# MATHS AND PHYSICS FOR GAME

Vuong Ba Thinh
Department of CS –CSE Faculty - HCMUT

# Outline

- Math – Linear Algebra for Game
  - Vector
  - Matrix
- Physics
  - Collision Detection
  - Movement

# VECTOR

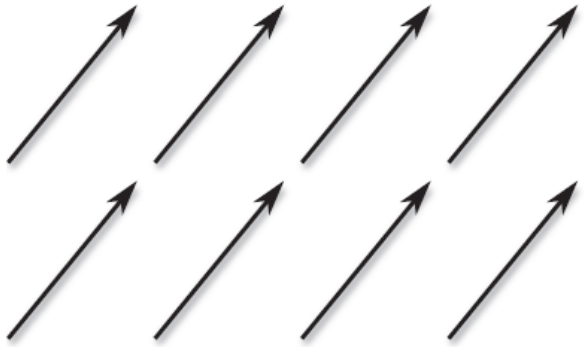
- Representation:

$$\vec{v} = \langle v_x, v_y, v_z \rangle$$

- In code:

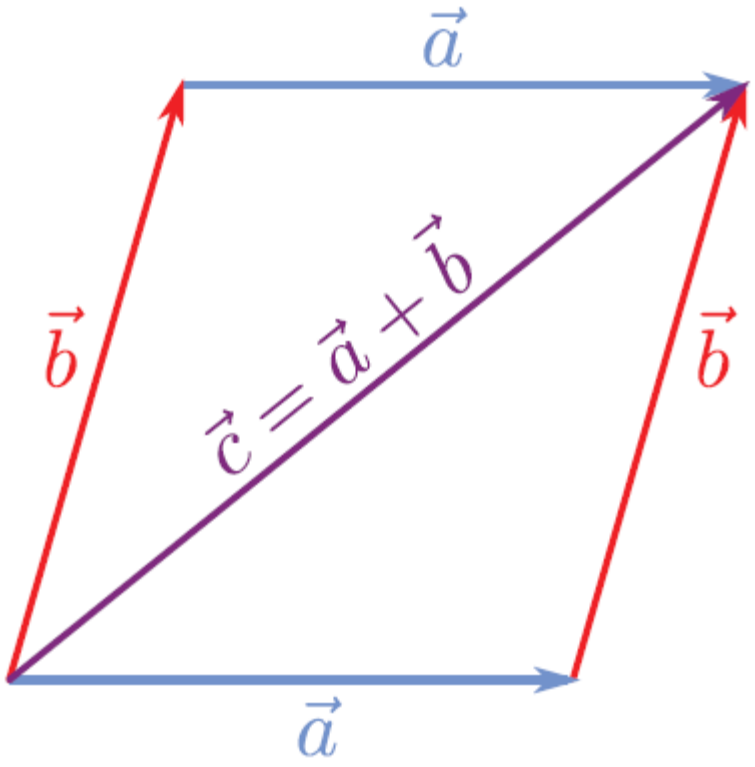
```
class Vector3
    float x
    float y
    float z
end
```

- No concept of position



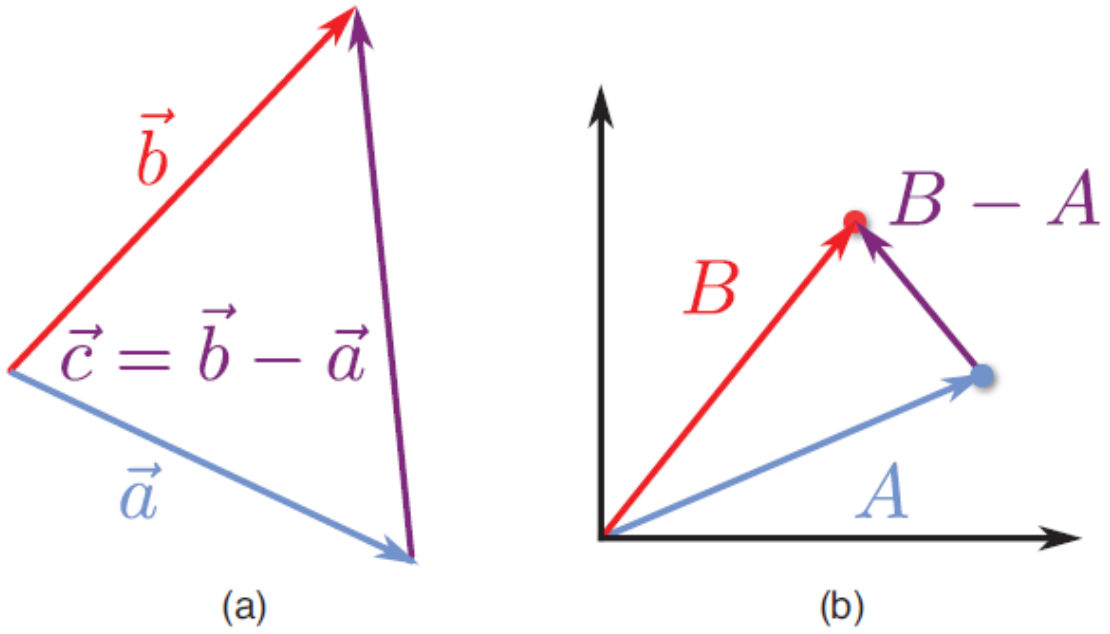
# ADDITION

$$\vec{c} = \vec{a} + \vec{b} = \langle a_x + b_x, a_y + b_y, a_z + b_z \rangle$$



# SUBTRACTION

$$\vec{c} = \vec{b} - \vec{a} = \langle b_x - a_x,\ b_y - a_y,\ b_z - a_z \rangle$$




Vector subtraction (a), and representing points as vectors from the origin (b).

# SUBTRACTION

- Example:  imagine you are designing a UI arrow system for a game

```
arrowVector = objective.position – player.position
```

# LENGTH, UNIT VECTOR, NORMALIZATION

- Length (magnitude)

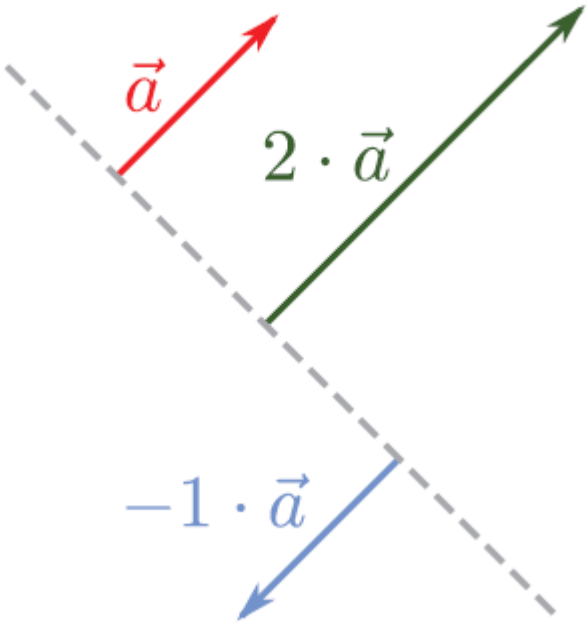
$$\left\|\vec{a}\right\| = \sqrt{a_x{}^2 + a_y{}^2 + a_z{}^2}$$

- Unit Vector:

$$\hat{a} = \left\langle \frac{a_x}{\left\|\vec{a}\right\|}, \frac{a_y}{\left\|\vec{a}\right\|}, \frac{a_z}{\left\|\vec{a}\right\|} \right\rangle$$

# SCALAR MULTIPLICATION

$$s \cdot \vec{a} = \langle s \cdot a_x,\ s \cdot a_y,\ s \cdot a_z \rangle$$



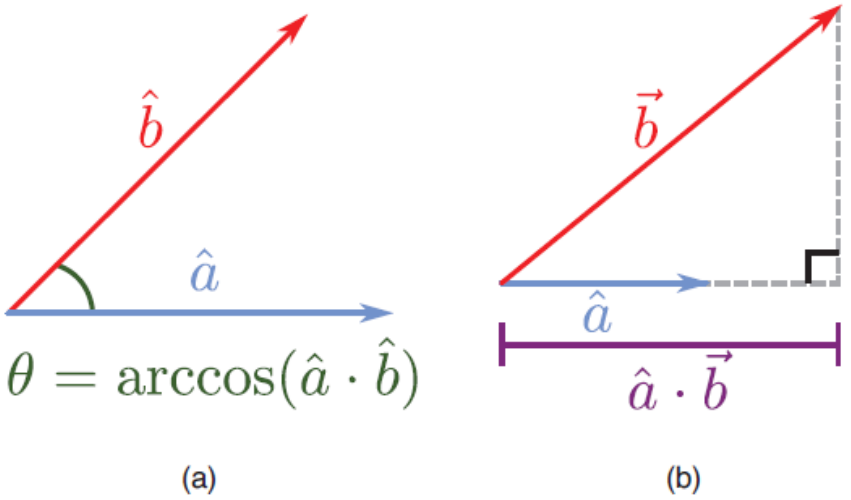
# DOT PRODUCT

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

$$\vec{a} \cdot \vec{b} = \left\|\vec{a}\right\| \left\|\vec{b}\right\| \cos\theta$$

$$\theta = \arccos\left( \frac{\vec{a} \cdot \vec{b}}{\left\|\vec{a}\right\| \left\|\vec{b}\right\|} \right)$$

# DOT PRODUCT



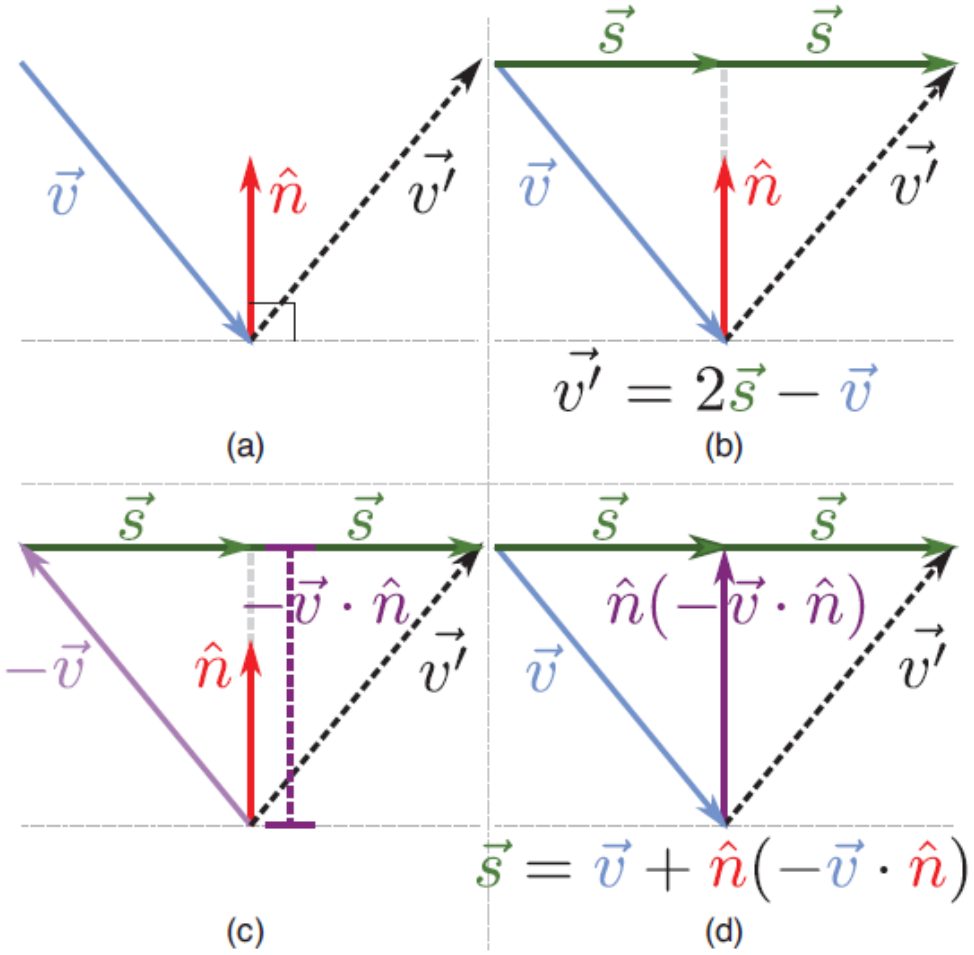

Angle between unit vectors (a), and scalar projection (b).

$$\vec{a} \cdot \vec{b} = \vec{b} \cdot \vec{a}$$

$$\vec{a} \cdot \left( \vec{b} + \vec{c} \right) = \vec{a} \cdot \vec{b} + \vec{a} \cdot \vec{c}$$

$$\vec{a} \cdot \left( \vec{b} \cdot \vec{c} \right) = \left( \vec{a} \cdot \vec{b} \right) \cdot \vec{c}$$

# SAMPLE PROBLEM: VECTOR REFLECTION



# SAMPLE PROBLEM: VECTOR REFLECTION

$$\vec{v}' = 2\vec{s} - \vec{v}$$

$$\vec{s} = \vec{v} + \hat{n}\left(-\vec{v} \cdot \hat{n}\right)$$

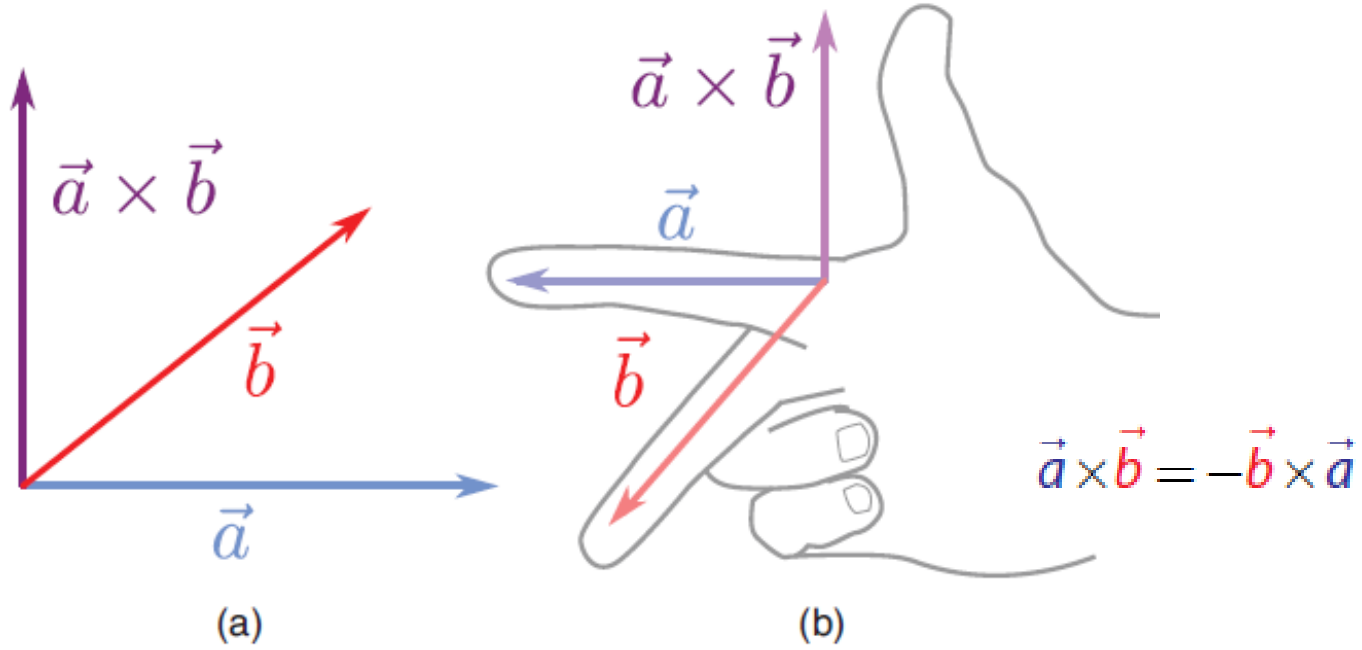
$$\vec{v}' = 2\left(\vec{v} + \hat{n}\left(-\vec{v} \cdot \hat{n}\right)\right) - \vec{v}$$

$$\vec{v}' = 2\vec{v} + 2\hat{n}\left(-\vec{v} \cdot \hat{n}\right) - \vec{v}$$

$$\vec{v}' = \vec{v} - 2\hat{n}\left(\vec{v} \cdot \hat{n}\right)$$

# CROSS PRODUCT

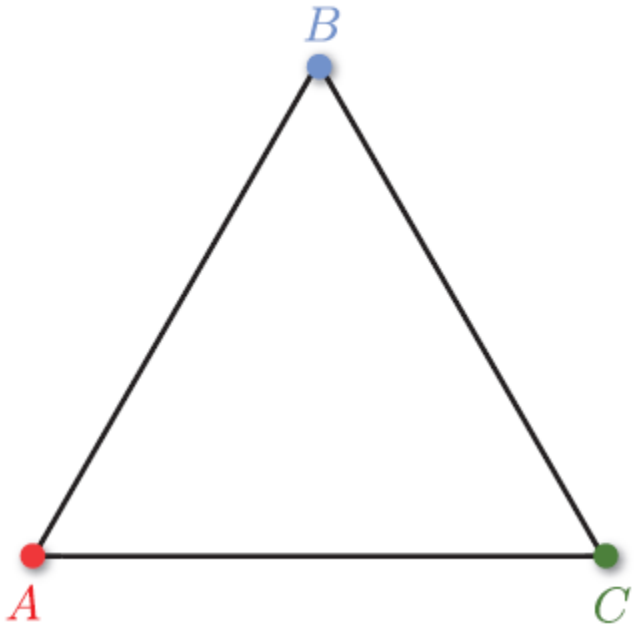
$$\vec{c} = \vec{a} \times \vec{b}$$




Cross product (a), and right-hand rule (b).

# CROSS PRODUCT

$$\vec{c} = \vec{a} \times \vec{b} = \langle a_y b_z - a_z b_y,\ a_z b_x - a_x b_z,\ a_x b_y - a_y b_x \rangle$$



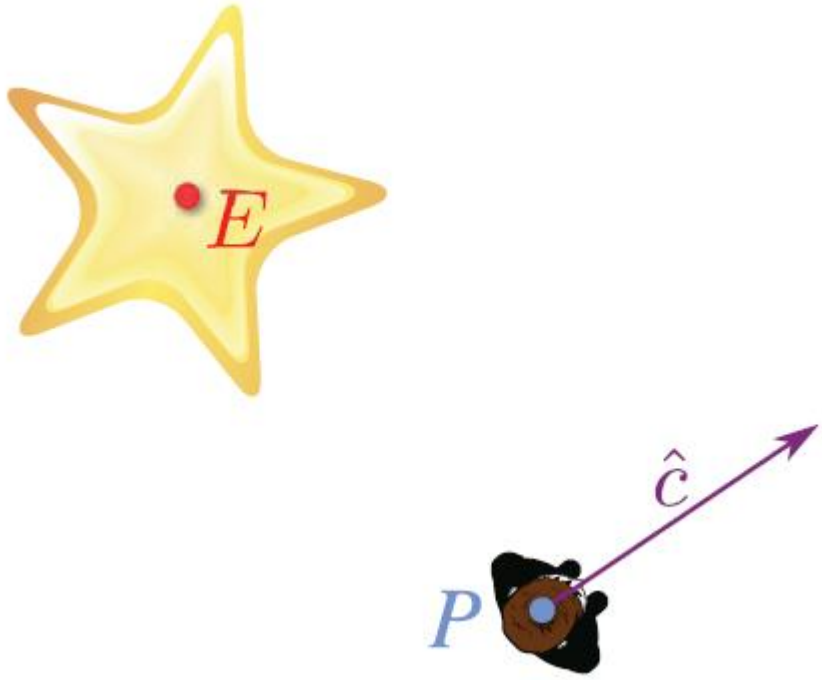

$$\vec{u} = B - A$$

$$\vec{v} = C - A$$

$$\vec{n} = \vec{u} \times \vec{v}$$

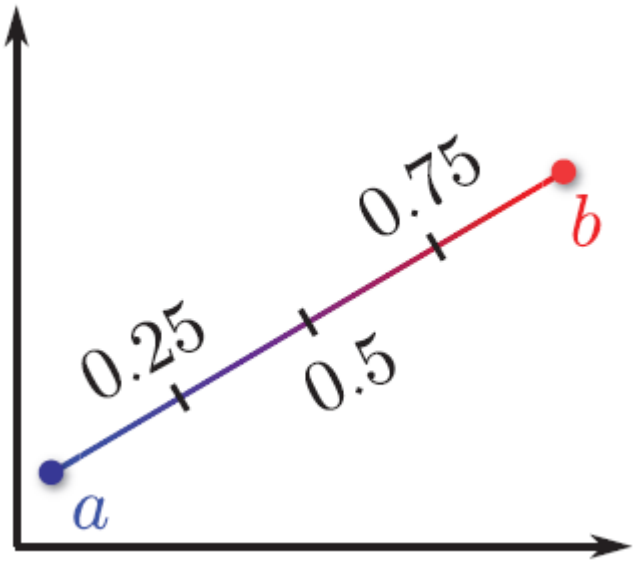
$$\hat{n} = normalize(\vec{n})$$

# SAMPLE PROBLEM: ROTATING A 2D CHARACTER

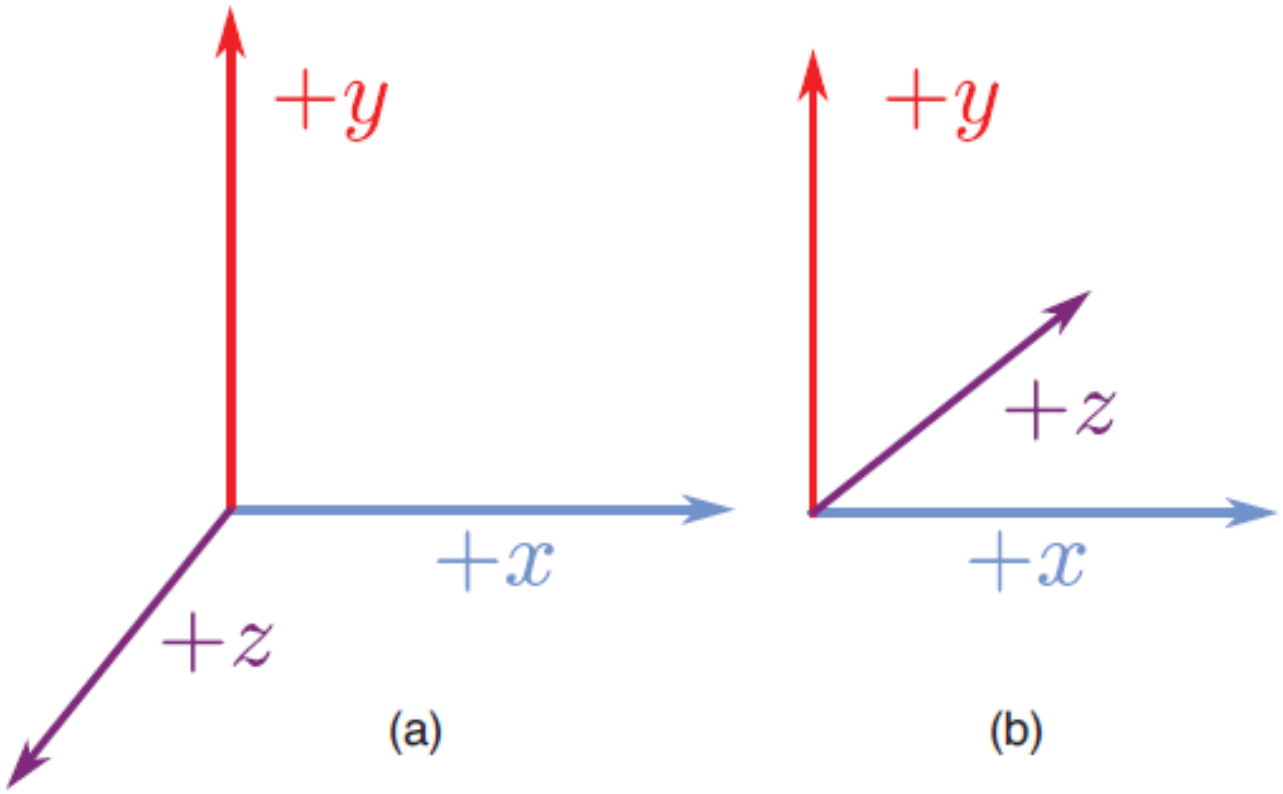


# LINEAR INTERPOLATION

$$Lerp(a,b,f) = (1-f)\cdot a + f\cdot b$$



# COORDINATE SYSTEMS




Sample right-handed (a) and left-handed (b) coordinate systems.

# MATRIX

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

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix}$$

# ADDITION

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} + \begin{bmatrix} j & k & l \\ m & n & o \\ p & q & r \end{bmatrix} = \begin{bmatrix} a+j & b+k & c+l \\ d+m & e+n & f+o \\ g+p & h+q & i+r \end{bmatrix}$$

# SCALAR MULTILICATION

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

# MULTILICATION

$$\begin{bmatrix} C_{1,1} & C_{1,2} & C_{1,3} & C_{1,4} \\ C_{2,1} & C_{2,2} & C_{2,3} & C_{2,4} \\ C_{3,1} & C_{3,2} & C_{3,3} & C_{3,4} \\ C_{4,1} & C_{4,2} & C_{4,3} & C_{4,4} \end{bmatrix} = \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\ A_{2,1} & A_{2,2} & A_{2,3} & A_{2,4} \\ A_{3,1} & A_{3,2} & A_{3,3} & A_{3,4} \\ A_{4,1} & A_{4,2} & A_{4,3} & A_{4,4} \end{bmatrix} \times \begin{bmatrix} B_{1,1} & B_{1,2} & B_{1,3} & B_{1,4} \\ B_{2,1} & B_{2,2} & B_{2,3} & B_{2,4} \\ B_{3,1} & B_{3,2} & B_{3,3} & B_{3,4} \\ B_{4,1} & B_{4,2} & B_{4,3} & B_{4,4} \end{bmatrix}$$

$$C_{1,1} = A_{1,1} \cdot B_{1,1} + A_{1,2} \cdot B_{2,1} + A_{1,3} \cdot B_{3,1} + A_{1,4} \cdot B_{4,1}$$

$$A \times B \neq B \times A$$

$$A \times (B + C) = A \times B + A \times C$$

$$A \times (B \times C) = (A \times B) \times C$$

# INVERSE

$$I = A \times A^{-1}$$

- I: identity matrix

# TRANSPOSE

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

# TRANSFORMING 3D VECTORS BY MATRICES

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

$$x' = x \cdot a + y \cdot b + z \cdot c$$

$$y' = x \cdot d + y \cdot e + z \cdot f$$

...

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

$$x' = a \cdot x + b \cdot y + c \cdot z$$

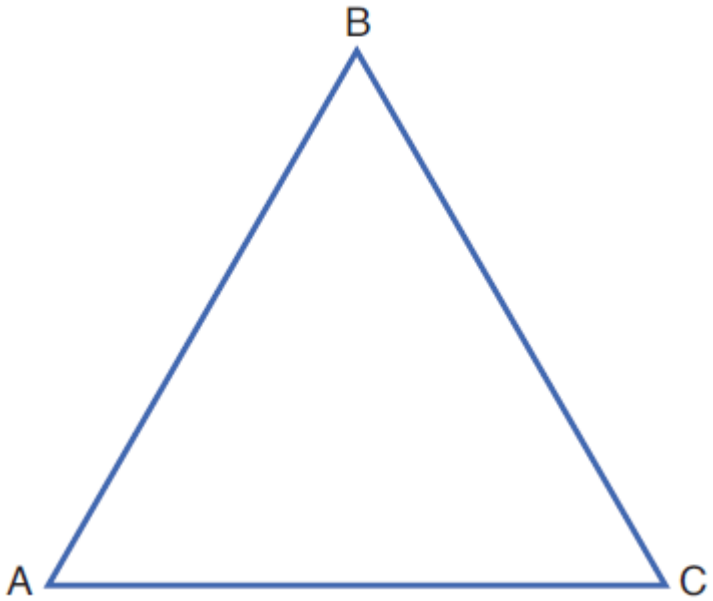
...

# PHYSICS

- Plane
- Ray, and Line Segment
- Collision Geometry
- Physics-based Movement
- Physics Middleware

# PLANE

$$P \cdot \hat{n} + d = 0$$




```
struct Plane
    Vector3 normal
    float d
end
```

# RAYS AND LINE SEGMENTS

$$R(t) = R_0 + \vec{v}t$$

```
struct RayCast
    Vector3 startPoint
    Vector3 endPoint
end
```

# COLLISION GEOMETRY

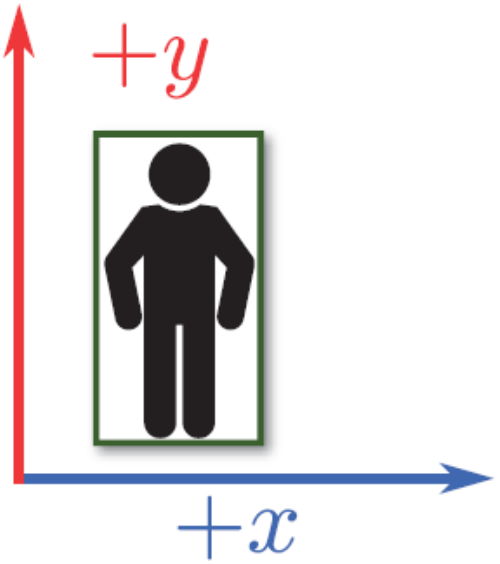
- What is it???
- Why do we use it?

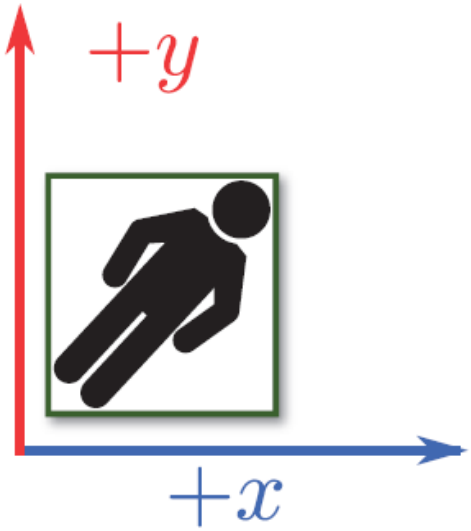
# BOUNDING SPHERE

```
class BoundingSphere
    Vector3 center
    float radius
end
```

# AXIS-ALIGNED BOUNDING BOX

```
class AABB2D
    Vector2 min
    Vector2 max
end
```



# CAPSULE

```
struct Capsule2D
    Vector2 startPoint
    Vector2 endPoint
    float radius
end
```

# CONVEX POLYGONS

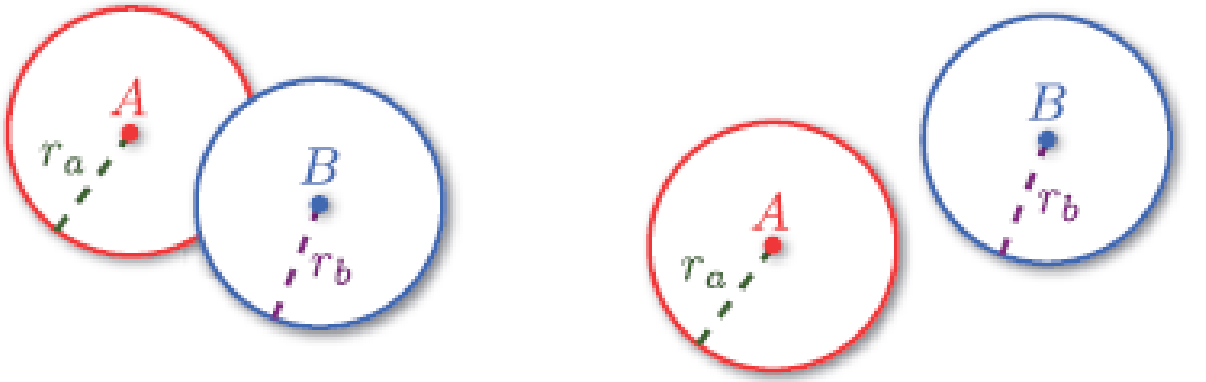
# LIST OF COLLISION GEOMETRIES

- ???

# COLLISION DETECTION

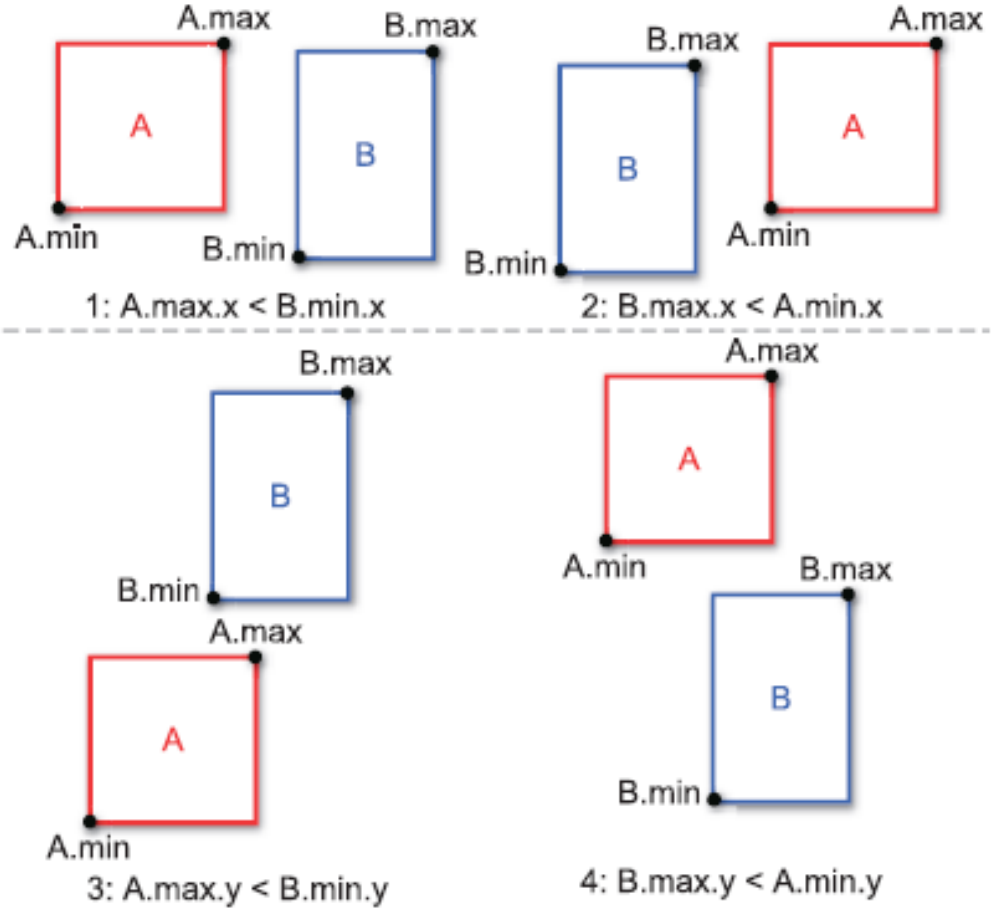
- Sphere versus Sphere Intersection
- AABB versus AABB Intersection
- Line Segment versus Plane Intersection
- Line Segment versus Triangle Intersection
- Sphere versus Plane Intersection
- Swept Sphere Intersection
- Collision Response
- Optimizing Collisions

# SPHERE VERSUS SPHERE INTERSECTION




Two spheres intersect (a) and do not intersect (b).

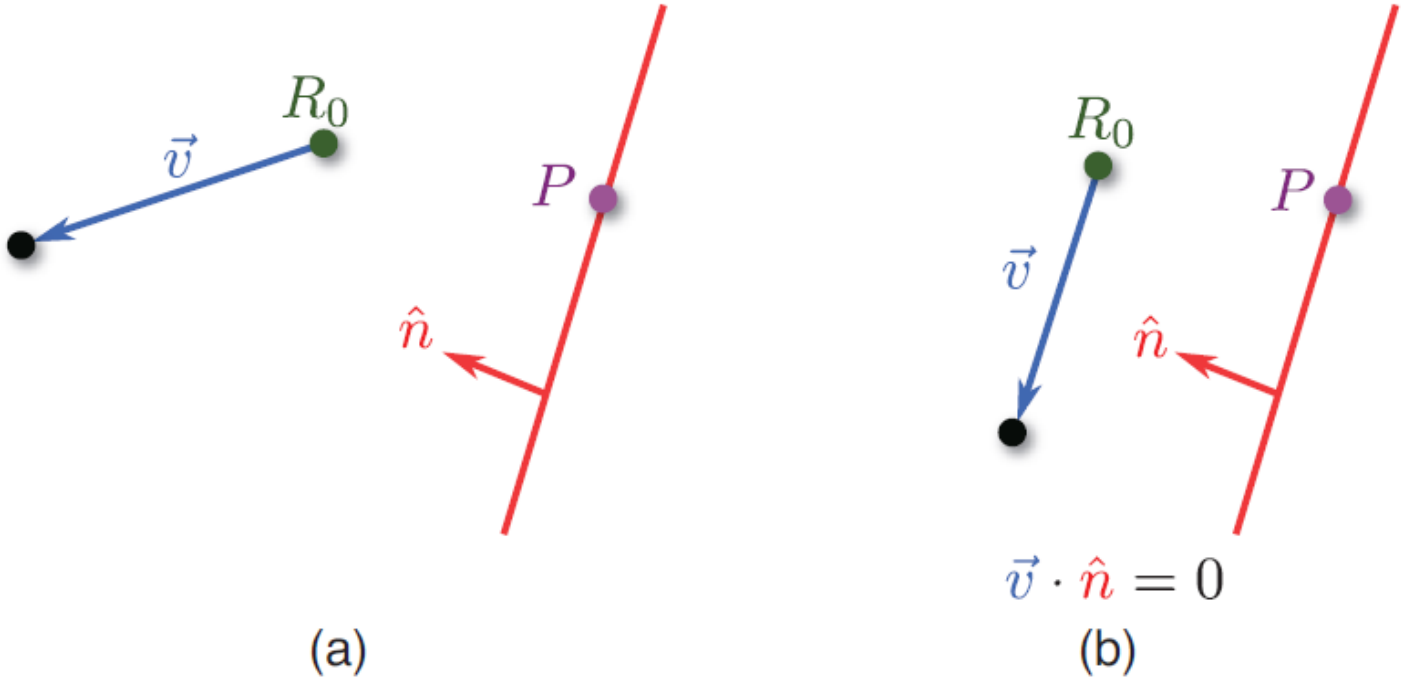
# AABB VERSUS AABB INTERSECTION



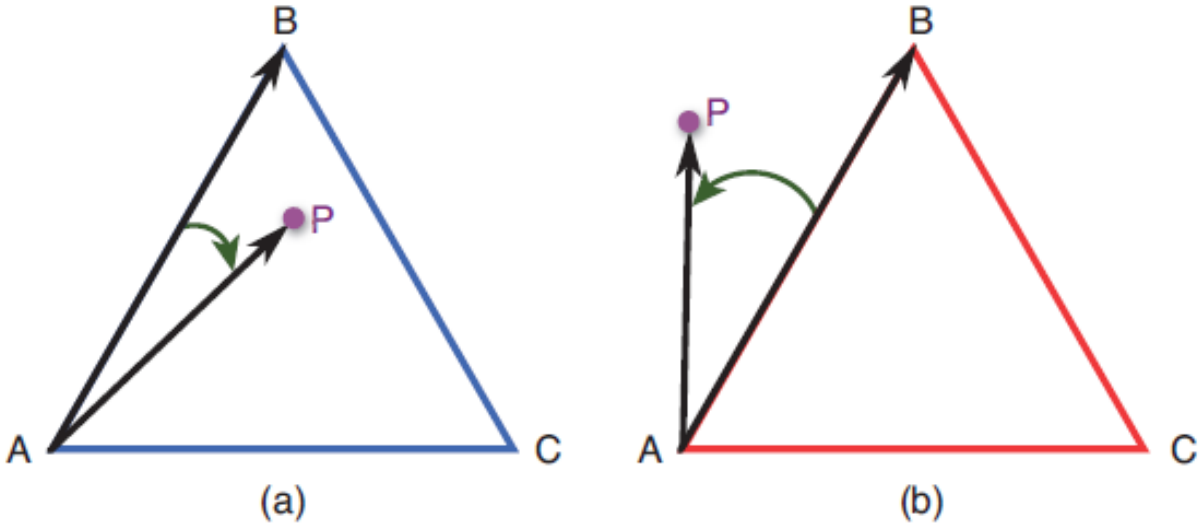
# LINE SEGMENT VERSUS PLANE INTERSECTION

$$R(t) = R_0 + \vec{v}t$$

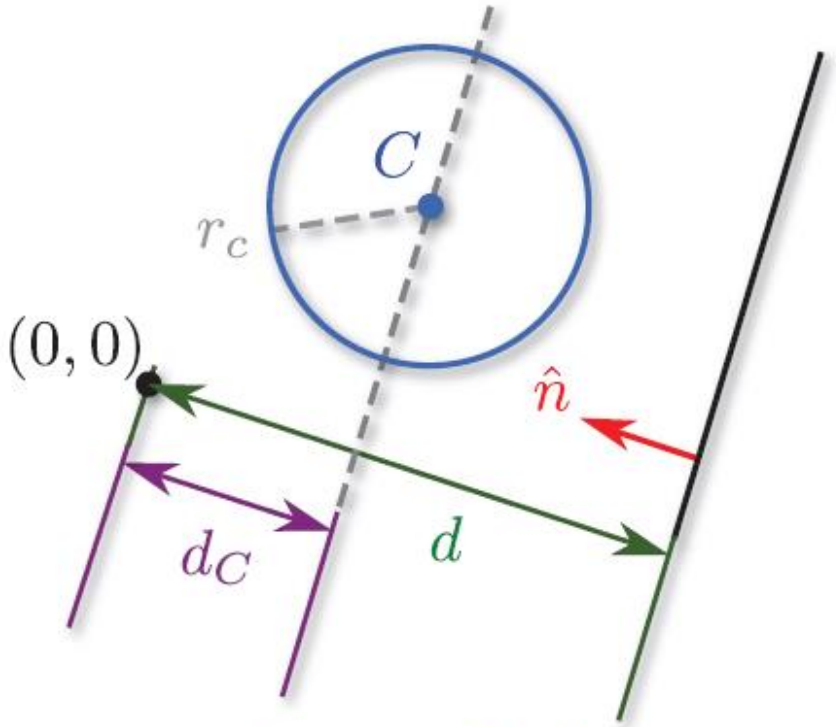
$$P \cdot \hat{n} + d = 0$$



# LINE SEGMENT VERSUS TRIANGLE INTERSECTION



# SPHERE VERSUS PLANE INTERSECTION

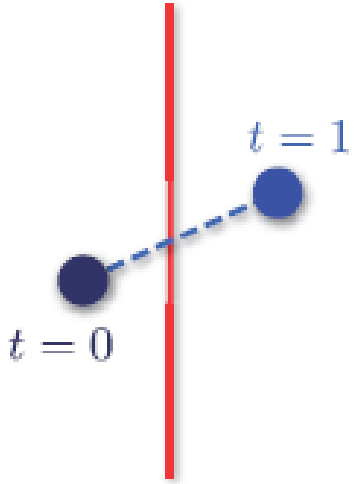



$$d_C = -C \cdot \hat{n}$$

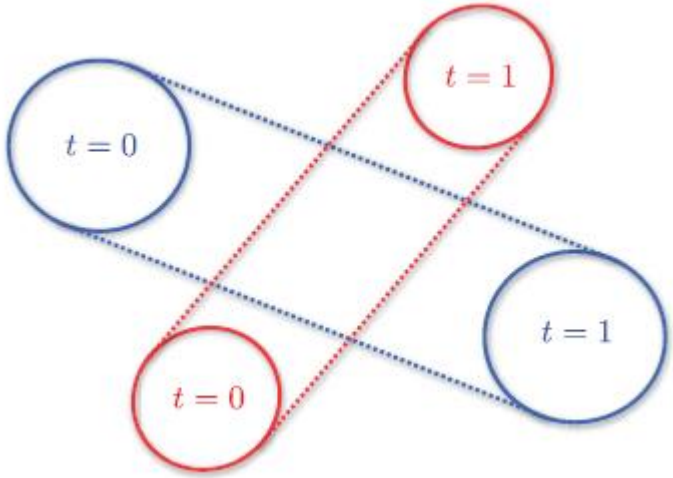
$$|d - d_C| > r_C$$

# Swept Sphere Intersection

- Bullet through paper problem

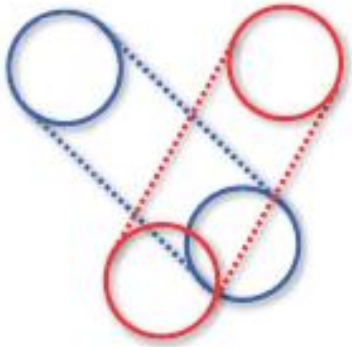



- Swept Sphere Intersection



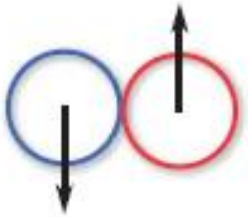
$b^2 - 4ac < 0$

No Intersection



$b^2 - 4ac > 0$

Full Intersection



$b^2 - 4ac = 0$

# COLLISION RESPONSE

- Read chapter 7 [1]

# OPTIMIZING COLLISIONS

- Read chapter 7 [1]

# PHYSICS-BASED MOVEMENT

- Overview

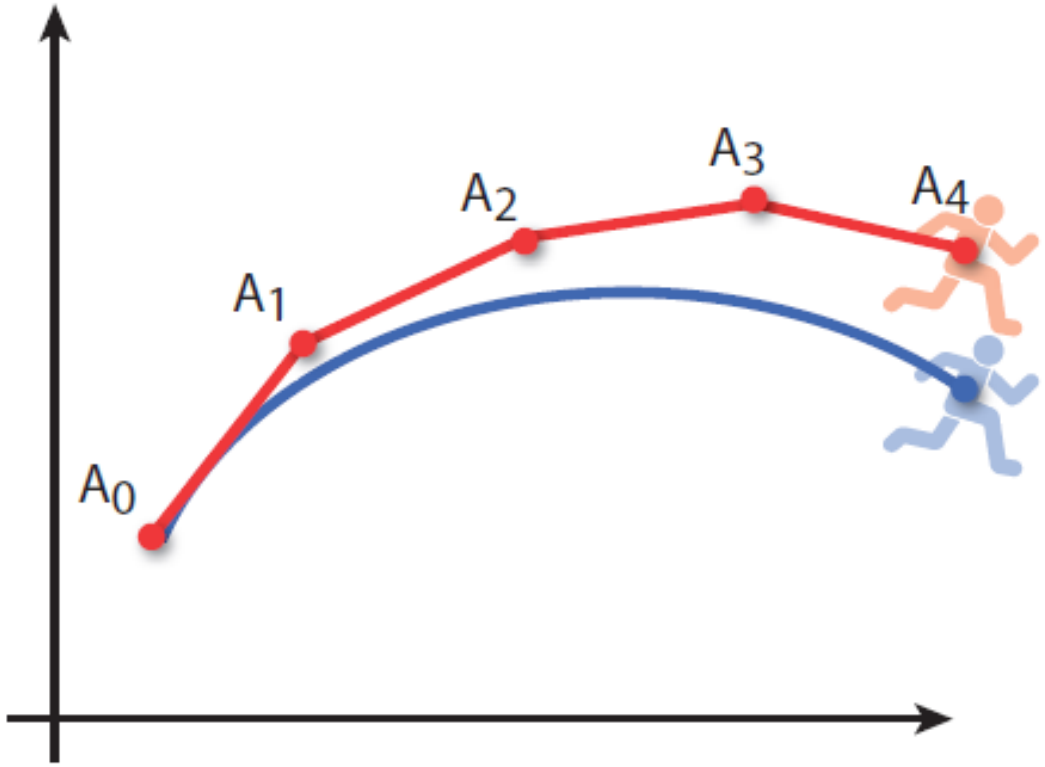
$$F = m \cdot a$$

$$v(t) = \frac{dr}{dt}$$

$$a(t) = \frac{dv}{dt}$$

- Problem???

# ISSUES WITH VARIABLE TIME STEPS



# EULER AND SEMI-IMPLICIT EULER INTEGRATION

```
class PhysicsObject
   // List of all the force vectors active on this object
   List forces
   Vector3 acceleration, velocity, position
   float mass

    function Update(float deltaTime)
       Vector3 sumOfForces = sum of forces in forces
       acceleration = sumOfForces / mass

       // Euler Integration
       position += velocity * deltaTime
       velocity += acceleration * deltaTime
    end
end
```

# VELOCITY VERLET INTEGRATION

```
function Update(float deltaTime)
   Vector3 sumOfForces = sum of forces in forces

   // Velocity Verlet Integration
   Vector3 avgVelocity = velocity + acceleration * deltaTime / 2.0f
   // Position is integrated with the average velocity
   position += avgVelocity * deltaTime
   // Calculate new acceleration and velocity
   acceleration = sumOfForces / mass
   velocity = avgVelocity + acceleration * deltaTime / 2.0f
end
```

# PHYSICS MIDDLEWARE

- 3D: PhysX
- 2D: Box2D

# MORE READING


- Lengyel, Eric. **Mathematics for 3D Game Programming and Computer Graphics** (Third Edition) . Boston: Course Technology, 2012
- Ericson, Christer. **Real-time Collision Detection** . San Francisco: Morgan Kaufmann, 2005
- Fielder, Glenn. Gaffer on Games – **Game Physics**. http://gafferongames.com/game-physics/