

Professional Certificate in AR/VR Development and 3D Graphics

Summary

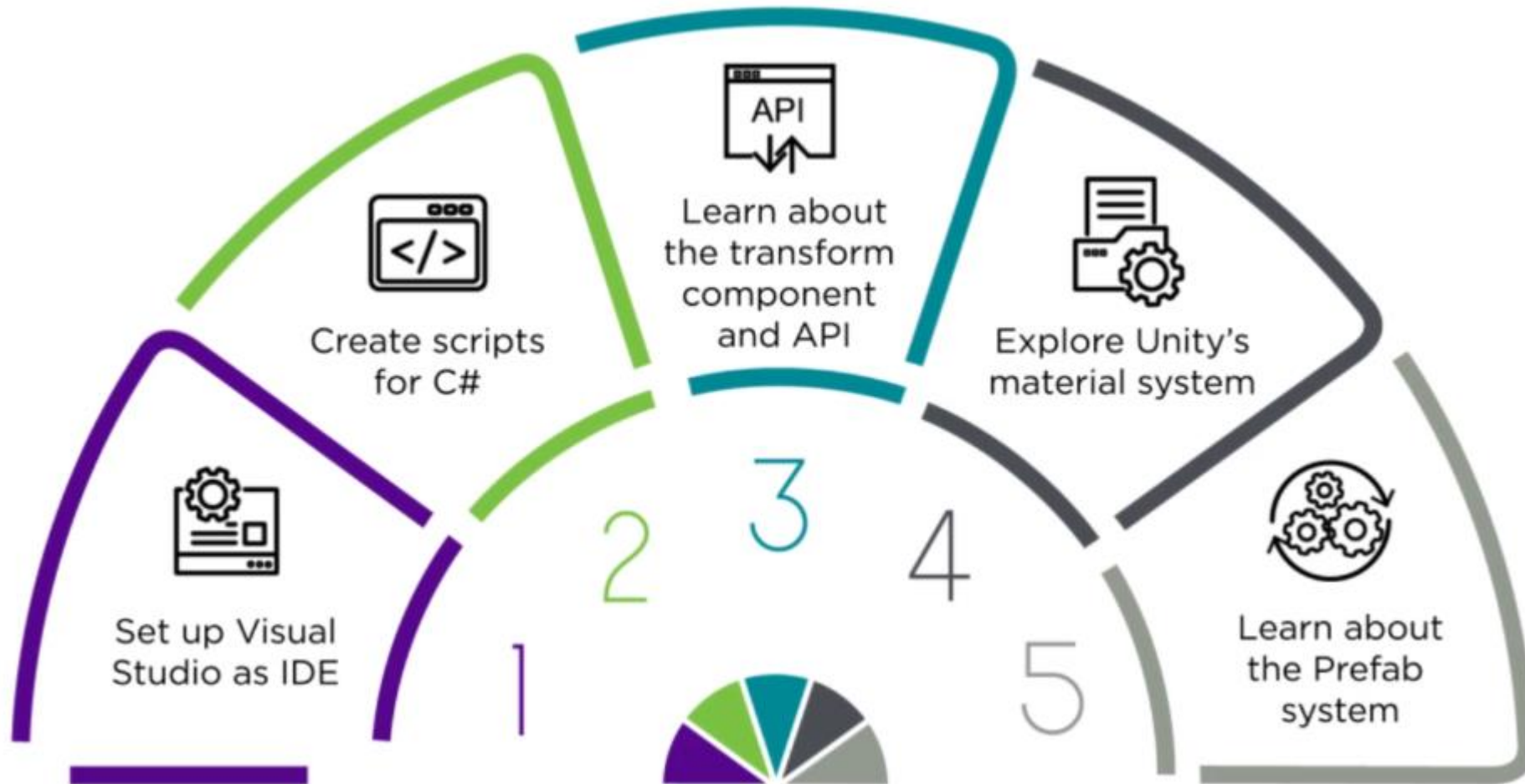
Week 5: Scripting in Unity



NYU

TANDON SCHOOL
OF ENGINEERING

Overview



Basic Topics in the C# Language

One should be familiar with the following:

Data and
variables

Arithmetic

Conditional
statements

Functions and
access modifiers

Iteration in C#

Object-oriented
programming
with C#



NYU

TANDON SCHOOL
OF ENGINEERING

Creating Custom Components in Unity

The bulk of Unity development happens when creating custom components. Custom components can be used to create behaviors such as:

Moving platforms



Created by leveraging the transform API

Breakables barrels



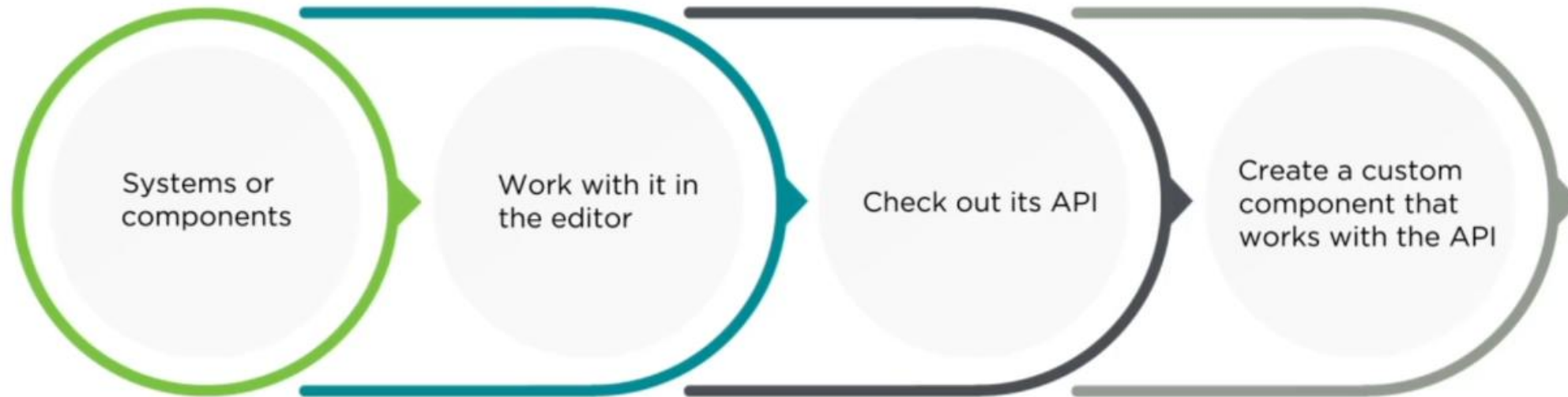
Created by leveraging rigid body, physics, and collider APIs

Functioning doors

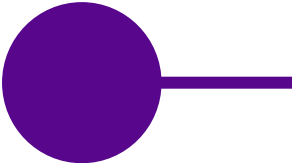


Created by using triggers and scene manager APIs

Role of Scripting in Unity



Setting up Visual Studio



Set up Visual Studio in Unity, by selecting it as the external script editor. He explains the tools and functions of Visual Studio, including:

IntelliSense which is an auto-completion aid that provides snippets of documentation

Messages which are special Unity functions

Unity messages wizard which shows a list of message functions and their documentation

Other editors may also include these features; however, they may have to be set up.

Hello World

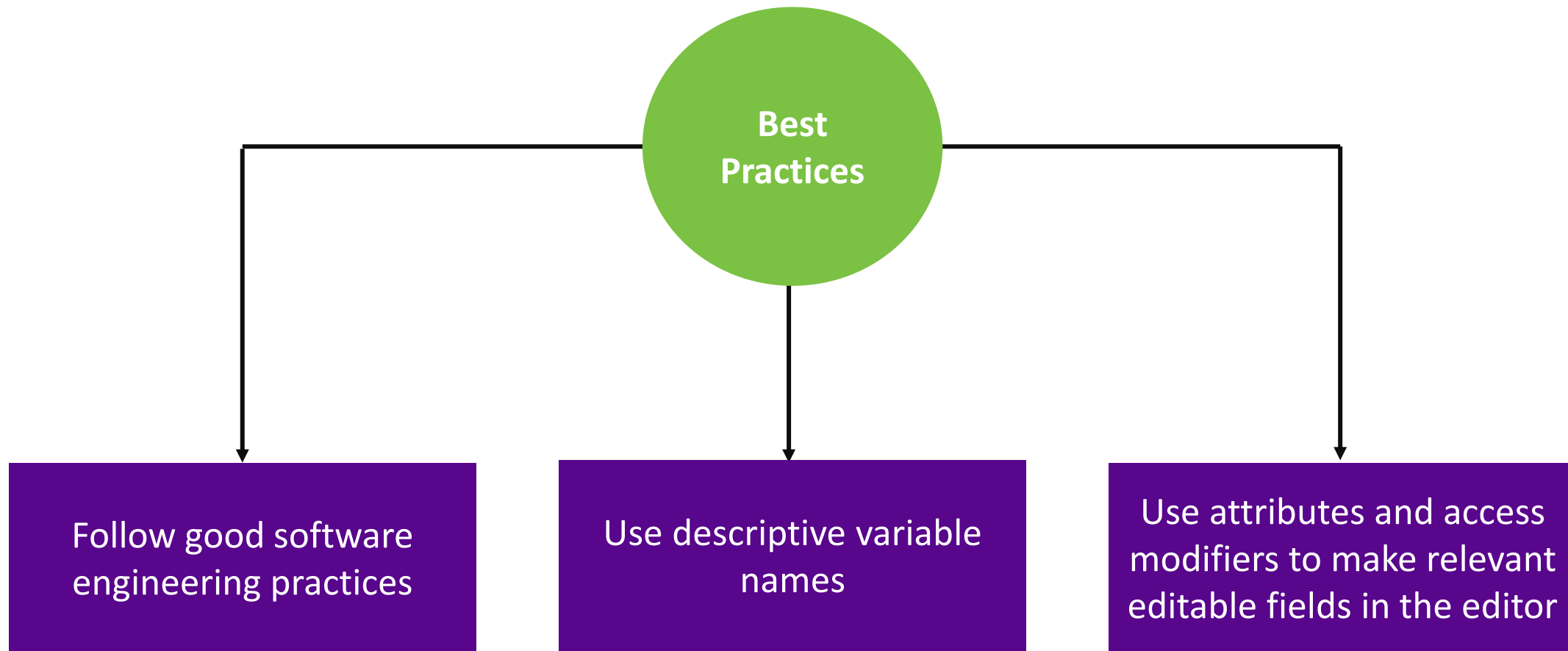
Create a Hello World script and attach it to a GameObject by dragging and dropping in Unity.

Text can be printed on the console using the Debug command.

Creating Custom Components

Scripting within Unity can create special components for a virtual environment. Build a rotator component and add an oscillating platform to a scene using C# scripting and vector mathematics.

Creating Custom Components: Best Practices



Unity Rendering System

The four pieces to rendering in Unity are:

Meshes

The main graphic primitive of unity, which define the shape of objects

Materials

Define how a surface should be rendered by including references to textures, tiling information, color tints, and more

Shaders

Small scripts that contain the mathematical calculations and algorithms for calculating the color of each pixel rendered

Textures

Bitmap images that materials use to calculate the surface color of a GameObject

Unity Rendering System

The Mesh Renderer and Mesh Filter components take in mesh and material inputs to define the shape and visual characteristics of a GameObject

Shaders and textures are tied to the material appearing as configuration options you can specify

Scripting Materials

The material system is one of several systems in Unity such as the physics engine, the rendering system (via the mesh renderer component), or the animation system (via the animator component). The scripting concepts used in the material system can work within any component or arbitrary system in Unity.

Applying a material to a GameObject is as simple as dragging the material onto the object. Existing materials can be modified, or new materials can be created depending on the situation.

When working with scripting it is important learn how to:

- Get a reference to a component
- programmatically change the properties of a component

Introducing the Prefab System

Create, configure, and store a GameObject with all of its components, property values, and child GameObjects as a reusable Prefab asset

The Prefab asset acts as a template to create new instances of the same object

Use the Prefab system to duplicate a GameObject with specific configurations

Creating Prefabs is a better way of reusing assets

Prefabs automatically sync throughout a project, allowing efficient changes to all instances at once

Use the Prefab system to reuse GameObjects with complicated scripts and settings

Prefabs

Create prefabs from a few different sources, then how to edit a feature within a prefab.

Instantiate prefabs at runtime to spawn bonus collectables using empty GameObjects and prefabs with customized scripting.

3D Vector Multiplication

$$\vec{A} \cdot \vec{B} = (a_x b_x, a_y b_y, a_z b_z)$$

Used when vectors represent an abstract quantity or in data science



NYU

TANDON SCHOOL
OF ENGINEERING

Methods of Multiplication for Geometric Quantities

Dot product

$$\vec{A} \cdot \vec{B} = \sum_{i=0}^{n-1} a_i b_i$$

A scalar quantity which results from the sum of each component of the two vectors multiplied together

For two 3D Vectors:

$$\vec{A} \cdot \vec{B} = a_x b_x + a_y b_y + a_z b_z$$

The dot product is a scalar and not a vector

The dot product determines how parallel or perpendicular two vectors are to each other.

Dot Product in Unity

In Unity, the Dot static method is used to calculate the dot product between two variables of the same type.

```
Vector3 a = new Vector3(1.0f, 1.0f, 1.0f); //Define a 3D vector, A
```

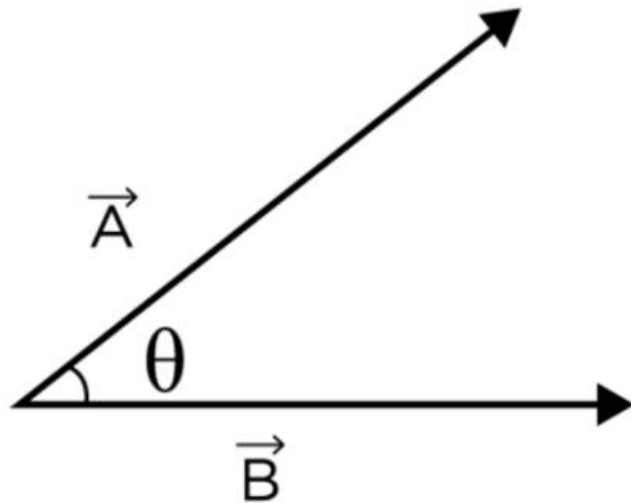
```
Vector3 b = new Vector3(0.0f, 1.0f, -1.0f); //Define another 3D vector, B
```

```
float dp = Vector3.Dot(a, b); //Will return the dot product of A and B
```

Dot Product

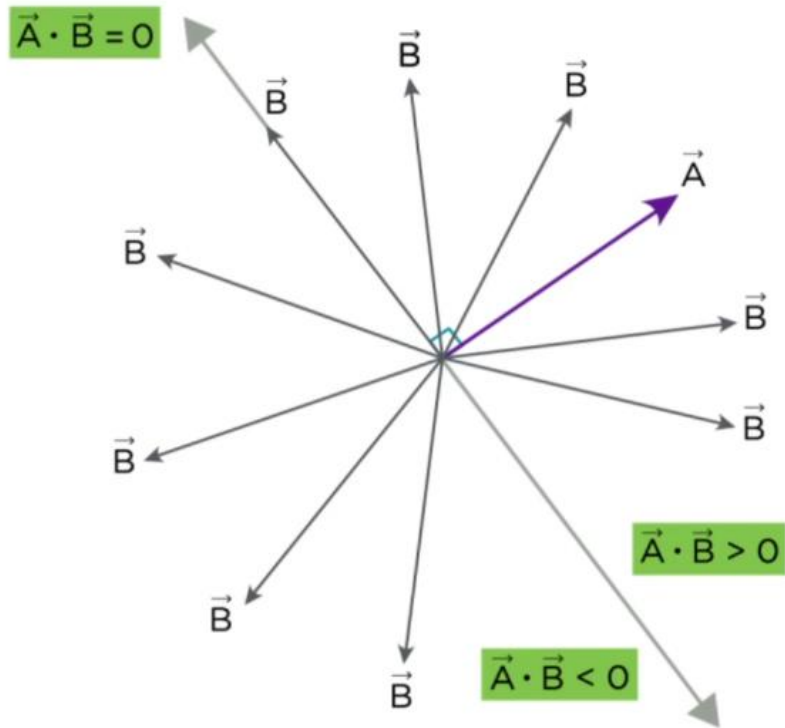
In relation to the angle between two vectors:

$$\vec{A} \cdot \vec{B} = \|\vec{A}\| \|\vec{B}\| \cos \theta$$



Dot Product

The value of the dot product is at its largest when \vec{A} , \vec{B} are parallel and pointing in the same direction.




Key points

- When $\vec{A} \cdot \vec{B}$ are parallel and point in opposite directions, then the value of the dot product attains its largest negative value, because θ is 180 and $\cos 180$ is -1
- Similarly if θ is 90, then the dot product is zero

Cross Product

Algebraically, the cross product between two 3D vectors **A** and **B** is defined as follows:

$$\vec{A} \times \vec{B} = (A_y B_z - A_z B_y, A_z B_x - A_x B_z, A_x B_y - A_y B_x)$$


The dot product will give a scalar quantity, and applies to vectors of any dimensionality, while the cross product will give a third vector which is perpendicular to the other two and only applies to 3D vectors.

Cross Product in Unity

```
Vector3 a = new Vector3(1.0f, 1.0f, 1.0f); //Define a 3D vector, A
```

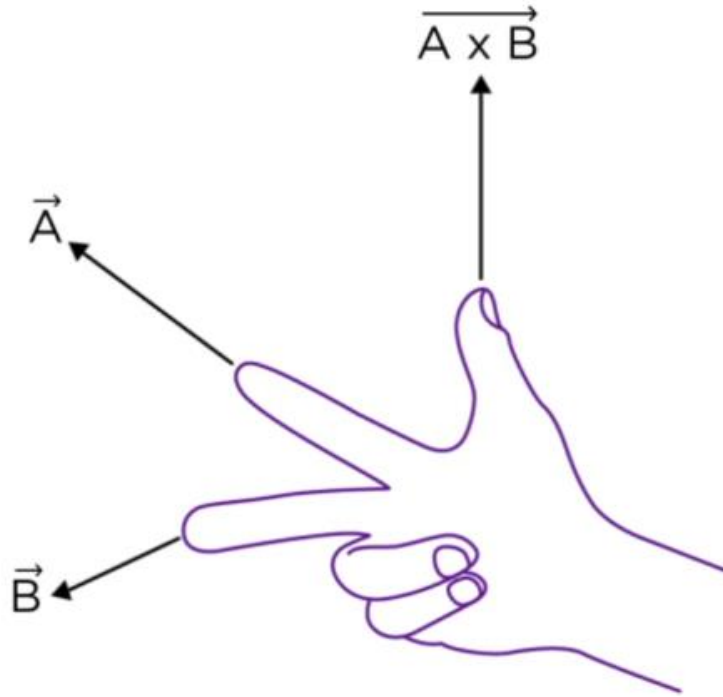
```
Vector3 b = new Vector3(0.0f, 1.0f, -1.0f); //Define another 3D vector, B
```

```
float dp = Vector3.Dot(a, b); //Will return the dot product of A and B
```

```
Vector3 cp = Vector3.Cross(a, b); //Will return the cross product of A and B as a Vector3
```

Right-Hand Rule

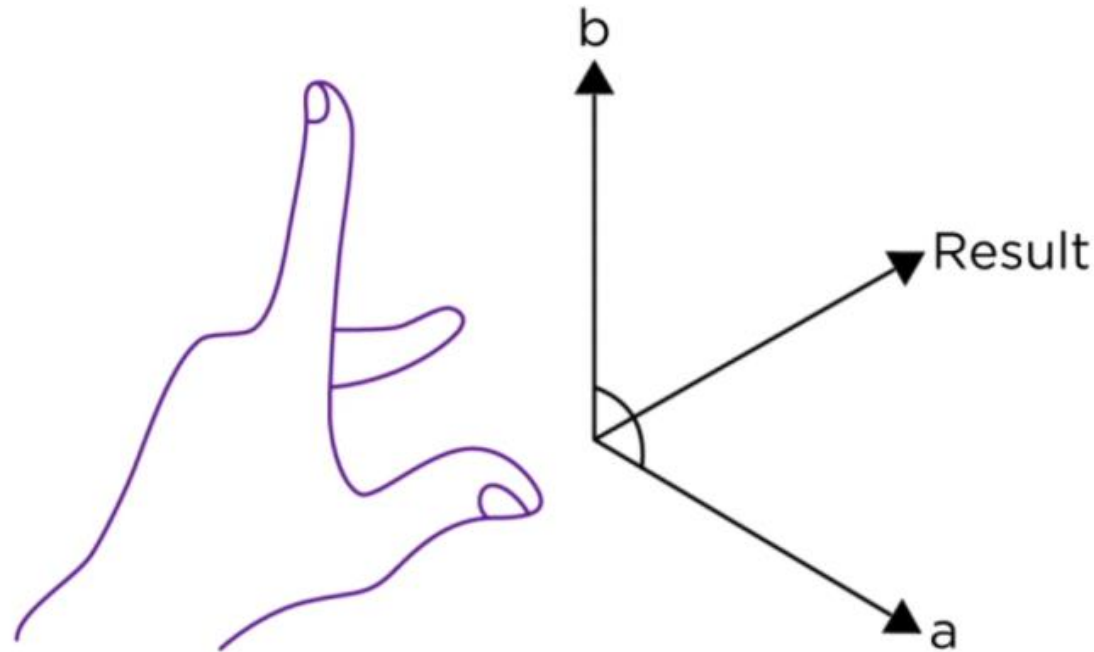
The decision for which direction qualifies as positive or negative is based purely on convention and depends on the **handedness** of the coordinate system.



Hand Rules in Unity

Mathematical convention states that all coordinate systems are right-handed unless specified otherwise.

Unity follows the left-hand rule when calculating cross products.



Properties of Cross Product

Property	Description
$\mathbf{A} \times \mathbf{B} = -\mathbf{B} \times \mathbf{A}$	Anticommutativity of the cross product
$\mathbf{A} \times (\mathbf{B} + \mathbf{C}) = \mathbf{A} \times \mathbf{B} + \mathbf{A} \times \mathbf{C}$	Distributive law for the cross product
$(t\mathbf{A}) \times \mathbf{B} = \mathbf{A} \times (t\mathbf{B}) = t(\mathbf{A} \times \mathbf{B})$	Scalar factorization for the cross product
$\mathbf{A} \times (\mathbf{B} \times \mathbf{C}) = \mathbf{B}(\mathbf{A} \cdot \mathbf{C}) - \mathbf{C}(\mathbf{A} \cdot \mathbf{B})$	Vector triple product
$(\mathbf{A} \times \mathbf{B})^2 = \mathbf{A}^2\mathbf{B}^2 - (\mathbf{A} \cdot \mathbf{B})^2$	Lagrange's identity

These are the basic properties of the cross product. The letters **A**, **B**, and **C** represent the 3D vectors, and the letter *t* represents a scalar value.

Vector Decomposition

Vector decomposition is the act of defining a particular vector in terms of two or more vectors with specific alignments that add up to the original vector.

The simplest case of vector decomposition involves decomposing a particular vector into parts that are aligned to the coordinate axes.

$$\hat{i} = (1, 0, 0)$$

$$\hat{j} = (0, 1, 0)$$

$$\hat{k} = (0, 0, 1)$$

Vector Decomposition and Dot Product

$$\vec{V} \cdot \hat{i} = V_x$$

$$\vec{V} \cdot \hat{j} = V_y$$

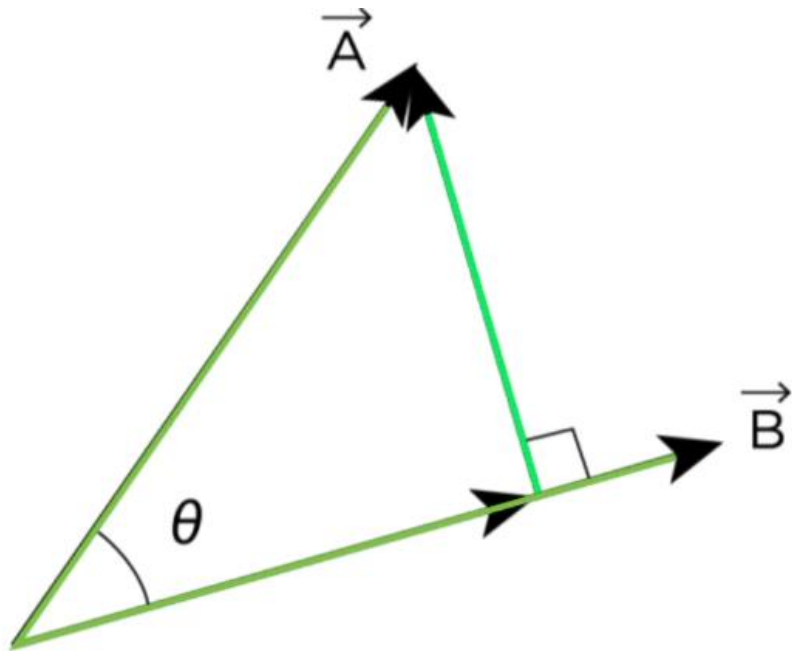
$$\vec{V} \cdot \hat{k} = V_z$$

$$\vec{V} = (\vec{V} \cdot \hat{i})\hat{i} + (\vec{V} \cdot \hat{j})\hat{j} + (\vec{V} \cdot \hat{k})\hat{k}$$

Vector Projection

$$\vec{A}_{\parallel \vec{B}} = \frac{(\vec{A} \cdot \vec{B})}{\|\vec{B}\|} \frac{\vec{B}}{\|\vec{B}\|} = \frac{\vec{A} \cdot \vec{B}}{\|\vec{B}\|^2} \vec{B}$$

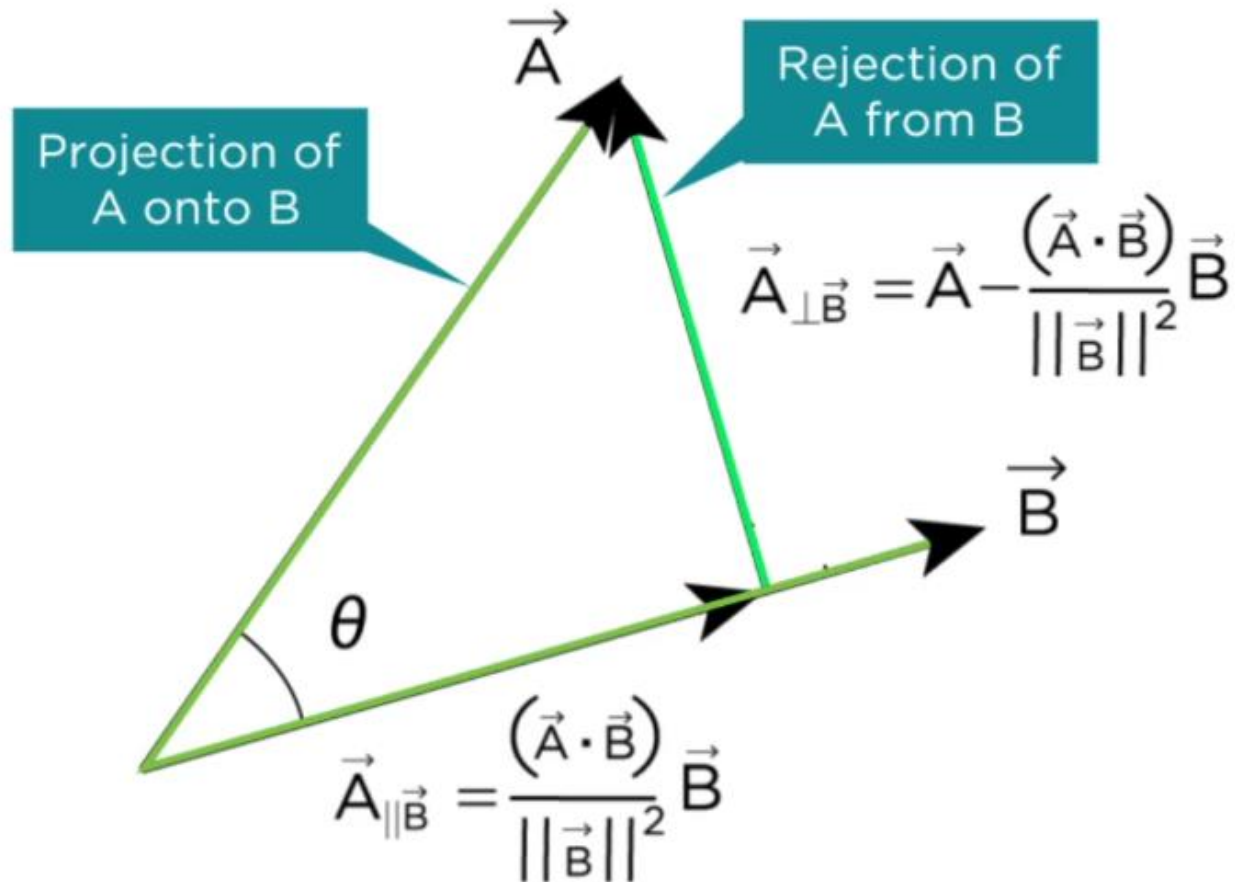
Rejection of A from B



Rejection of A from B

$$\vec{A}_{\perp \vec{B}} = \vec{A} - \vec{A}_{\parallel \vec{B}} = \vec{A} - \frac{(\vec{A} \cdot \vec{B})}{||\vec{B}||^2} \vec{B}$$

Rejection of A from B



Magnitude of Projection and Rejection

Projection

$$||\vec{A}_{\parallel \vec{B}}|| = ||\vec{A}|| \cos(\theta)$$

Rejection

$$||\vec{A}_{\perp \vec{B}}|| = ||\vec{A}|| \sin(\theta)$$

Vector Projection and Rejection in Unity

```
Vector3 a = new Vector3(1.0f, 1.0f, 1.0f); //Define a 3D vector, A
```

```
Vector3 b = new Vector3(0.0f, 1.0f, -1.0f); //Define another 3D vector, B
```

```
float dp = Vector3.Dot(a, b); //Will return the dot product of A and B
```

```
Vector3 cp = Vector3.Cross(a, b); //Will return the cross product of A and B  
as a Vector3
```

```
Vector3 projAToB = Vector3.Project(a, b); //Will return the projection of  
A onto B
```

```
Vector3 rejAFromB = a - projAToB; //Unity does not have a built in function  
for a rejection
```



NYU

TANDON SCHOOL
OF ENGINEERING

Summary



Create and attach custom components to GameObjects



Use the Transform API to work with transforms programmatically



Be familiar with the basics of Unity's material system



Work with Unity's material system in the editor as well as from scripts



Use Unity's prefab system to create reusable template GameObjects for projects and instantiate new GameObjects into scenes



NYU

**TANDON SCHOOL
OF ENGINEERING**