ETC Programmer Student Workshop 1

Basic Linear Algebra for Game Programmers

Topics Covered Today

- Quick Recap of Vectors
- Dot Product
- Cross Product
- Vector Projection

Focus of Today

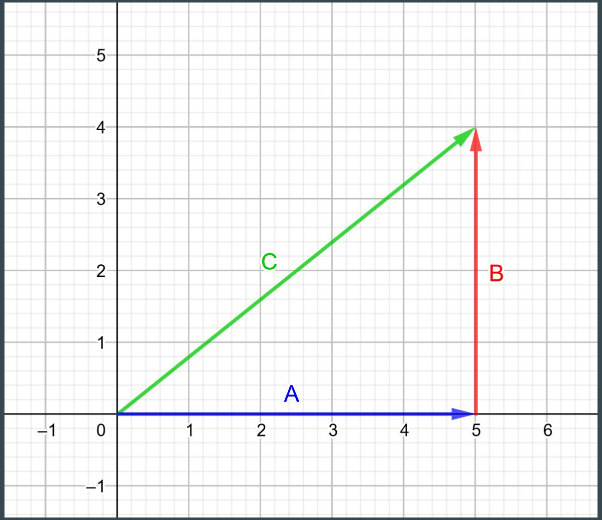
- How can we apply linear algebra in our games?
- How is it useful to us as programmers?
- What are some scenarios where we can apply these linear algebra concepts?

A Quick Recap of Vectors

- A quantity that has both direction and magnitude (length).
- Used to represent many things such as direction, velocity, forces etc.
- A vector of length 1 is known as a *unit vector*.
- In games, we generally use 2D and 3D vectors.
  - $V_{2D} = (x, y)$
  - $V_{3D} = (x, y, z)$

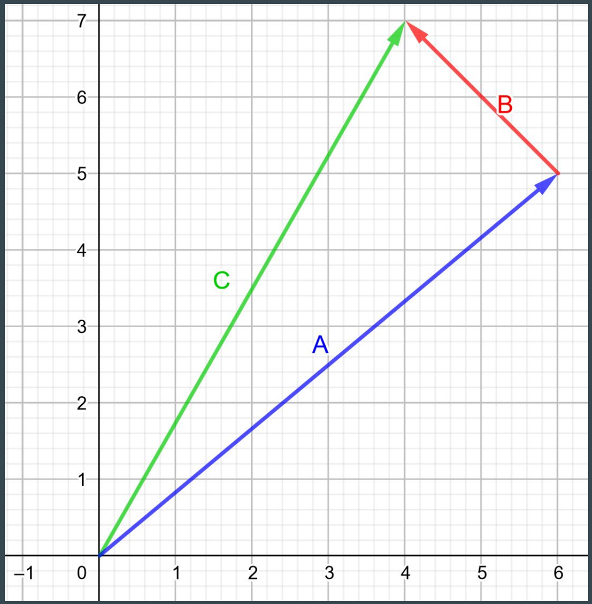
Vector Addition (Tip-to-Tail)

- Vectors can be summed using the tip-to-tail method.
- Example 1:
  - A = (5, 0)
  - B = (0, 4)
  - C = A + B = (5, 4)



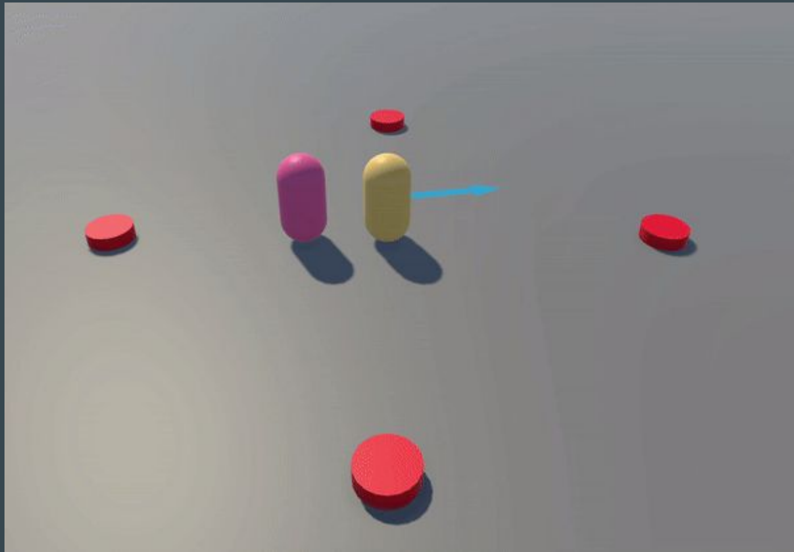
Vector Addition (Tip-to-Tail)

- Vectors can be summed using the tip-to-tail method.
- Example 2:
  - A = (6, 5)
  - B = (-2, 2)
  - C = A + B = (4, 7)



Dot Product - Example 1 (Horror Game)

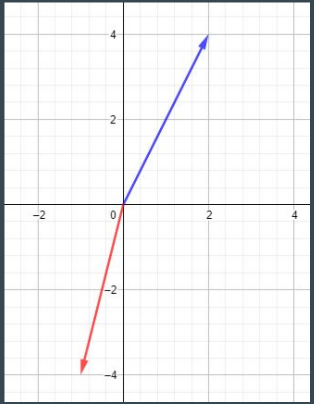
- Horror Game
  - Player trapped in a room with multiple monster spawners.
  - We only want to spawn monsters BEHIND the player to maximise the scariness.
  - Use dot product to check if spawner is behind player.



Dot Product - What is it?

- What is the intuition behind it?
  - Tells us how much 2 vectors point in the same direction.
- Formula
  - Let $A = (x_A, y_A, z_A)$, $B = (x_B, y_B, z_B)$
  - Then, $A \cdot B = (X_A)(X_B) + (Y_A)(Y_B) + (Z_A)(Z_B)$
- Useful tips:
  - If the angle between 2 vectors are less than 90°, the dot product is POSITIVE.
  - If the angle between 2 vectors are more than 90°, the dot product is NEGATIVE.
  - If the angle between 2 vectors are exactly 90°, the dot product is ZERO.

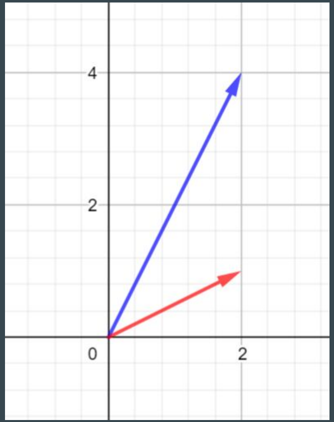
Dot Product - Positivity and Negativity



Angle > 90°:

- $(2, 4) \cdot (-1, -4)$
- $= (2)(-1) + (4)(-4)$
- $= -18$

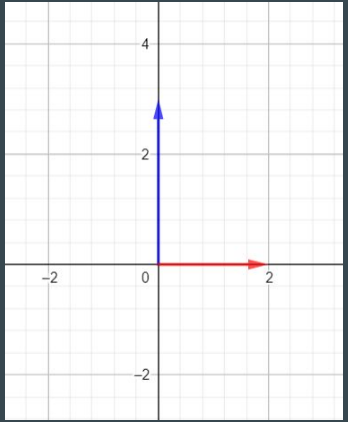
$\therefore (2, 4) \cdot (-1, -4) < 0$



Angle < 90°:

- $(2, 4) \cdot (2, 1)$
- $= (2)(2) + (4)(1)$
- $= 8$

$\therefore (2, 4) \cdot (2, 1) > 0$



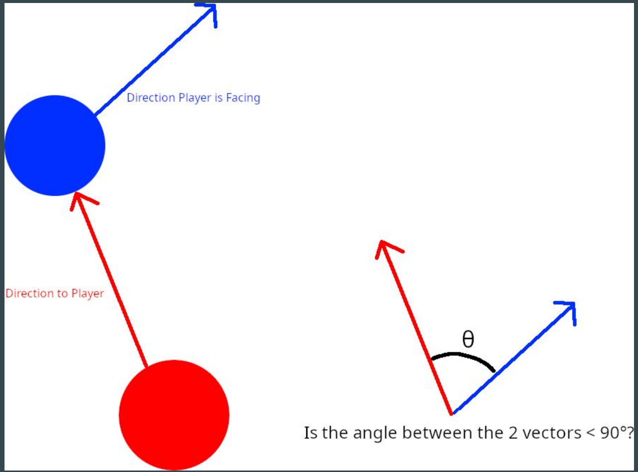
Angle = 90°:

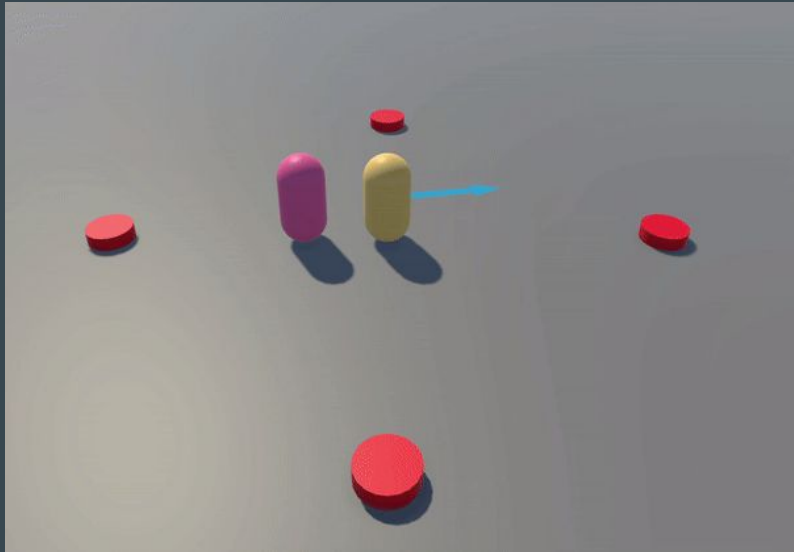
- $(0, 3) \cdot (2, 0)$
- $= (0)(2) + (3)(0)$
- $= 0$

$\therefore (0, 3) \cdot (2, 0) = 0$

Code Snippet

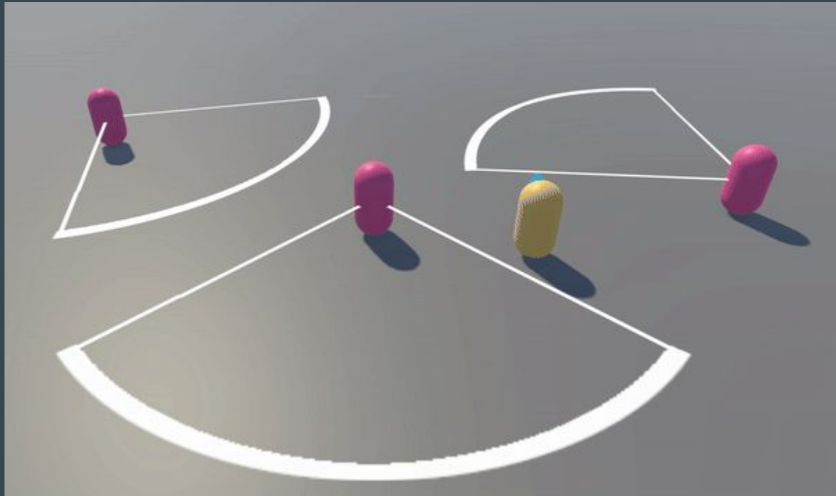
```
// Code Snippet (Simplified for presentation purposes.)
0 references
private void SpawnMonster() {
    // What is the direction from the spawner to the player?
    Vector3 directionToPlayer = player.transform.position - transform.position;
    // Where is the player facing?
    Vector3 playerForwardDirection = player.transform.forward;
    /* If the player is facing the in general direction of the direction from the
     * spawner to it, then the spawner is behind the player. */
    bool isBehindPlayer = Vector3.Dot(directionToPlayer, playerForwardDirection) > 0.0f;

    if (isBehindPlayer)
        // Spawn monster...
}
```





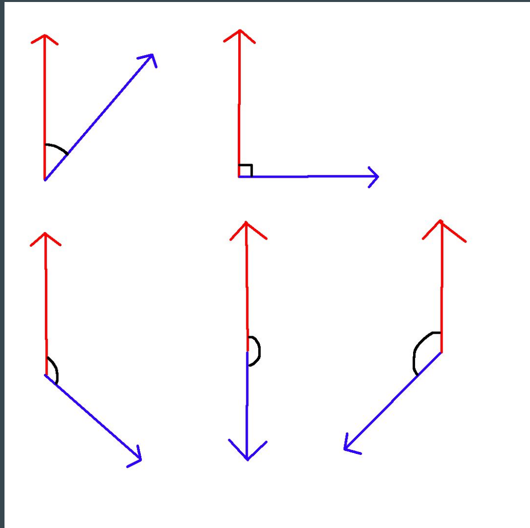
Dot Product - Example 2 (Stealth Game)

- Stealth Game
  - Player is sneaking past enemy.
  - Enemy's field of view is 90°.
  - Enemy detects player if player is within the enemy's field of view.
  - Use dot product to check if the player is inside the enemy's field of view.



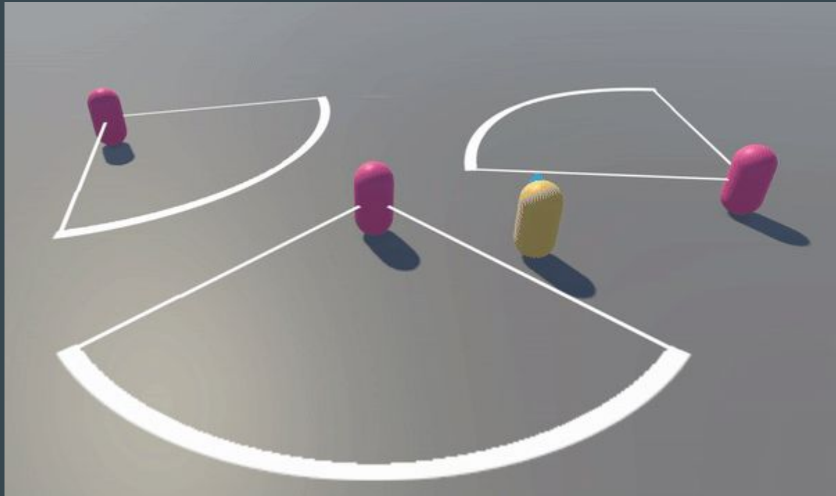
Dot Product - Angle Between 2 Vectors

- Recall the formula to find the smallest angle between 2 vectors.
    - $A \cdot B = |A||B|\cos\theta$
    - $\cos\theta = (A \cdot B) / (|A||B|)$
    - $\theta = \mathrm{acos}[(A \cdot B) / (|A||B|)]$, **where $0 \leq \theta \leq \pi$**
- The above formula is further simplified if A and B are unit vectors.
    - If A and B are unit vectors, $|A||B| = 1$, then $\theta = \mathrm{acos}(A \cdot B)$.
- This formula only gives you the *smallest angle* between 2 vectors!
    - θ is never negative!
    - θ doesn't give you the < 0 and $> \pi$ range!



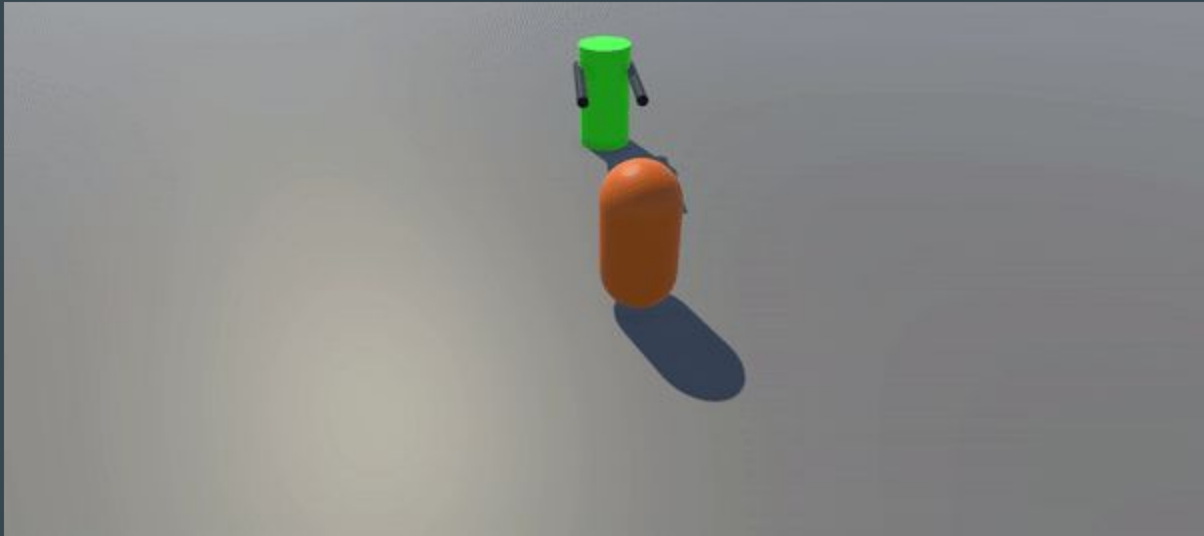
Code Snippet

```
private bool DetectPlayer() {
    // Get the direction and square distance to the player.
    Vector3 directionToPlayer = (player.transform.position - transform.position);
    float sqrDistanceToPlayer = directionToPlayer.sqrMagnitude;

    // If the player is beyond our detection range, it is not detected.
    // Note: Think about why we choose to use the square distance instead of the distance.
    if (detectionRange * detectionRange < sqrDistanceToPlayer)
        return false;

    // Where is the enemy facing?
    Vector3 enemyForward = transform.forward;

    // Get the angle to the player. Remember to convert it from radians to degrees.
    float angleToPlayer = Mathf.Acos(Vector3.Dot(enemyForward, directionToPlayer.normalized)) * Mathf.Rad2Deg;
    float fieldOfView = 90.0f;

    // Remember to half the FOV, since the FOV is the whole arc from the left to the right.
    return angleToPlayer < (fieldOfView * 0.5f);
}
```



Example 3 - Tower Defence (Mini Exercise)

- Tower Defence
  - You are in-charge of programming the turret in a tower defense game.
  - You need to rotate the turret to face the enemy.
  - You already know how to calculate how many degrees to rotate your turret.
    - But how do you know to rotate clockwise or anti-clockwise?

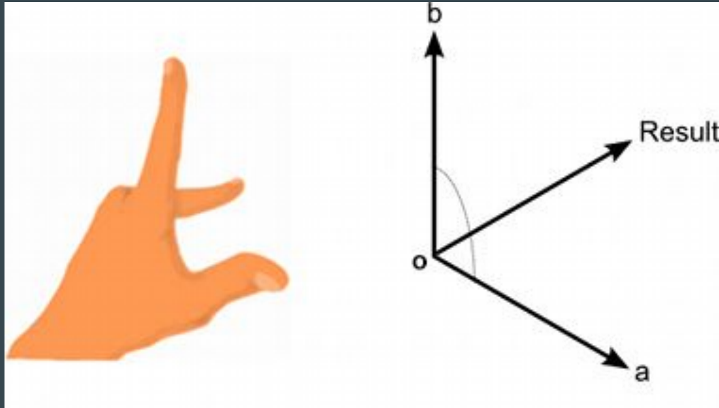


Example 3 - Tower Defence (Mini Exercise)

- Take a minute to think about it!
- Try to recall your undergrad linear algebra!
- Hint:
  - It’s written in the contents section of this presentation. 😛
  - All examples here will be uploaded to GitHub for future reference.
    - https://github.com/TypeDefinition/ETC-Programmer-Workshops

Cross Product - What is it?

- What is the intuition behind it?
  - Takes in 2 vectors and spits out a vector orthogonal to the input vectors.
  - Length of resultant vector is equivalent to area of parallelogram form by the 2 input vectors.
  - $A \times B = -(B \times A)$
    - *Cross product is anticommutative! Order of operands matter!*
- Formula
  - Let $A = (x_A, y_A, z_A)$, $B = (x_B, y_B, z_B)$
  - Then, $A \times B = |A||B|\sin\theta$
  - And, $A \times B = (y_A z_B - z_A y_B, z_A x_B - x_A z_B, x_A y_B - y_A x_B)$
- Notes:
  - Unity uses a left-hand coordinate system.
  - Stretch your fingers as shown in the diagram on the right.
  - Cross product of your thumb and index finger is your middle finger.
  - Cross product of your index finger and your thumb is the opposite of your middle finger.

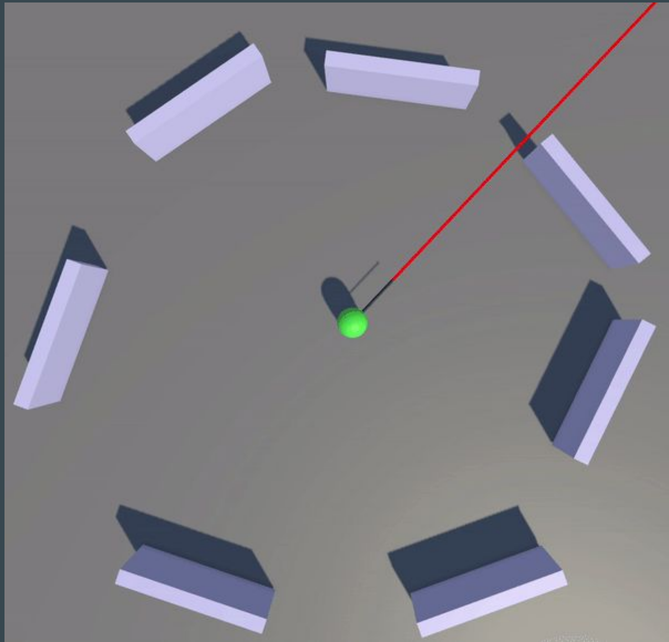


Code Snippet

```
// Code Snippet
1 reference
float GetRotationAngle() {
    // Find our forward direction.
    Vector3 turretForward = new Vector3(transform.forward.x, 0.0f, transform.forward.z).normalized;

    // Find the direction from the turret to the monster.
    Vector3 directionToMonster = monster.transform.position - transform.position;
    directionToMonster.y = 0.0f; // We only want the horizontal direction. Ignore any verticality.
    directionToMonster.Normalize(); // Normalize the direction since we do not need the magnitude.

    // Calculate the angle we need to rotate the turret to face the monster.
    float dotProduct = Vector3.Dot(turretForward, directionToMonster);
    /* Note: Due to floating point precision error, it is possible for the dot product to be greater
     *       than 1 even though the vectors are normalised.
     *       Since the valid range of inputs for acos is [-1, 1], limit dotProduct to be a maximum of 1. */
    float angleToMonster = Mathf.Acos(Mathf.Min(dotProduct, 1.0f)) * Mathf.Rad2Deg;

    // Use the cross product to determine if monster is clockwise or anti-clockwise from where the turret is facing.
    Vector3 crossProduct = Vector3.Cross(turretForward, directionToMonster);

    /* Note that Unity's coordinate system is left-handed.
     * Thus, if the cross product is pointing up, then we need to rotate clockwise. Otherwise, rotate anti-clockwise.
     * In a left-handed coordinate system, a positive rotation on the y-axis rotates clockwise. */
    return (crossProduct.y > 0.0f) ? angleToMonster : -angleToMonster;
}
```

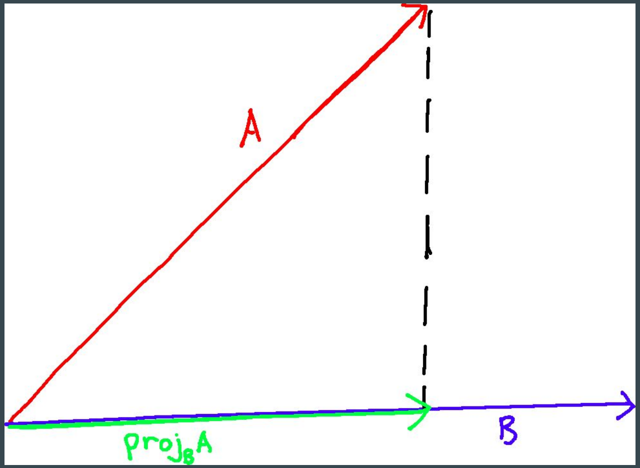
Vector Projection - Example 4 (Laser)

- Laser
  - You are making a platformer game.
  - One of the obstacles in the game is a laser which bounces off the walls.
  - Use vector projection to calculate the laser bounce direction.



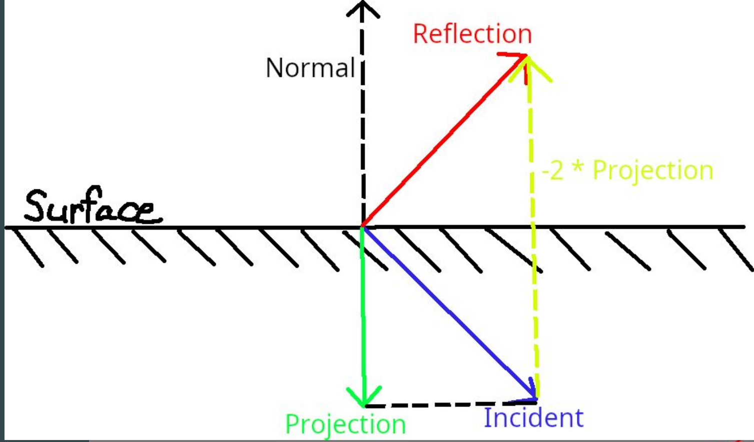
Vector Projection - What is it?

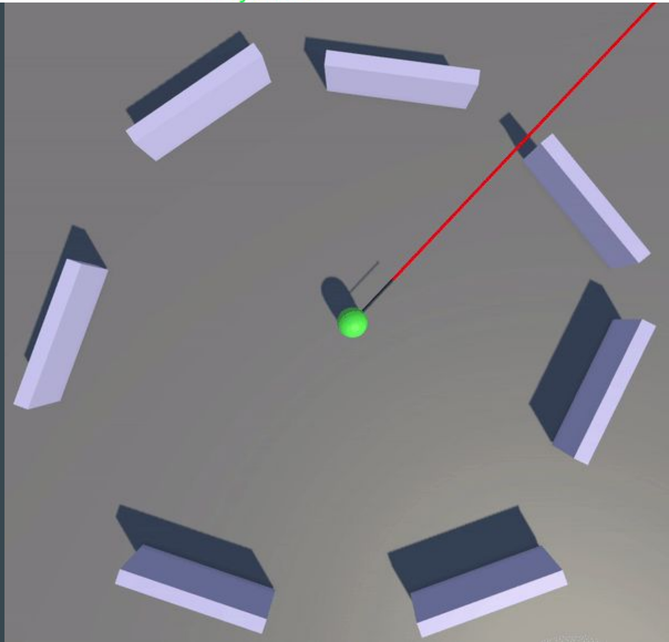
- What is the intuition behind it?
  - Projecting A onto B, or $proj_B A$, gives us the component of A that is in the same direction as B.
- Formula
  - $proj_B A = [(A \cdot B)/(B \cdot B)] * B$
- Notes:
  - Notice that the vector projection formula normalises B.
  - Therefore the projection is not affected by the magnitude of B, for |B| > 0.



Code Snippet

```
private void SetLasers() {
    const int maxLines = 3;
    const float maxDistance = 100.0f;

    foreach (LineRenderer line in lines)
        line.enabled = false;

    Vector3 startPos = transform.position; // Where the line starts.
    Vector3 direction = transform.forward; // Where the line points towards.

    // Let's render each laser segment.
    for (int i = 0; i < maxLines; ++i) {
        LineRenderer line = lines[i];
        line.enabled = true;

        // Check if the laser hits a surface.
        RaycastHit hit;
        if (!Physics.Raycast(startPos, direction, out hit, maxDistance)) {
            // If there is no surface, just render a long straight line in the current direction.
            line.SetPositions(new Vector3[] { startPos, startPos + direction * maxDistance });
            break;
        }
        // If a surface exists, render a line from the current point to the surface.
        line.SetPositions(new Vector3[] { startPos, hit.point });

        // Update the start position for the next line segment.
        startPos = hit.point;

        /* Find the projection of the player's forward direction onto the normal of the surface.
         * Note that both transform.forward and hit.normal are unit vectors. */
        Vector3 projection = Vector3.Project(direction, hit.normal);
        /* Calculate the reflection of the ray off the surface.
         * Note that the reflection direction will also be a unit vector. */
        direction -= projection * 2.0f;
    }
}
```





Q&A

The End