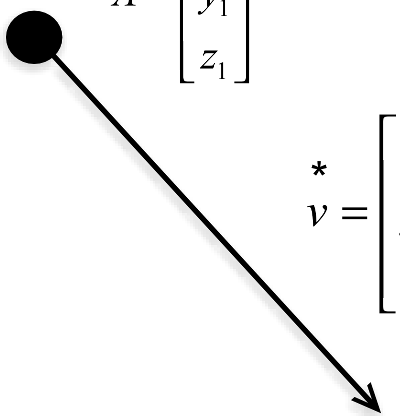


$\hat{i}, \hat{j}, \hat{k}$ are “basis vectors”

Points vs. vectors

- Same form: $[x,y,z]^T$
- Same class in vecmath library: `Vector3f`
- Both are specified as numbers:
  - *coordinates* w.r.t. some basis
- But, conceptually:
  - points are positions
  - vectors have *direction* and *length*
- In homogeneous coordinates, you can think of
  - points as [x,y,z,1]
  - vectors as [x,y,z,0]

point + vector = _____

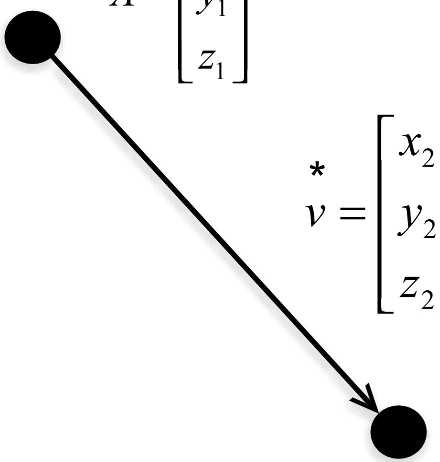


$$A = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}$$

$$\overset{*}{v} = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}$$

point + vector = point

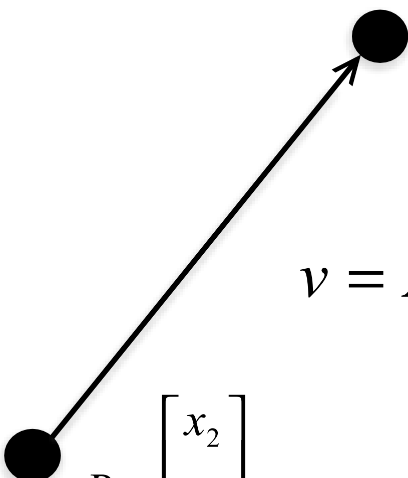
$$A = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}$$



$$\overset{*}{v} = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}$$

$$B \quad = A + \overset{*}{v} = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} + \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} x_1 + x_2 \\ y_1 + y_2 \\ z_1 + z_2 \end{bmatrix}$$

point – point = vector



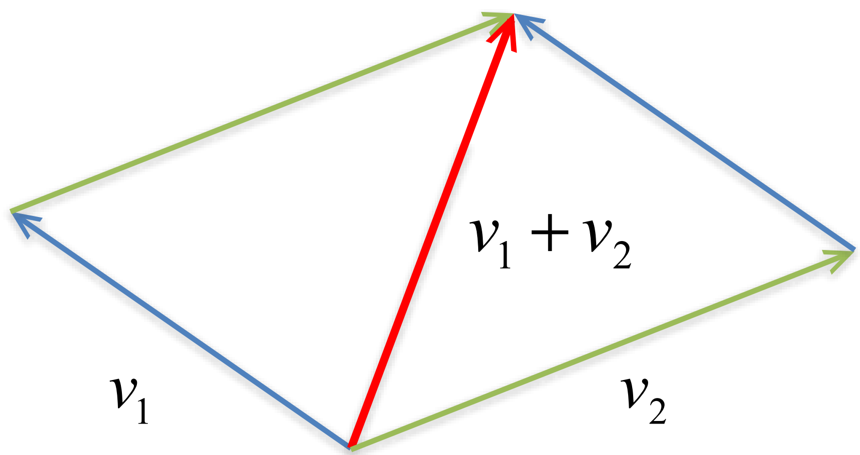
$$A = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}$$

$$v = A - B$$

$$B = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}$$

$$\overset{*}{v} = A - B = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} - \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} x_1 - x_2 \\ y_1 - y_2 \\ z_1 - z_2 \end{bmatrix}$$

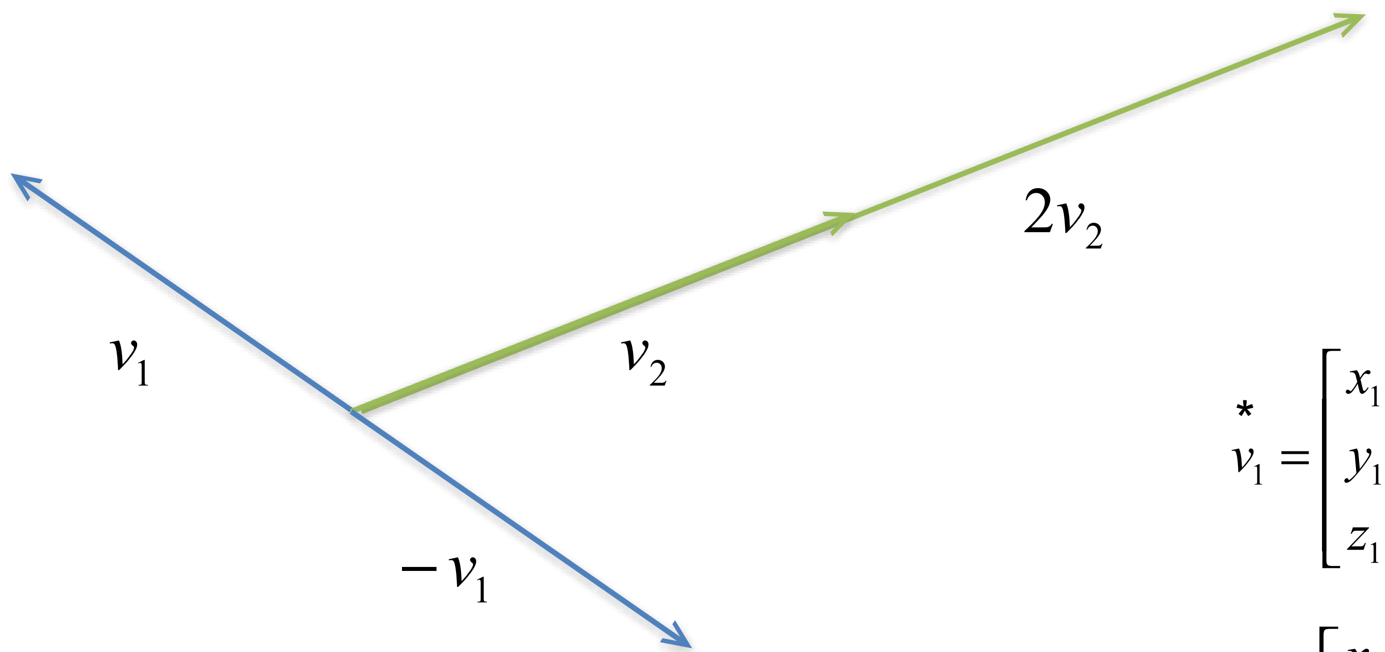
vector + vector = vector



$$\overset{*}{v}_1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \qquad \overset{*}{v}_2 = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}$$

$$\overset{*}{v}_1 + \overset{*}{v}_2 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} + \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} x_1 + x_2 \\ y_1 + y_2 \\ z_1 + z_2 \end{bmatrix} = \overset{*}{v}_2 + \overset{*}{v}_1$$

Vector operations



$$\overset{*}{v}_1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \quad -\overset{*}{v}_1 = \begin{bmatrix} -x_1 \\ -y_1 \\ -z_1 \end{bmatrix}$$

$$\overset{*}{v}_2 = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} \quad 2\overset{*}{v}_2 = \begin{bmatrix} 2x_2 \\ 2y_2 \\ 2z_2 \end{bmatrix}$$

Vector space axioms

$$u + (v + w) = (u + v) + w$$

$$v + w = w + v$$

$$a(v + w) = av + aw$$

$$v + 0 = v$$

$$(a + b)v = av + bv$$

$$v + w = 0 \Rightarrow w = -v$$

$$a(bv) = (ab)v$$

$$1v = v$$

$$v - w = w + (-v)$$

$$\frac{v}{a} = \left(\frac{1}{a}\right)v^*$$

More vector operations

$$\vec{v}_1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \qquad \vec{v}_2 = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}$$

Dot product $\quad \vec{v}_1 \cdot \vec{v}_2 = x_1x_2 + y_1y_2 + z_1z_2 = \vec{v}_1^{\,T}\vec{v}_2$

Norm (length) $\quad \|\vec{v}_1\| = \sqrt{\vec{v}_1 \cdot \vec{v}_1} = \sqrt{x_1^{\,2} + y_1^{\,2} + z_1^{\,2}}$

Normalization $\quad \hat{v}_1 = \dfrac{\vec{v}_1}{\|\vec{v}_1\|} \longrightarrow \|\hat{v}_1\| = 1$

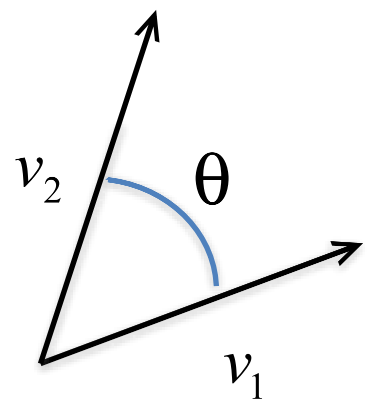
Properties of the dot product

commutative $v_1 \cdot v_2 = v_2 \cdot v_1$

distributive $v_1 \cdot (v_2 + v_3) = v_1 \cdot v_2 + v_1 \cdot v_3$

$$(av_1) \cdot (bv_2) = (ab)(v_1 \cdot v_2)$$

Angle between two vectors

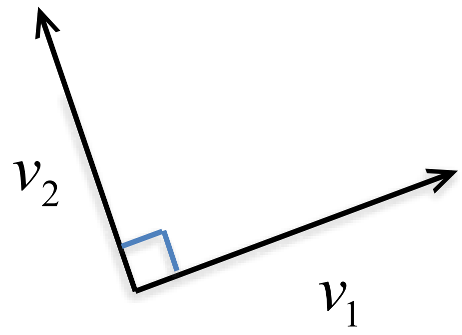


$$\cos\theta = \frac{v_1 \cdot v_2}{|v_1||v_2|}$$

Orthogonal vectors

- Two vectors are *orthogonal* if:

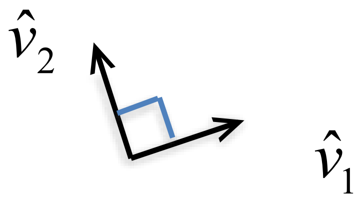
$$v_1 \cdot v_2 = 0$$



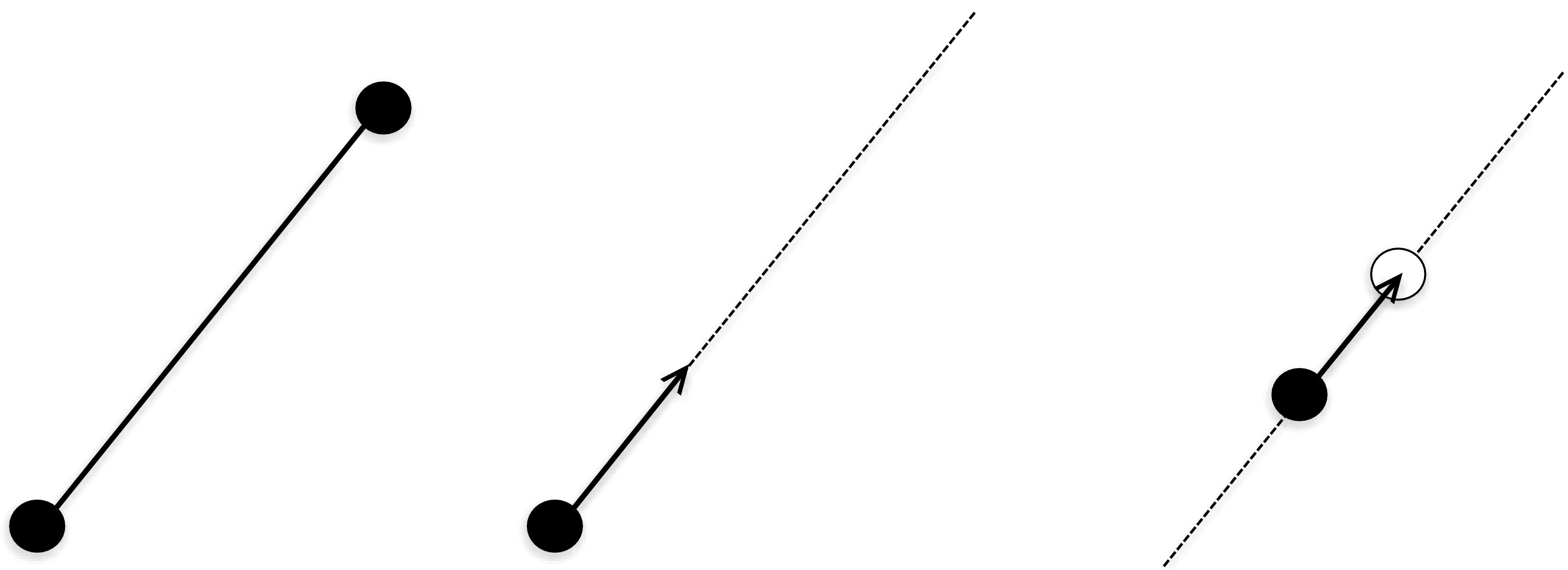
Orthonormal vectors

- Two vectors are *orthonormal* if:

$$v_1 \cdot v_2 = 0 \qquad \left\| v_1 \right\| = 1 \qquad \left\| v_2 \right\| = 1$$



Lines

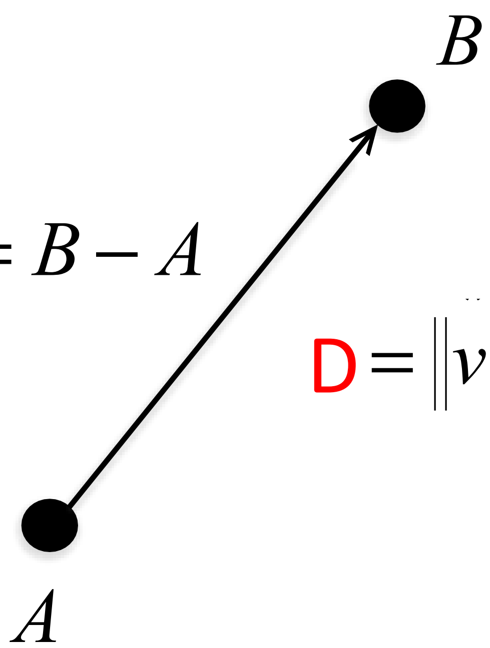


Line segment:
two endpoints

Ray (half-line):
origin + direction vector
$P = O + t * d, t > 0$

Line:
two points define a line
$P = O + t * d$

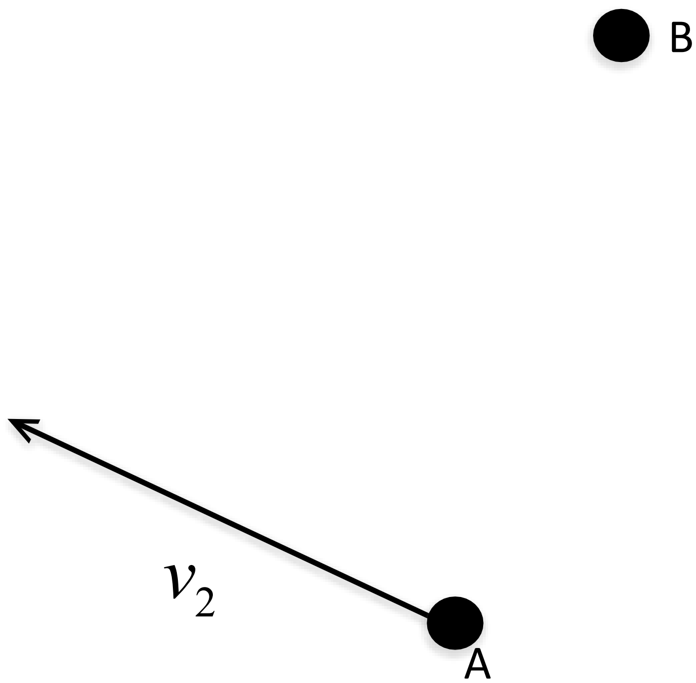
Distance between points



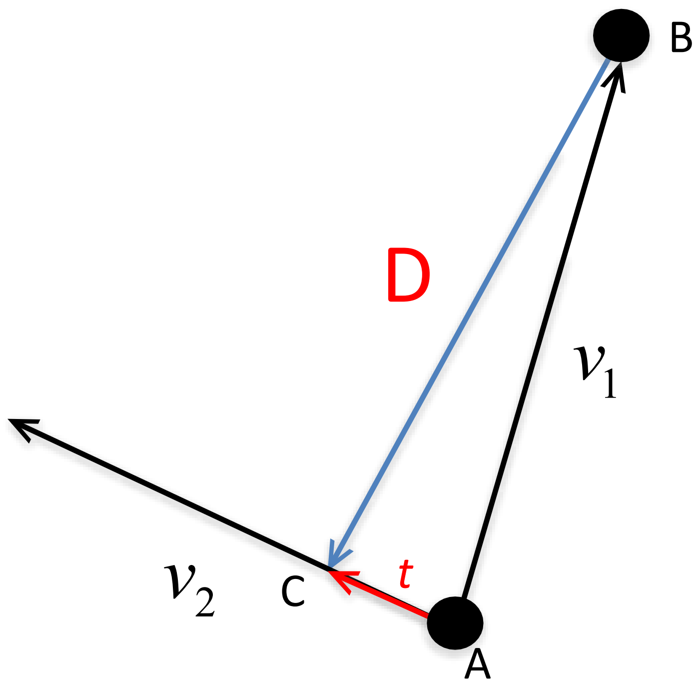
$$v = B - A$$

$$\mathrm{D} = \left\| v \right\|$$

closest distance from point to line



closest distance from point to line



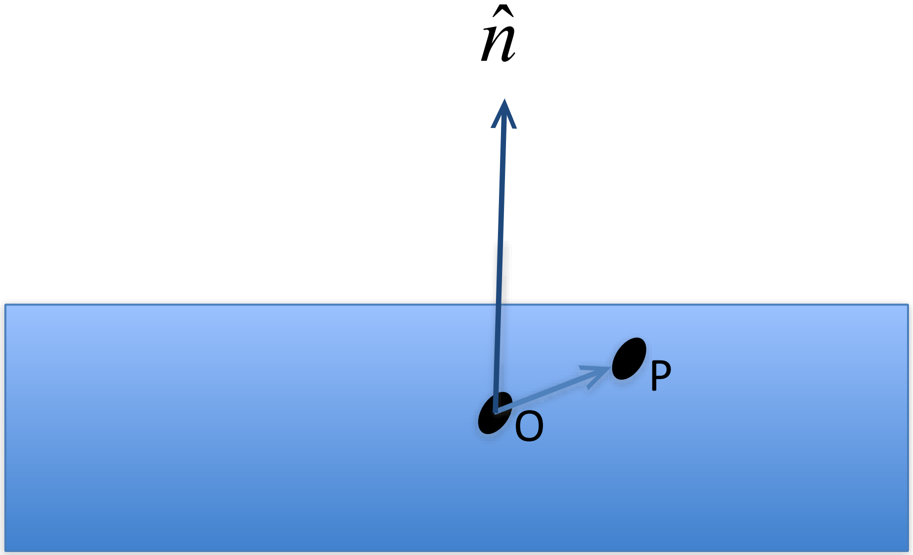
$$v_1 = B - A$$

$$t = \overset{*}{v}_1 \cdot \frac{\underset{*}{v_2}}{\|v_2\|}$$ Why?

$$C = A + t\,\frac{\underset{*}{v_2}}{\|v_2\|}$$

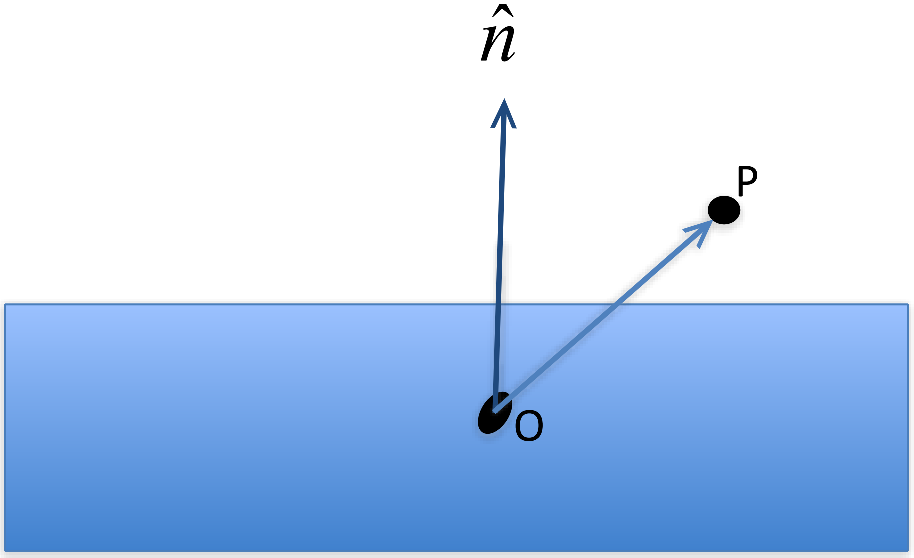
$$D = \|C - B\|$$

A plane



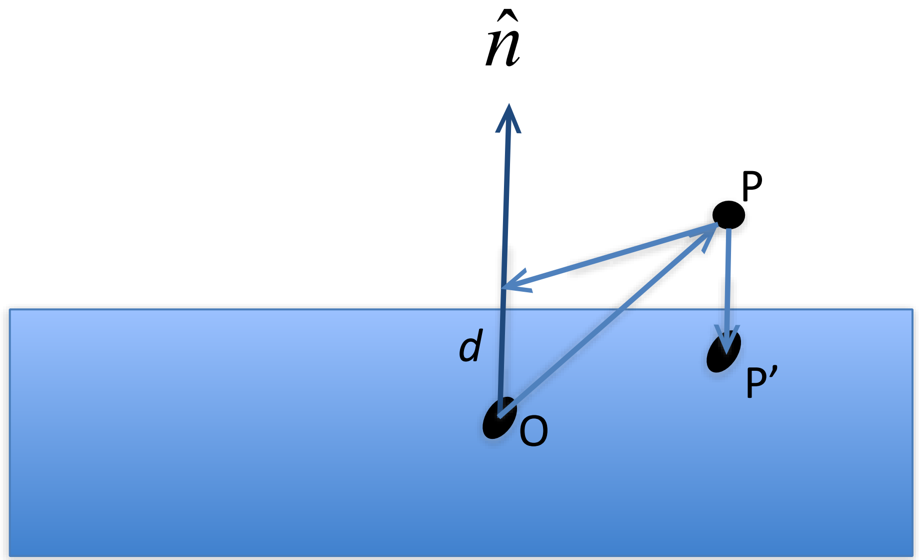
$$(P - O) \cdot \hat{n} = 0$$

closest distance from point to plane



$$(P - O) \cdot \hat{n} = ?$$

closest distance from point to plane

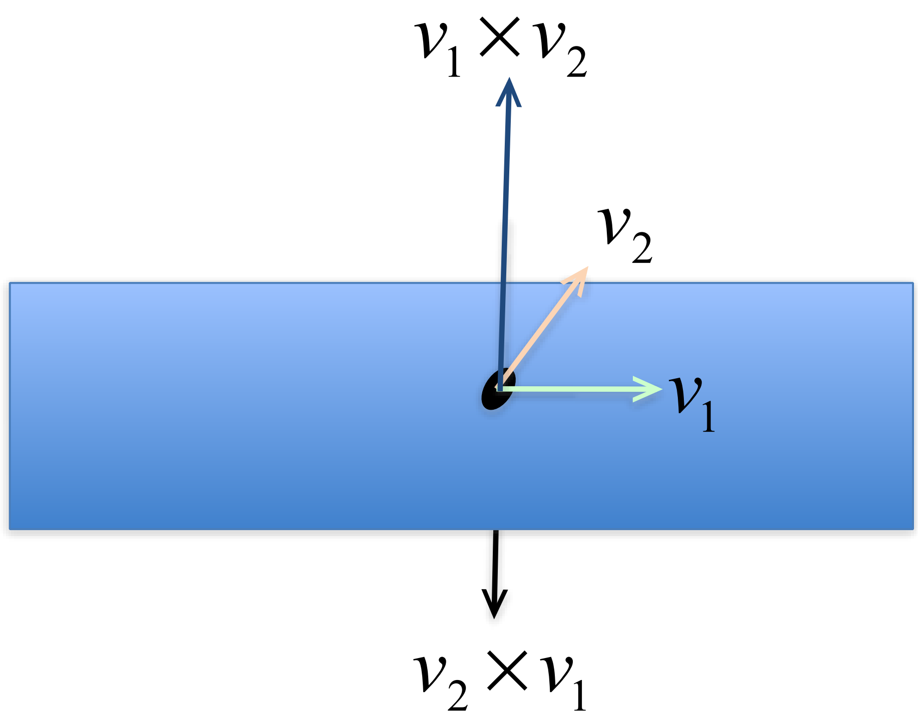


$$\left(P - O\right) \cdot \hat{n} = d$$

$$P' = P - d\hat{n}$$

$$\left(P' - O\right) \cdot \hat{n} = \left(P - d\hat{n} - O\right) \cdot \hat{n} = \left(P - O\right) \cdot \hat{n} - d\hat{n} \cdot \hat{n} = d - d = 0$$

The cross product



$$\overset{*}{v}_1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \quad \overset{*}{v}_2 = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} \quad \overset{*}{v}_1 \times \overset{*}{v}_2 = \begin{bmatrix} 0 & -z_1 & y_1 \\ z_1 & 0 & x_1 \\ y_1 & x_1 & \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}$$

Properties of the cross product

anti-commutative $$u \times v = -(v \times u)$$

distributive over:

addition $$u \times (v + w) = (u \times v) + (u \times w)$$

scalar multiplication $$(au) \times v = u \times (av) = a(u \times v)$$

Example using vecmath

```
Vector3f pt1( 1, 1, 1 );
Vector3f pt2( 4, 7, 6 );
Vector3f v1 = pt2 - pt1;

cout << “pt1: ”; pt1.print();
cout << “pt2: ”; pt2.print();
cout << “v1: ”; v1.print();

Vector3f v2( -3, 4, 5 );
cout << “v2: ”; v2.print();

Vector3f v1_cross_v2 = Vector3f::cross( v1, v2 );
cout << “v1 x v2: ”; v1_cross_v2.print();
cout << “|v1 x v2|: ” << v1_cross_v2.abs() << endl;

v1_cross_v2.normalize();
cout << “v1 x v2: ”; v1_cross_v2.print();
cout << “|v1 x v2|: ” << v1_cross_v2.abs() << endl;

float v1_dot_v2 = Vector3f::dot( v1, v2 );
cout << “v1 . v2: ” << v1_dot_v2 << endl;

float v1_cross_v2__dot_v2 = Vector3f::dot( Vector3f::cross( v1, v2 ), v2 );
cout << “( v1 x v2 ) . v2: ” << v1_cross_v2__dot_v2 << endl;
```

pt1 : <1 1 1>
pt2 : <4 7 6>
v1 : <3 6 5>
v2 : <-3 4 5>

v1 x v2 : <10 -30 30>
|v1 x v2| : <43.589>

v1 x v2 : <0.229 -0.688 0.688>
|v1 x v2| : 1

v1 . v2 : 40

v1 x v2 . v2 : 0

Matrix operations

$$A=\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \qquad A^T=\begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{bmatrix}$$

$$AA=A^2=\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}=\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

Not commutative $AB \neq BA$

$$\det(AB)=\det(BA)$$

Associative $A(BC)=(AB)C$

$$c(AB)=(cA)B$$

$$(Ac)B=A(cB)$$

$$(AB)c=A(Bc)$$

(where c is a scalar)

3x3 matrix inverse

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$A^{-1} = 1/\mathrm{DET} * \left| \begin{array}{ccc} a_{33}a_{22}-a_{32}a_{23} & -(a_{33}a_{12}-a_{32}a_{13}) & a_{23}a_{12}-a_{22}a_{13} \\ -(a_{33}a_{21}-a_{31}a_{23}) & a_{33}a_{11}-a_{31}a_{13} & -(a_{23}a_{11}-a_{21}a_{13}) \\ a_{32}a_{21}-a_{31}a_{22} & -(a_{32}a_{11}-a_{31}a_{12}) & a_{22}a_{11}-a_{21}a_{12} \end{array} \right|$$

$$\mathrm{DET} = a_{11}(a_{33}a_{22}-a_{32}a_{23})-a_{21}(a_{33}a_{12}-a_{32}a_{13})+a_{31}(a_{23}a_{12}-a_{22}a_{13})$$

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

Matrix is *singular* (not invertible) when det = 0.

In vecmath:

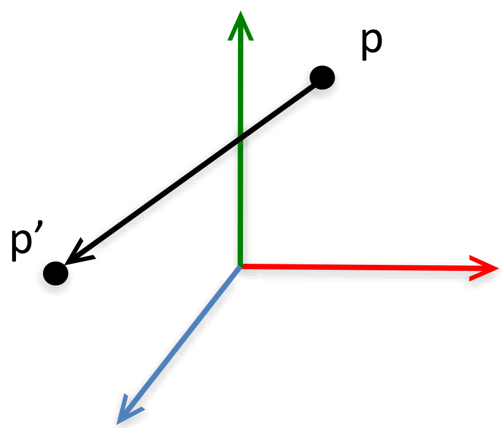
```
Matrix3f A( … );
bool isSingular;
Matrix3f invA = A.inverse( &isSingular, 0.001f );
```

Matrices and Vectors

$$M = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \qquad \overset{*}{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

$$M\overset{*}{v} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} a_{11}v_1 + a_{12}v_2 + a_{13}v_3 \\ a_{21}v_1 + a_{22}v_2 + a_{23}v_3 \\ a_{31}v_1 + a_{32}v_2 + a_{33}v_3 \end{bmatrix}$$

Transform Matrices



$$\overset{*}{p}' = M\overset{*}{p} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$

$$M = TR \neq RT$$

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation or translation first?

What point are we rotating around?

In vecmath

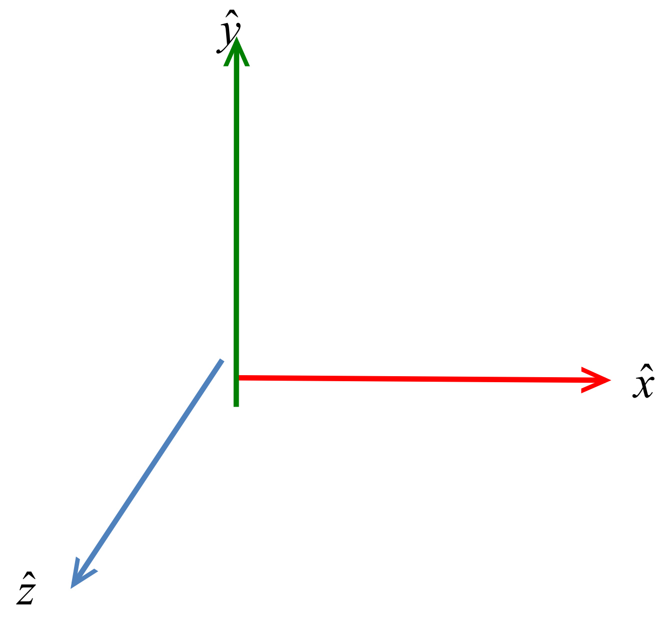
```
Matrix4f T = Matrix4f::translation( Vector3f( 1, 2, 3 ) );
Matrix4f R = Matrix4f::rotation
    ( Vector3f( 1, 0, 0 ), M_PI / 4.0f );

cout << "T = " << endl; T.print();
cout << "R = " << endl; R.print();
cout << "T*R = " << endl; ( T * R ).print();
cout << "R*T = " << endl; ( R * T ).print();

Vector4f p( 0, 0, 0, 1 );

Vector4f transformed_p = T * R * p;
cout << "T*R: "; transformed_p.print();

transformed_p = R * T * p;
cout << "R*T: "; transformed_p.print();
```

```
T =
[1 0 0 1]
[0 1 0 2]
[0 0 1 3]
[0 0 0 1]

R =
[1 0 0 0]
[0 0.707107 -0.707107 0]
[0 0.707107 0.707107 0]
[0 0 0 1]

T*R =
[1 0 0 1]
[0 0.707107 -0.707107 2]
[0 0.707107 0.707107 3]
[0 0 0 1]

R*T =
[1 0 0 1]
[0 0.707107 -0.707107 -0.707107]
[0 0.707107 0.707107 3.53553]
[0 0 0 1]

T*R: [1 2 3 1]
R*T: [1 -0.707107 3.53553 1]
```

In OpenGL

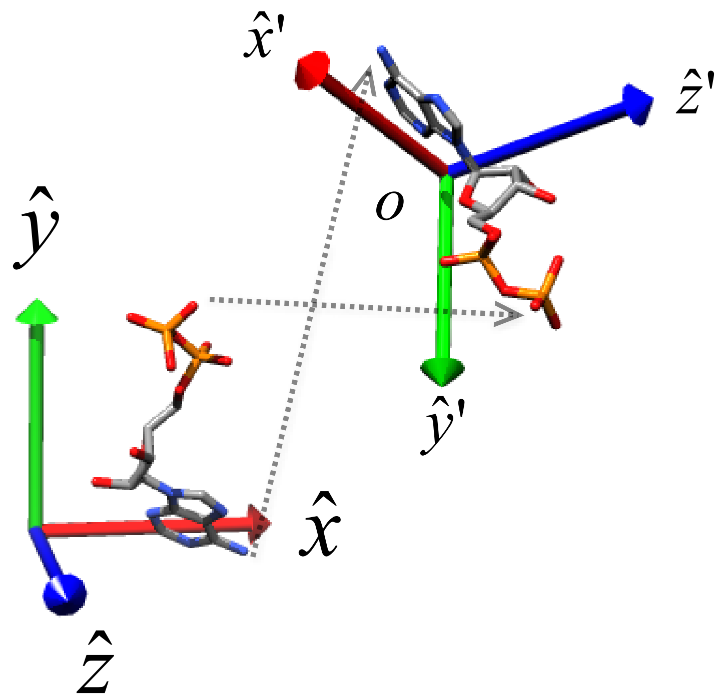
```
glMatrixMode( GL_MODELVIEW );  // Current matrix affects objects positions
glLoadIdentity();              // Initialize to the identity

// Position the camera at [0,0,5], looking at [0,0,0],
// with [0,1,0] as the up direction.
gluLookAt(0.0, 0.0, 5.0,
          0.0, 0.0, 0.0,
          0.0, 1.0, 0.0);

glTranslated ( 1, 2, 3 ); // the translation is applied second
glRotated ( 45.0, 1, 0, 0 ); // the rotation is applied first

// draw object
```

Orthonormal basis



$$\hat{x} \cdot \hat{y} = \hat{x} \cdot \hat{z} = \hat{y} \cdot \hat{z} = 0$$

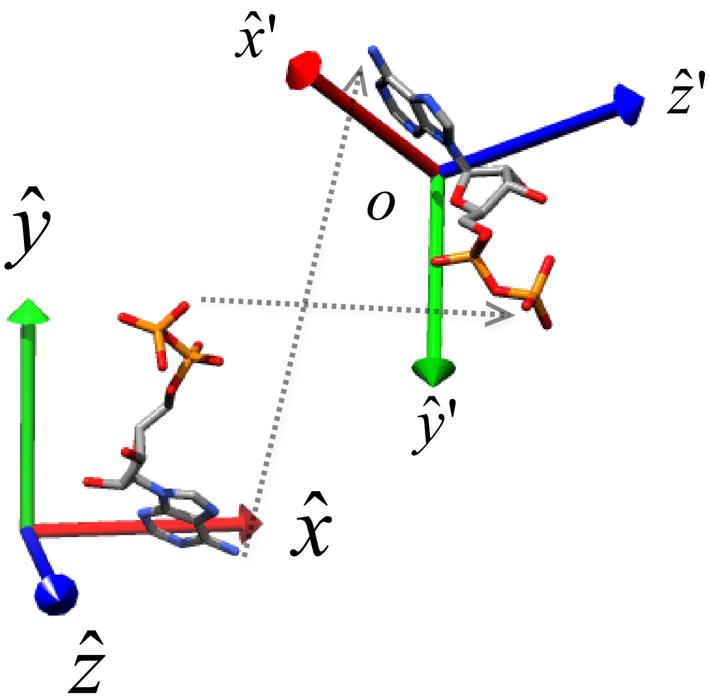
Frames of reference



$$T = \begin{bmatrix} \hat{x}' & \hat{y}' & \hat{z}' & o \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T = \begin{bmatrix} \hat{x}'_x & \hat{y}'_x & \hat{z}'_x & o_x \\ \hat{x}'_y & \hat{y}'_y & \hat{z}'_y & o_y \\ \hat{x}'_z & \hat{y}'_z & \hat{z}'_z & o_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Frames of reference

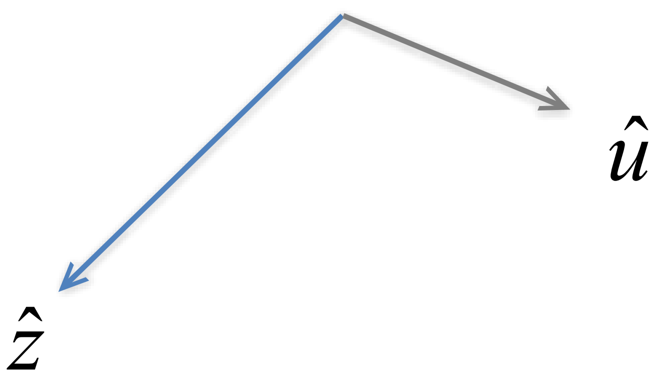


$$\begin{bmatrix} \hat{x}'_x & \hat{y}'_x & \hat{z}'_x & o_x \\ \hat{x}'_y & \hat{y}'_y & \hat{z}'_y & o_y \\ \hat{x}'_z & \hat{y}'_z & \hat{z}'_z & o_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \hat{x}'$$

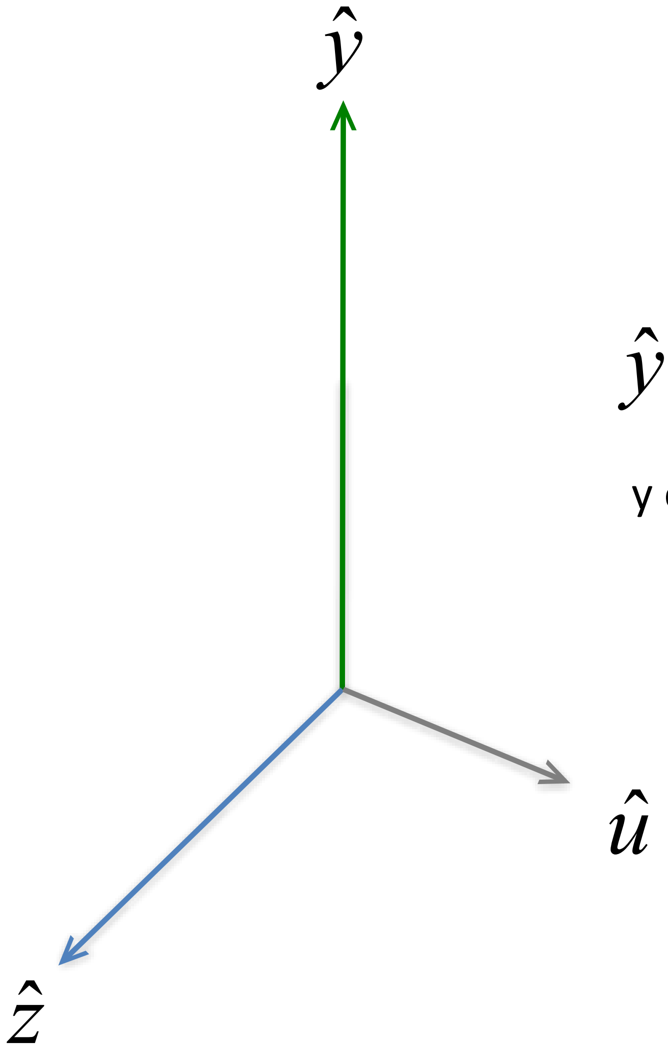
$$\begin{bmatrix} \hat{x}'_x & \hat{y}'_x & \hat{z}'_x & o_x \\ \hat{x}'_y & \hat{y}'_y & \hat{z}'_y & o_y \\ \hat{x}'_z & \hat{y}'_z & \hat{z}'_z & o_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \hat{o} + \hat{x}'$$

How to create orthonormal basis

Given vectors u and z,
u not orthogonal to z



How to create orthonormal basis



$$\hat{y} = \hat{z} \times \hat{u}$$

y orthogonal to both z and u

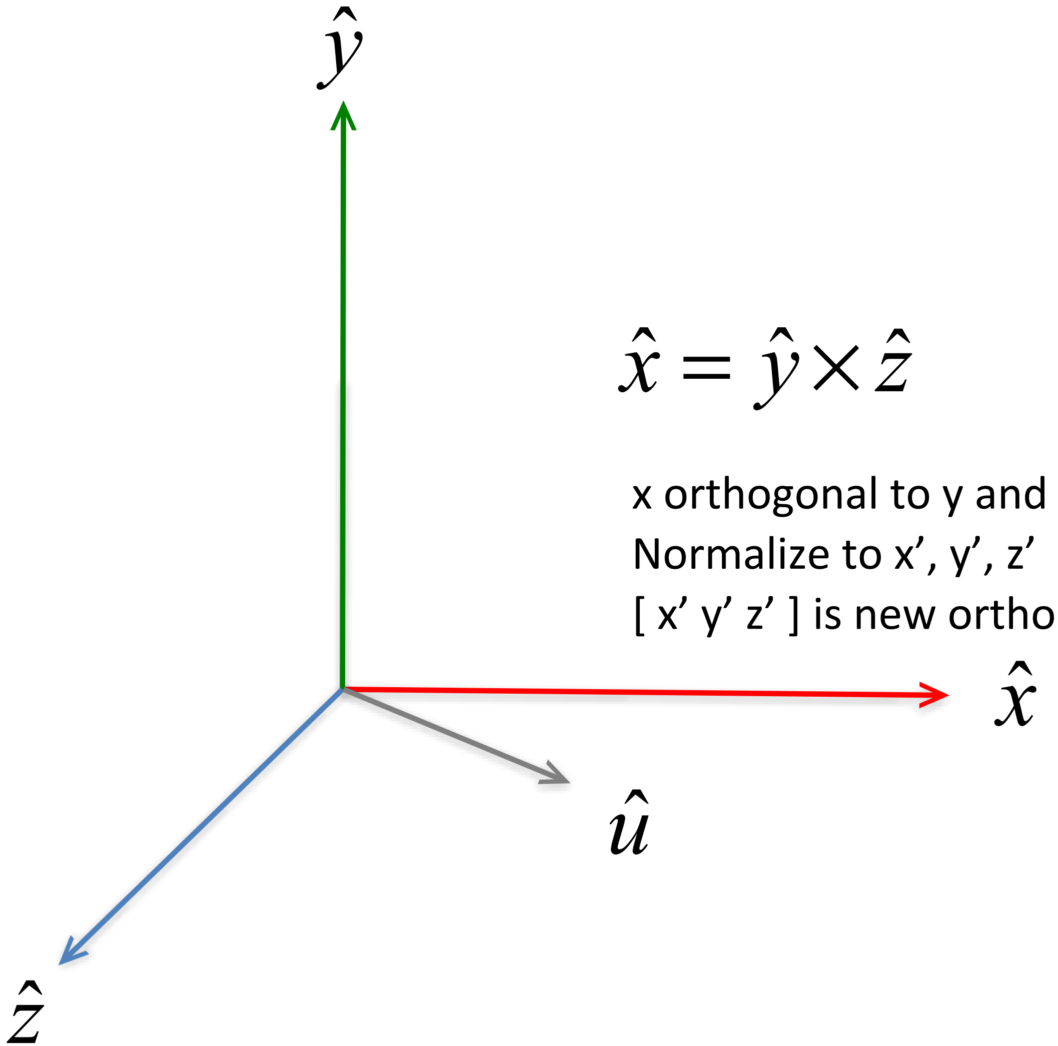
How to create orthonormal basis

$$\hat{x} = \hat{y} \times \hat{z}$$

x orthogonal to y and z

Normalize to x', y', z'

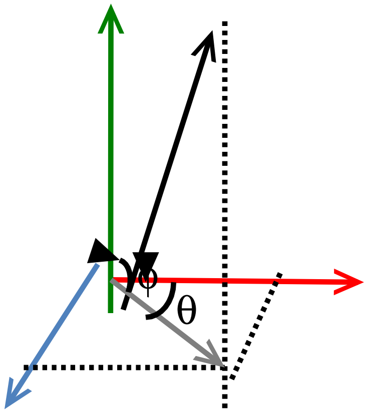
[x' y' z'] is new orthonormal basis



More on rotation matrices

- Rotation matrices
  - 9 variables
  - Only 3 degrees of freedom
    - Rotation axis: 3 numbers, but…
      - But given x, y, z^2 = 1 – x^2 – y^2
      - Or two angles: *azimuth* and *elevation*
    - 1 for rotation angle
  - `Matrix4f::rotation( axis, angle )`

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$



- Problems
  - Error accumulates after concatenating rotations by matrix multiplication, due to limited precision (“drift”)
    - Orthogonalization is non-trivial

- Solution: quaternions
  - In lecture on 9/30

Quaternions

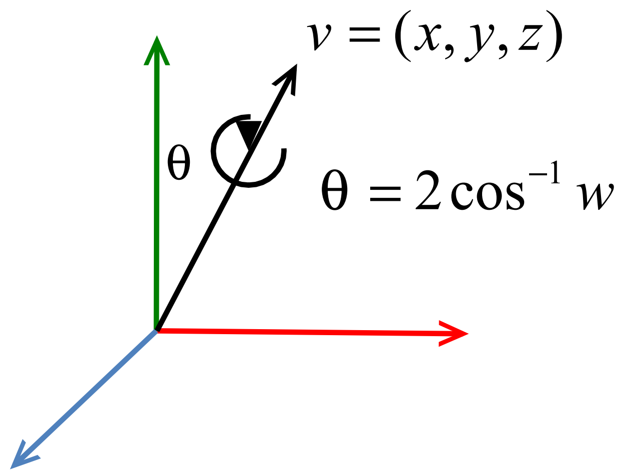
$$Q = w + xi + yj + zk \qquad i^2 = j^2 = k^2 = ijk = -1$$

For rotations, we use normalized quaternions, where:

$$w^2 + x^2 + y^2 + z^2 = 1$$

Quaternions

$$Q = w + xi + yj + zk$$



Composit rotations by multiplying quaternions: $Q^3 = Q^2 Q^1 \left(\neq Q^1 Q^2 \right)$

Quaternions with VL

Use Vec4 for quaternion:

```
Vec4d Q(TVReal x, TVReal y, TVReal z, TVReal w);
```

To convert it into a rotation matrix:

```
Mat3d M; M.MakeHRot(Q);
```

(multiplication and normalization, you may have to implement yourself)

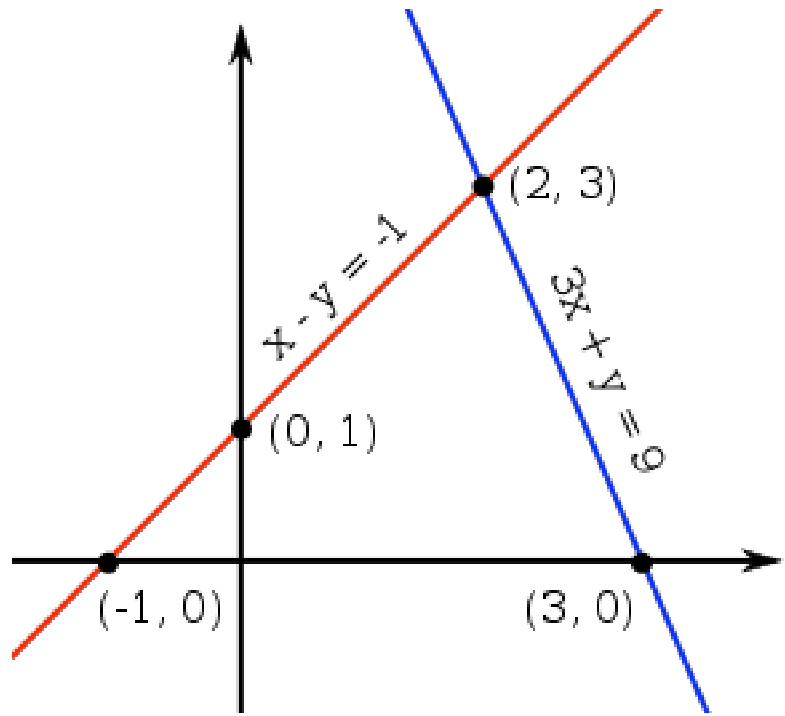
Linear systems of equations

$$x - 3y = 1$$
$$3x + y = 9$$

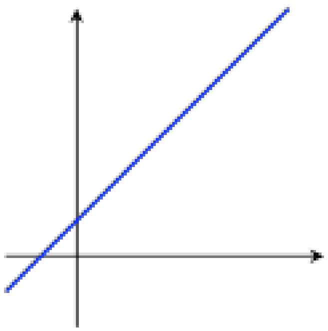
$$\begin{bmatrix} 1 & -3 \\ 3 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 9 \end{bmatrix}$$

$$A\hat{x} = b$$

$$\overset{*}{x} = A^{-1}\overset{*}{b} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

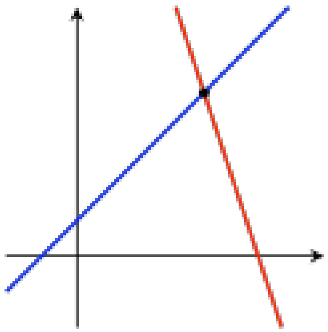


Solutions of linear systems



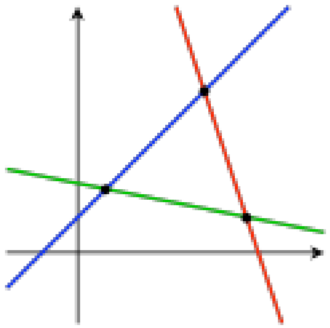
$$x - 3y = 1$$

- One equation, two variables
- Underdetermined
- Infinite solutions



$$\begin{aligned} x - 3y &= 1 \\ 3x + y &= 9 \end{aligned}$$

- Two equations, two variables
- Unique solution if equations are independent
- Infinite solutions if equations are dependent

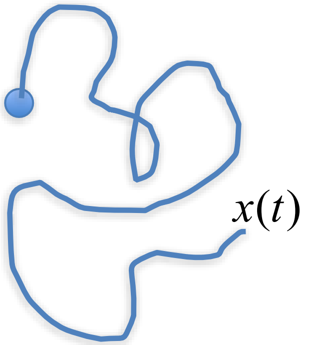


$$\begin{aligned} x - 3y &= 1 \\ 3x + y &= 9 \\ -x + 2y &= 3 \end{aligned}$$

- Three equations, two variables
- Overdetermined
- No solution if equations are independent
- Unique solution if any two of the equations are dependent
- Infinite solutions if all equations are dependent

Ordinary differential equations

- E.g. Newton's law: $ma = F$



$$m\frac{d^2x(t)}{dt^2} = 0$$

Can solve analytically

$$m\frac{d^2x(t)}{dt^2} = F(x(t))$$

Can't solve analytically when $F(x(t))$ is complicated

$$m\frac{dv(t)}{dt} = F(x(t)) \qquad \frac{dx(t)}{dt} = v(t)$$

First-order forms

Forward-Euler method

Taylor expansion:

$$m\frac{dv(t)}{dt} = F\left(\overset{*}{x}(t)\right)$$

$$\overset{*}{v}(t+h) = \overset{*}{v}(t) + h\frac{d\ddot{v}(t)}{dt} + h^2\frac{d^2\ddot{v}(t)}{dt^2} + \ldots$$

$$\frac{d\ddot{v}(t)}{dt} = \frac{\ddot{v}(t+h) - \ddot{v}(t)}{h} - h\cancel{\frac{d^2\ddot{v}(t)}{dt^2}}$$

$$Error \propto h$$

$$m\frac{v(t+h) - v(t)}{h} = F\left(\overset{*}{x}(t)\right)$$

$$\overset{*}{v}(t+h) = \overset{*}{v}(t) + \frac{h}{m}F\left(\overset{*}{x}(t)\right)$$

$$\overset{*}{v}_{t+dt} = \overset{*}{v}_t + \frac{dt}{m}F\left(\overset{*}{x}_t\right)$$

Initial condition

$$v_{t=0} = v_0$$

Forward-Euler method

$$\overset{*}{v}_{t+dt} = \overset{*}{v}_t + \frac{dt}{m} F\left(\overset{*}{x}_t\right)$$

Initial condition

$$v_{t=0} = v_0$$

Treat x,y,z components independently:

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}_{t+dt} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}_t + \frac{dt}{m} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}_t \qquad \begin{bmatrix} x_x \\ x_y \\ x_z \end{bmatrix}_{t+dt} = \begin{bmatrix} x_x \\ x_y \\ x_z \end{bmatrix}_t + dt \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}_t$$

Problems: stability (pick a small *dt*), small masses, error

Runge-kutta method

$$k_1 = \frac{dt}{m} F\left(\overset{*}{x}_t\right)$$

$$k_2 = \frac{dt}{m} F\left(\overset{*}{x}_{t+k_1/2}\right)$$

$$k_3 = \frac{dt}{m} F\left(\overset{*}{x}_{t+k_2/2}\right)$$

$$k_4 = \frac{dt}{m} F\left(\overset{*}{x}_{t+k_3}\right)$$

$$\overset{*}{v}_{t+dt} = \overset{*}{v}_t + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}$$

Note we have to evaluate the force 4 times, and do more calcualtion,
But it's worth it… error is now $\propto h^4$

What you can do with ODEs

- Particle systems… e.g. fluid simulation

Video:
http://www.youtube.com/watch?v=WruTNnF6Ztg&feature=related