Game Essentials

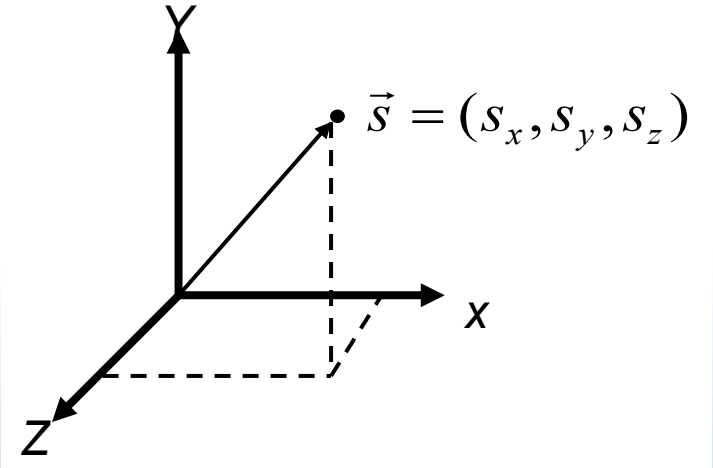
Vectors and vector operations

Game Essentials

- Scalar value
  - Single numbers
- Vectors
  - An element to represent physical quantities that have both magnitude and direction
    - Can be used to represent a position in space, a direction, a translation, a velocity, a force
    - For games, typical vector spaces of interest are regular 2D or 3D spaces
  - **Note**: Many programming languages have a vector data structure, typically an array of values, that may differ from the mathematical concept

Game Essentials

- Vectors
  - Can be used to represent a position in a vector space
    - Sometimes called a position vector
    - Specifies a unique position in 3D vector space



Game Essentials

- Unity
  - The Unity API defines a number of vector classes
    - `Vector2`, `Vector3`, `Vector4`
  - Create a vector using a constructor

```
Vector3 vec = new Vector3(2, 8, 0);
```

  - Can access a vector's x, y, z components

```
float x = vec.x;
```

  - Shorthand for (0, 0, 0)

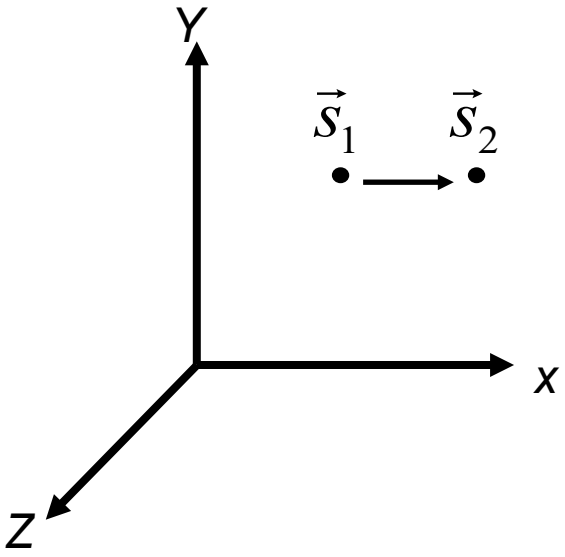
```
Vector3 vec = Vector3.zero;
```

  - Setting a GameObject's position

```
transform.position = new Vector3(0, 1, 0);
```

Game Essentials

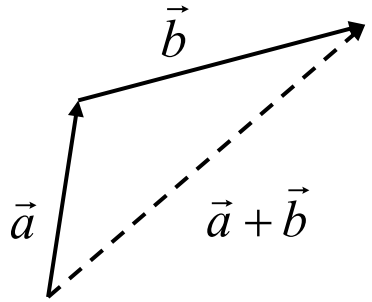
- Vectors
  - Can also represent displacement (or translation)
    - A change from one position to another
    - A position vector is a change of position from the origin to the target location



$$\begin{aligned}\Delta\vec{s} &= \vec{s}_2 - \vec{s}_1 \\ &= (\Delta s_x, \Delta s_y, \Delta s_z) \\ &= (s_{2x} - s_{1x}, s_{2y} - s_{1y}, s_{2z} - s_{1z})\end{aligned}$$

Game Essentials

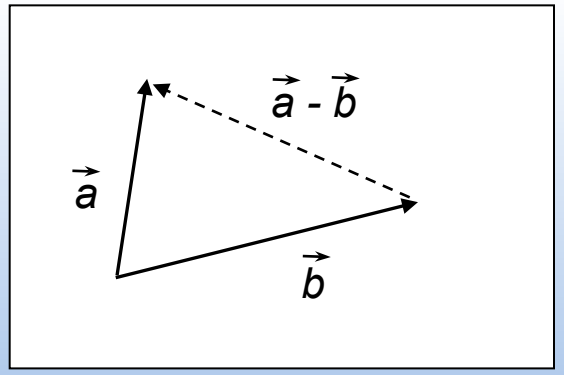
- Vector operations
  - Addition and subtraction
    - Geometrically, adding two vectors together is equivalent to placing them end to end

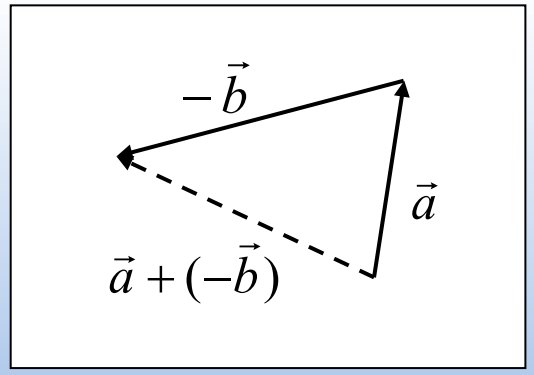


$$\vec{a}+\vec{b}=\left(a_x+b_x,a_y+b_y,a_z+b_z\right)$$

$$\vec{a}-\vec{b}=\left(a_x-b_x,a_y-b_y,a_z-b_z\right)=\vec{a}+\left(-\vec{b}\right)$$

```
Vector3 vec = vecA + vecB;
```





Game Essentials

- Vectors
  - A vector can be split into two elements $\vec{u} = |\vec{u}|\hat{\vec{u}}$
    1. Magnitude (i.e. the length of the vector)

$$|\vec{u}| = \sqrt{u_x{}^2 + u_y{}^2 + u_z{}^2}$$

```
float length = vec.magnitude;
```

    2. Direction
       - Vector of unit length, sometimes called the 'normalized' or 'unit-normal vector'
         - » Normalizing a vector $\hat{\vec{u}} = \frac{\vec{u}}{|\vec{u}|}$

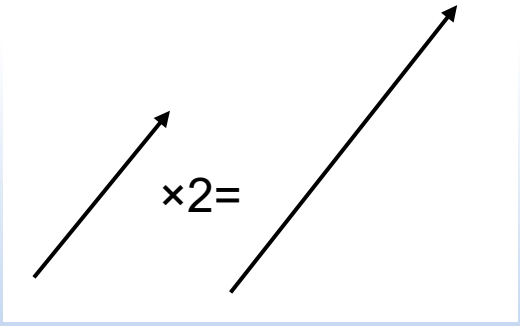
```
Vector3 unitVector = vec.normalized;
Vector3 unitVector = Vector3.Normalize(vec);
```

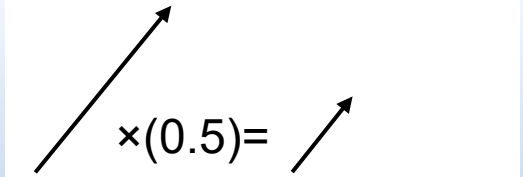
Game Essentials

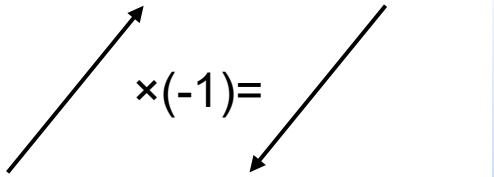
- Vector operations
  - Scalar-vector multiplication
    - Multiply all components of the vector by the scalar

$$k\vec{u} = k(u_x, u_y, u_z) = (ku_x, ku_y, ku_z)$$

```
Vector3 scaledVec = 2 * unitVector;
```

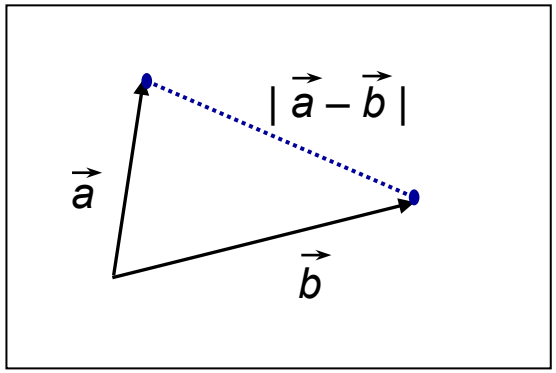






Game Essentials

- Vector operations
  - Distance between two vectors
    - Defined as the distance between their end points



$$d(\vec{a},\vec{b}) = |\vec{a} - \vec{b}| = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2 + (a_z - b_z)^2}$$

```
float distance = Vector3.Distance(vecA, vecB);
```

Game Essentials

- Vector operations
  - The dot product
    - Or the 'scalar product' – results in a single scalar value

$$\vec{a} \bullet \vec{b} = (a_x, a_y, a_z) \bullet (b_x, b_y, b_z) = (a_x b_x + a_y b_y + a_z b_z)$$

```
float dotProduct = Vector3.Dot(vecA, vecB);
```

    - Trigonometry of the dot product

$$\vec{a} \bullet \vec{b} = |\vec{a}||\vec{b}|\cos\theta$$

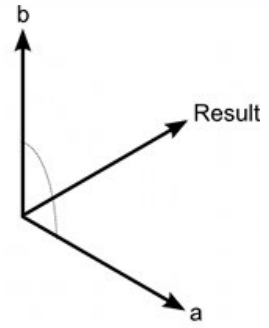
```
float angle = Vector3.Angle(vecA, vecB);
```

Example: orthogonal vectors

$\vec{a}\ \vec{b} = |\vec{a}||\vec{b}|\cos 90^\circ = 0$

Game Essentials

- Vector operations
  - The cross product
    - Results in a third vector which is perpendicular to the two input vectors
      - Its direction is based on the handedness of the coordinate system



$$\vec{a} \times \vec{b} = (a_x, a_y, a_z) \times (b_x, b_y, b_z) = (a_y b_z - a_z b_y, a_z b_x - a_x b_z, a_x b_y - a_y b_x)$$

```
Vector3 result = Vector3.Cross(vecA, vecB);
```

- Not commutative

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

Game Essentials

- Unity
  - ➢ Useful shorthands of the `Vector3` class

`Vector3.forward` – shorthand for `Vector3(0, 0, 1)`

`Vector3.right` – shorthand for `Vector3(1, 0, 0)`

`Vector3.up` – shorthand for `Vector3(0, 1, 0)`

`Vector3.zero` – shorthand for `Vector3(0, 0, 0)`

  - ➢ A GameObject's transform properties
    - Returns a normalized vector representing the respective axes of the transform in world space
      - Considers the rotation of the GameObject

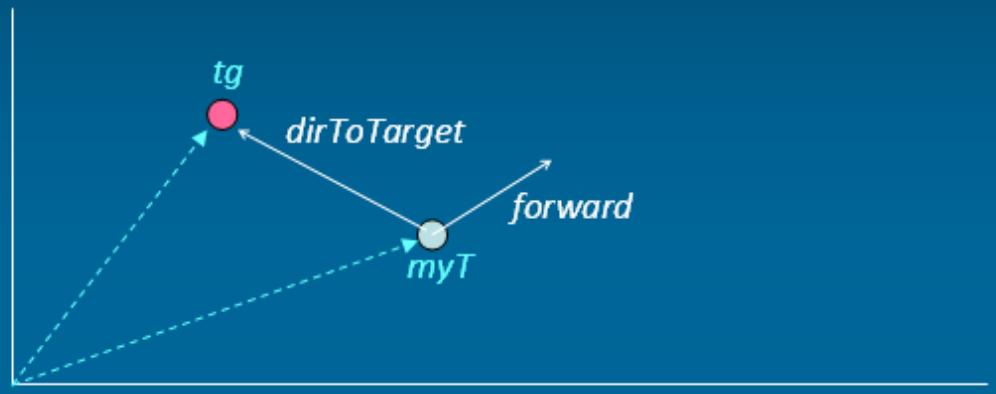
`transform.forward` – blue axis

`transform.right` – red axis

`transform.up` – green axis

Quiz

Your tank `myT` is moving forward. Your target is `tg`. How to calculate if the tank is moving away from the target.



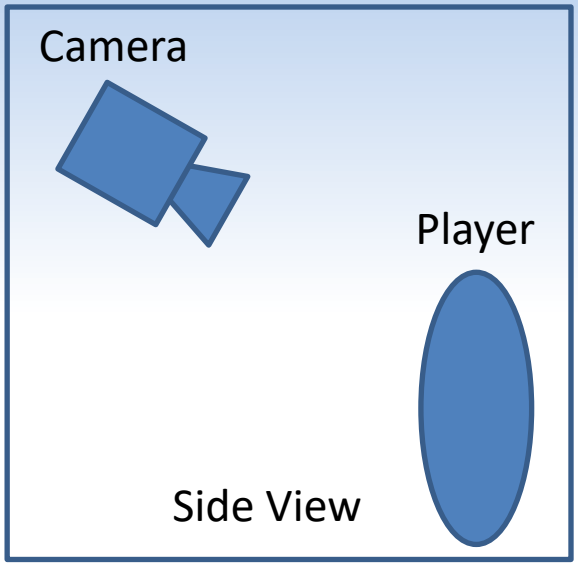
Solution 1:

```
Vector3 dirToTarget = tg.transform.position - transform.position;
float angle = Vector3.Angle( dirToTarget, transform.forward );
if( angle > 90 ) { // moving away }
```

Solution 2:

```
Vector3 dirToTarget = tg.transform.position - transform.position;
if( Vector3.Dot( dirToTarget, transform.forward ) < 0 )
  { // moving away }
```

Quiz



- Given a third-person camera
  - How do we move the player forward in the direction of the camera?

Solution 1:

```
Vector3 right = camera.transform.right;
Vector3 forward = Vector3.Cross(right, Vector3.up);
movement = forward * moveSpeed * Time.DeltaTime;
```

Solution 2:

```
Vector3 forward = camera.transform.forward;
forward.y = 0;
forward = Vector3.Normalize(forward);
movement = forward * moveSpeed * Time.DeltaTime;
```

References

- Among others, material sourced from
  - https://unity.com/
  - https://docs.unity3d.com
  - Jason Gregory, Game Engine Architecture, A.K. Peters