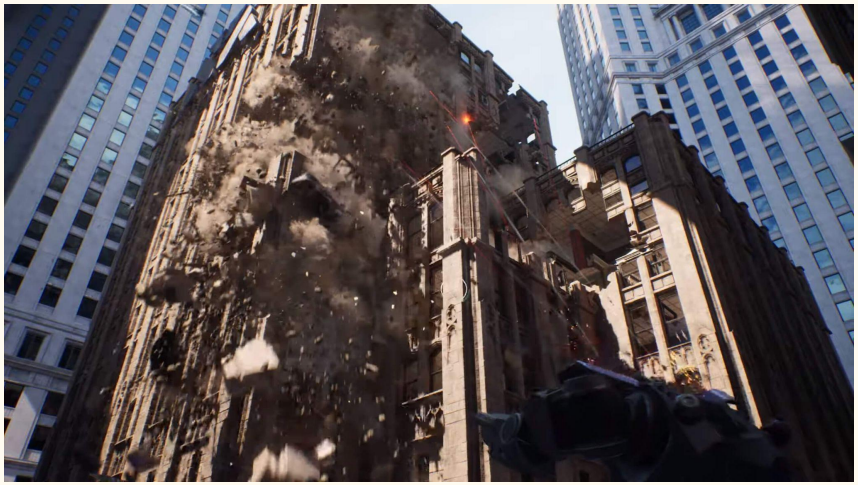
Unreal Engine 5 - Lesson 6 - Physics

—

Nicolas Serf - Gameplay Programmer - Wolcen Studio
serf.nicolas@gmail.com

Physics engine

- Let's start by speaking about **physics engine**
- Unreal Engine being a **game engine**, it needs to **relies** on a **physics system** in order to make **physics computation**, **collision detection**, etc...
- There is a chronology in physics engine usage for the engine
  - On **UE2**, Havok was used and then drops because of **lack** of **functionalities, slower** and a h**igh licensing cost**
  - On **UE3**, **software character movement** was made **in-engine**, while **PhysX** was used for **physics objects** and **vehicles.**
    - The transition from **Havok** to **PhysX** can be explained by many things like **NVidia GPU hardware acceleration**, **sponsoring**, **faster**, etc...
  - On **U4**, **character movement** was **reimplemented** on top of physics (PhysX by default). It does **simplified interaction** between **character and physics objects**.
    - With this change UE4 **require** PhysX in order to run.
  - In later stage of development of **UE4**, they start working on **animation physics engine** for cheaper **cloth** / **chains** / **hair**, used for Paragon & Fortnite. This leads to the **creation** of **Chaos Physics**, given more **control** and **optimization** scope.
  - On **U5**, **PhysX** was dropped to use **Chaos Physics** instead
    - Technically speaking, PhysX is still available to be compiled from source but needs to be ported to UE5



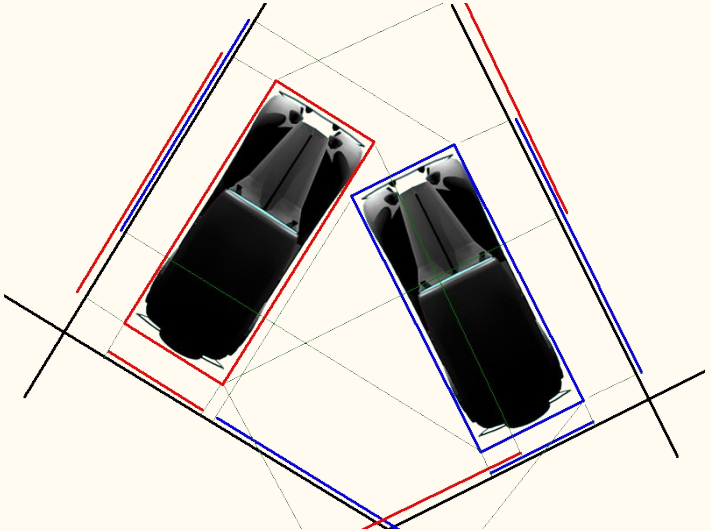
Chaos Engine

- **Chaos Engine** is now the **default** physics engine
- Physics is a really **difficult subject** with a lot of **point** that would require **entire lesson** to be simply **highlighted**
- In this lecture, we'll go through the **basics of physics** on which all **complex system** are **built on**. However, in order to offer the **possibility** to check by **yourself**, here are some **complex topics** that we may want to check out
    - Chaos destruction
    - Cloth Simulation
    - Fluid simulation
    - Hair Physics
    - Physics Fields
    - Etc...

Collision overview

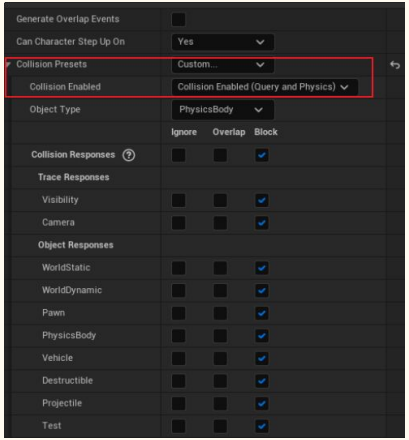
- When we are thinking about **physics**, we are thinking about **two** mains **topics** in most cases
  - **Collisions**
  - **Raycasting**

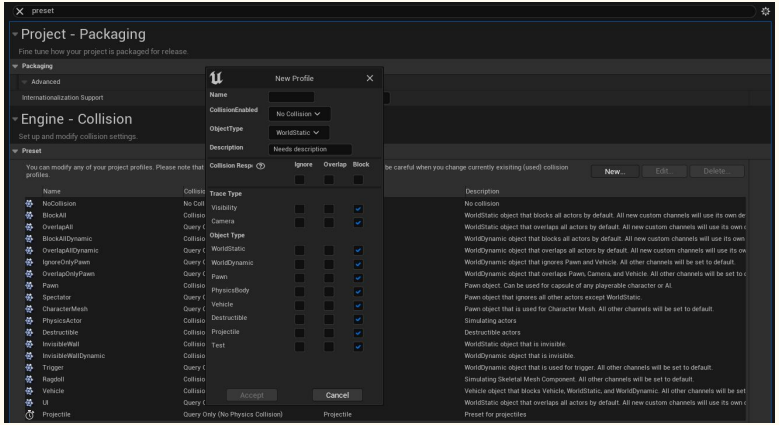
- Every object that needs to **generate** some sort of **geometry** to be **rendered** or use as a **collision data** inherit from **UPrimitiveComponent**
  - Inside them **ShapeComponents** are **collision component** which generated **geometry** only used for **collision detection**
  - **StaticMesh** & **SkeletalMesh** components contain **pre-built geometry** that is **rendered** but that can also be **used** for **collision detection**



Collision Presets & Collision Enabled

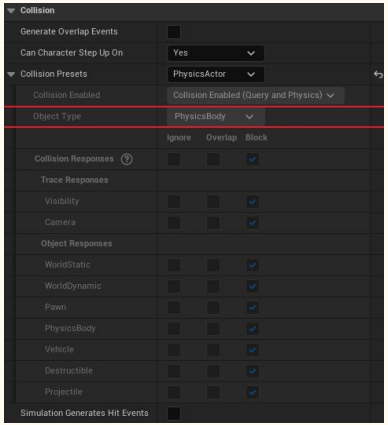
- Collision Presets allows to have a **predefined collision matrix** setup based on the **preset**. But using a preset **doesn't allow** to **modify** how it should behave. In order to do so, you need to select **"Custom"** in the **Collision Presets**
  - It is possible to **create** new preset through the **Project Settings tab**
- Collision Enabled has 4 different values :
  - **NoCollision** : **No representation** in the physics engine.
  - **QueryOnly** : Only for **spatial queries** (raycast, sweeps, etc...). It is useful for character movement and object that do not need physical simulation.
  - **Physics Only** : Only for **physics simulation** (rigid body, etc...). This is useful for simulated secondary motion character not needing per bone detection
  - **Collision Enabled** : For **both**, **spatial queries** (raycast, sweeps, etc...) and **simulation** (rigid body, constraints, etc...)

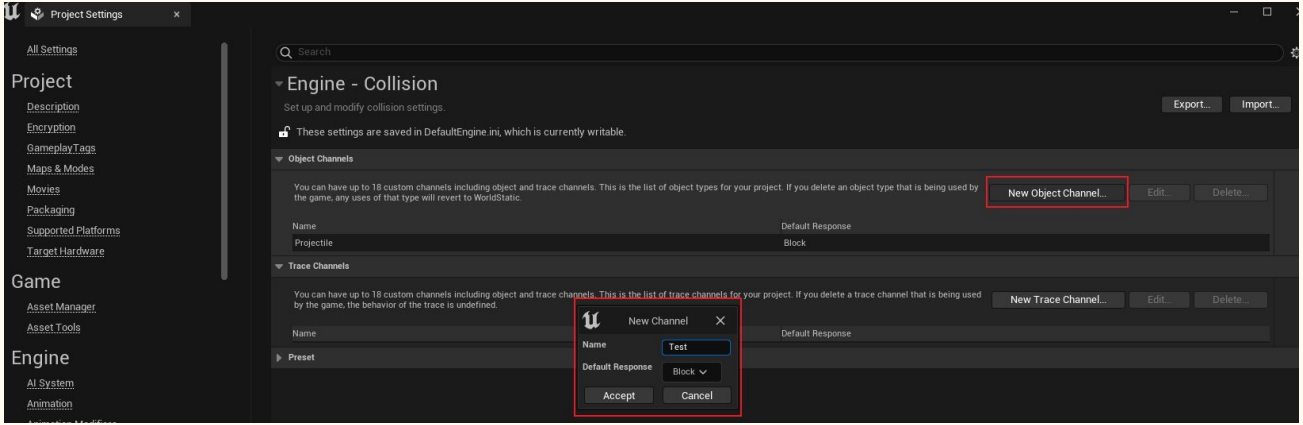




Object Type

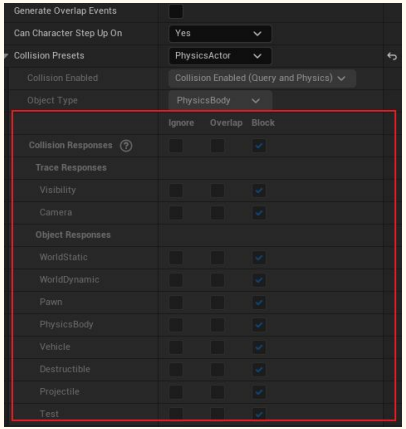
- Each collider get an **Object Type**
- **Object Type** are **essential** in **collision detection** and **interaction** has there are the type **used in object responses** when you are defining how an object **should interact** with the world
- By going into **Edit>Project Settings>Collision**, you'll be able to **create new object channel** that will be usable as **object type** in the collision
- As you can see during creation, **Object Type** has a **default response** that can be changed from this **panel** also





Collision responses

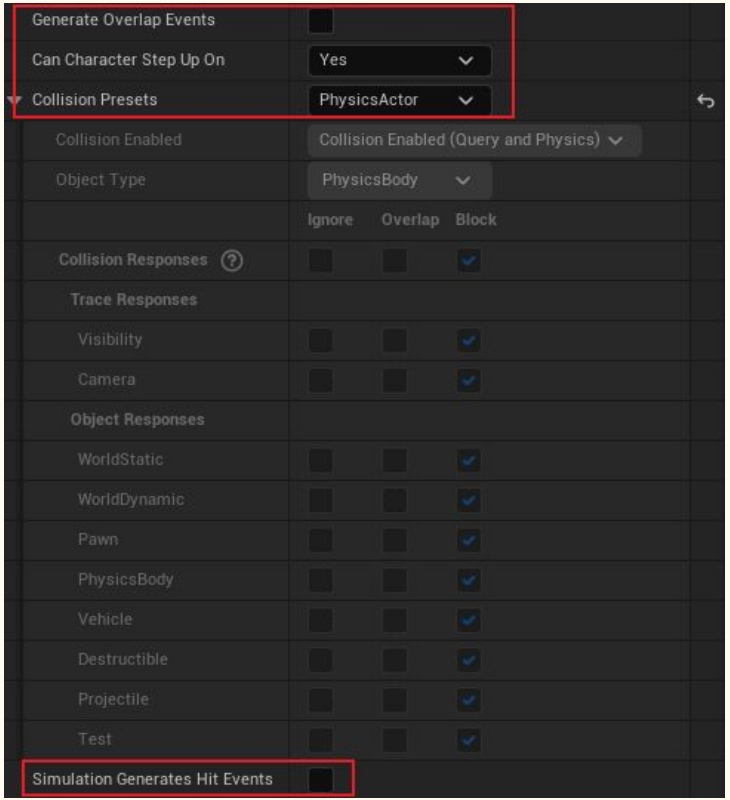
- Coming directly from **Object Type**, collision **responses** is the way to **precise** how the collision should be **handled**
  - **Ignore** : **No collision** or **overlap** event can be **trigger** between this object and the object type
  - **Overlap** : **No collision** can be trigger but **overlap** may be if an **overlap** or **block** is present in the other object type
  - **Block** : **Blocking** can be trigger if a block is **present** in the other object type



Other parameters & rules to keep in mind

- As you can see, there is a bunch of other parameters that are needed to configure as you would like
  - **Simulation Generated Hit Events** : It needs to be **enabled** in order to execute **Event Hit** which is used in **Blueprint**, **Destructible Actors**, **Triggers**, etc...
  - **Generate Overlap Events** : It is needed if you want actors to be in **overlap mode.** If you select a **response** as **overlap** but **don't enabled** this option, it will be **essentially** the same as **Ignore**
  - **CanCharacterStepUpOn** : It is **reflecting** how the **physics** should behave when a **mesh is falling / walking** on the **component.** If set to **false**, it will **reject** it with a **bump**

- There is some **rules** to keep in mind when dealing with collision
  - For 2 or more simulating object to block each other, they **both** needs to be **set** to **block** their **respective object types**
  - For 2 or more simulating object, if **one is block** and the **second one is overlap**, **overlap** will occurs but **not the block**
  - **Overlap events** can be generated even if an object **Blocks** another, especially at **high speeds**
    - It is **not recommended** for an object to have **both collision** and **overlap events**
  - If an **object** is set to **ignore** and the **second one to overlap**, **no overlap** will be fired

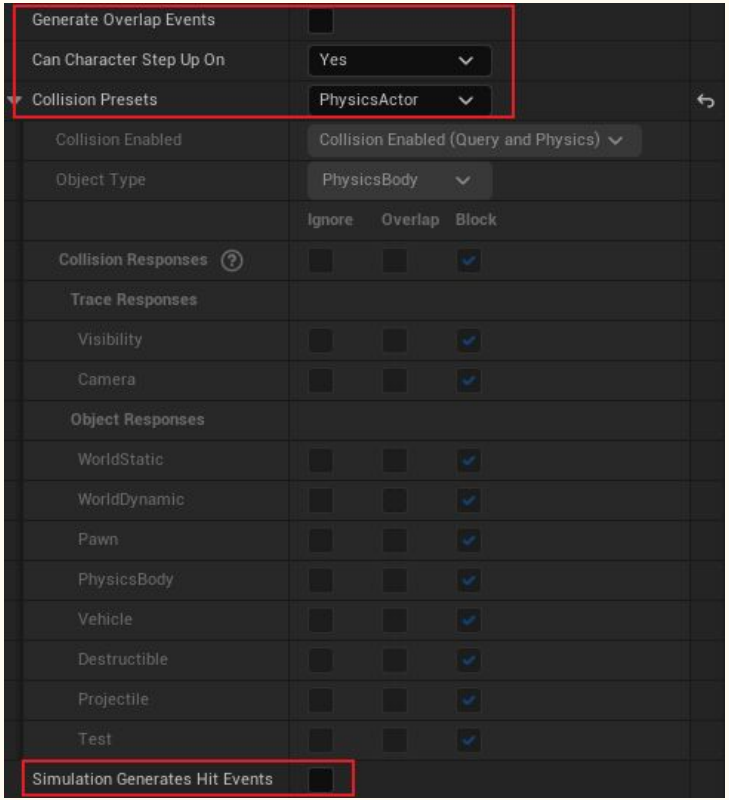


Overlap Events

- Unlike collisions that can **fire every frame**, overlap are based on **event** with
  - **ReceiveBeginOverlap** : It will be called when the **component** which is compatible in the collision matrix will have it's **collider entering** into the object one
  - **ReceiveEndOverlap** : It will be called the same way of **receive**, but when the **collider exit** the object one.

- Warning about this **type of event**, it will fire for **every overlap** object. It mean that you need to **properly setup collision matrix** or you may fall into p**awn armor pieces** for example that will trigger **overlap** event when you'll not want

- **REMINDER** : For an overlap to occurs, **both Actors** needs to enable **Generate Overlap Events.**



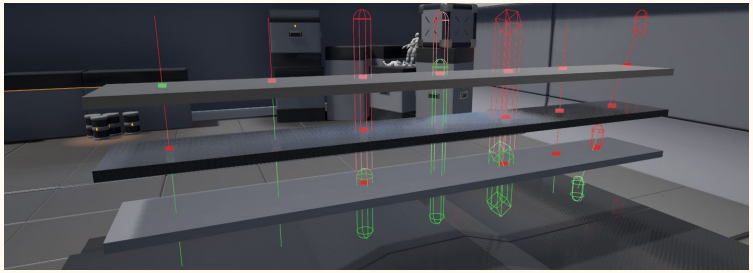
Trace overview

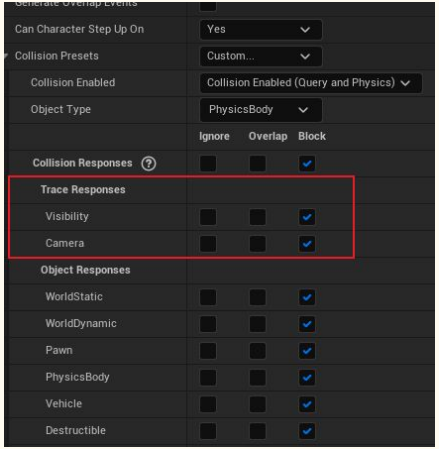
- Another important subject when it comes to physic is **Raycasting**
- Raycasting is **essentially** the **same** as **Trace** in some other engine or software packages. In order to stay inline with **Unreal naming convention**, we'll only use **traces** now

- Trace offer a method for **reaching out in your levels** and **getting** feedback on what is **present** along a **line segment**.
- You use them by providing **two end points** (**start** and **end**) and the **physic engine** will **traces** a line segment between those points, **reporting any Actors** (with collision) that it **hits**

- You'll use traces in **many circumstances**
  - Know if an Actor can see another
  - Get the normal of a specific polygon
  - Simulate high velocity weaponry
  - Check if an actor entered a space

- Traces offer a **reliable** and **computationally cheap** solution.
- Trace uses the physics system and therefore responses system.

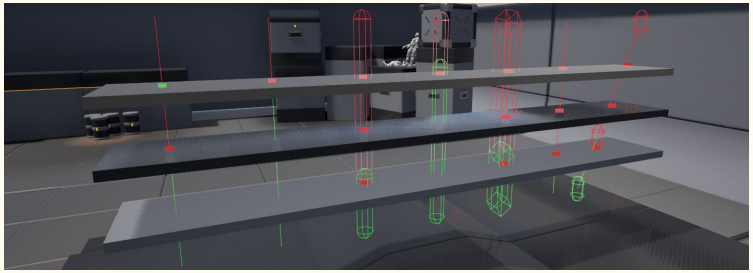


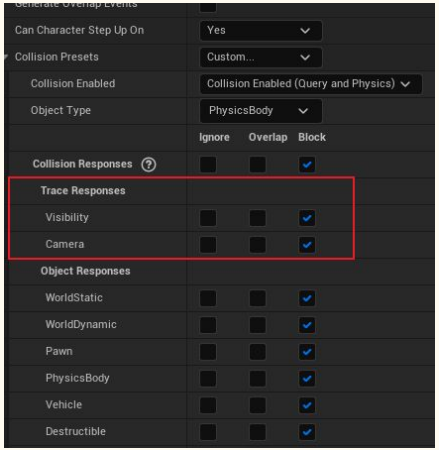


Trace by channel or Object Type

- Even if traces uses **response system** there is **2 options** that you can choose when tracing
  - **Channels** : Channels are used for t**hings like visibility** or **camera** and almost **exclusively** have to do with **tracing**.
    - This is a good way for example to **ensure** to **trace** against specific **actor** that **response** to that traces
    - Like **Object Type**, you can **create** custom trace response in **Project settings**
  - **Object types** : There are physics types of Actors with collision in your scene like Pawns, Vehicles, etc...

- The choice between tracing **by channel** or **object type** will always be **different** based on the **context** and there is **no global answer** to that question



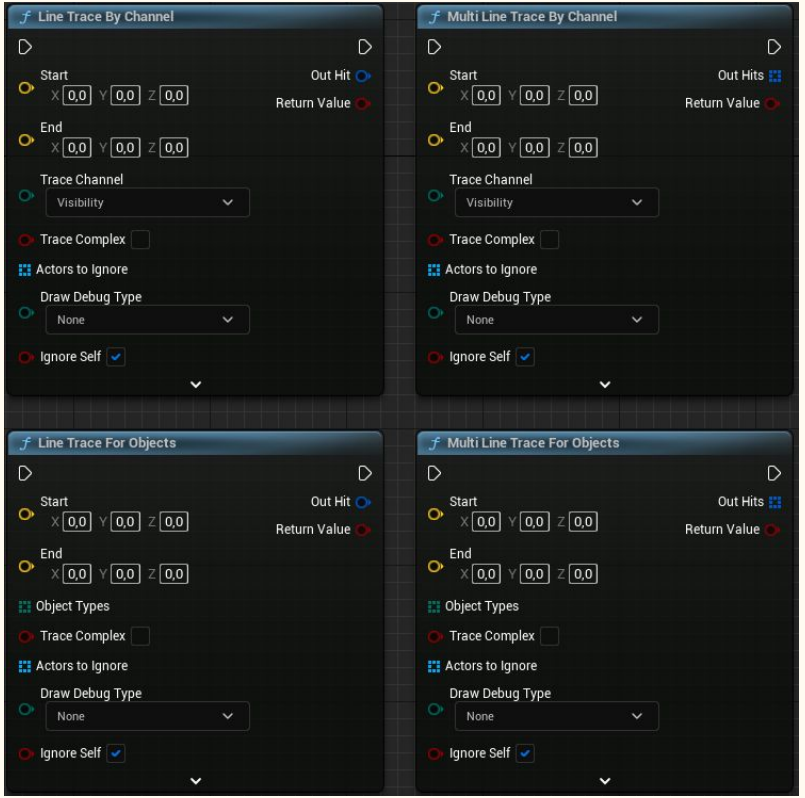


Trace Single or Multiple Hits

- When you are tracing, you can choose to return the **first things** that matches the criteria hit by trace or you can return **everything** hit by the trace that matches the criteria

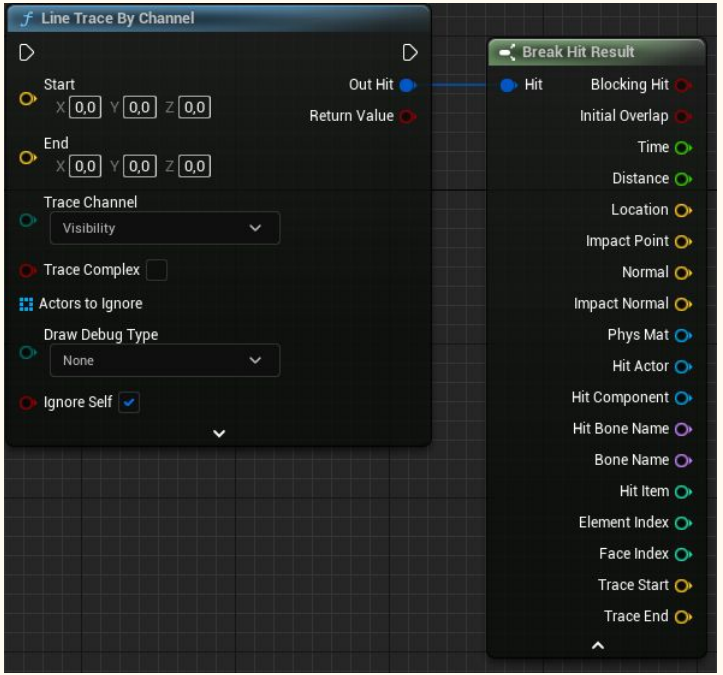
- **Note** : A special consideration is given to Multi trace by channel versus Multi trace for objects
    - With **MultiTrace by Channel**, the trace will return all **Overlaps** up to and **including** the first **Block**
    - With **Multi trace by object**, it will return **everything** that matches an object type trace trace is looking for.

- Keep in mind that **generally speaking**, **except** for trace for **objects**, if an object blocks the trace, it will make sure that the multi trace **will not registered** result **after** the **blocking one**



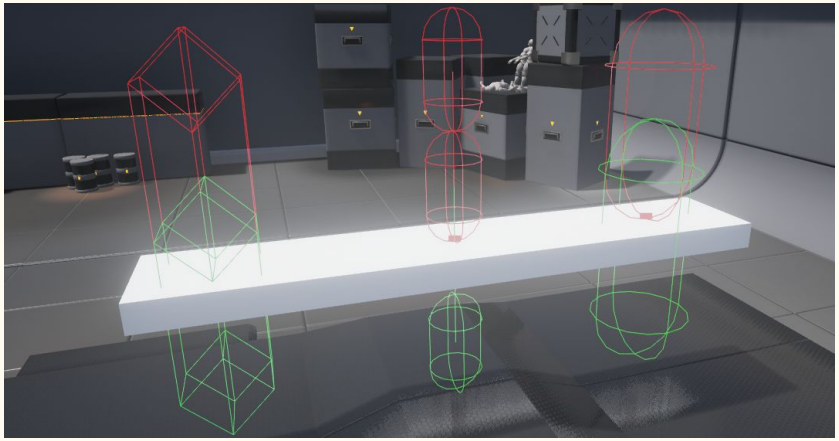
Hit Results

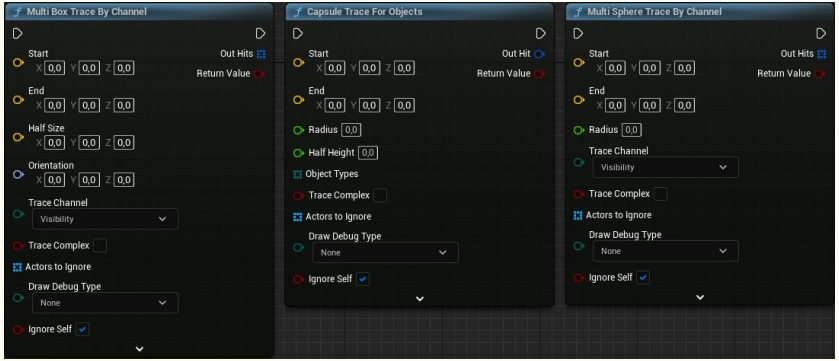
- When a trace hit something, it will return **true** and a **struct** which is available in **blueprint** and **C++**.
- There is a bunch of member in that struct that we can check on
  - **Blocking Hit** : Is the hit a blocking one ?
  - **Initial Overlap** : Is the first overlap ?
  - **Location** : **World space location** of the hit, **modified** based on the shape of the trace
  - **Impact Point** : **Absolute location** of the hit, **does not include** the **shape** of the trace, only the point of the hit
  - **Normal** : Direction of the trace
  - **Impact Normal** : Normal of the hit surface
  - **Phys Mat** : Physical Material of the hit surface
  - **Hit Actor** : The hitten Actor
  - Etc...
- Difference between **Location** and **ImpactPoint** only matter for **shape trace**
- Do not hesitate to **consult documentation** if you are not sure about the member variable



Shape traces

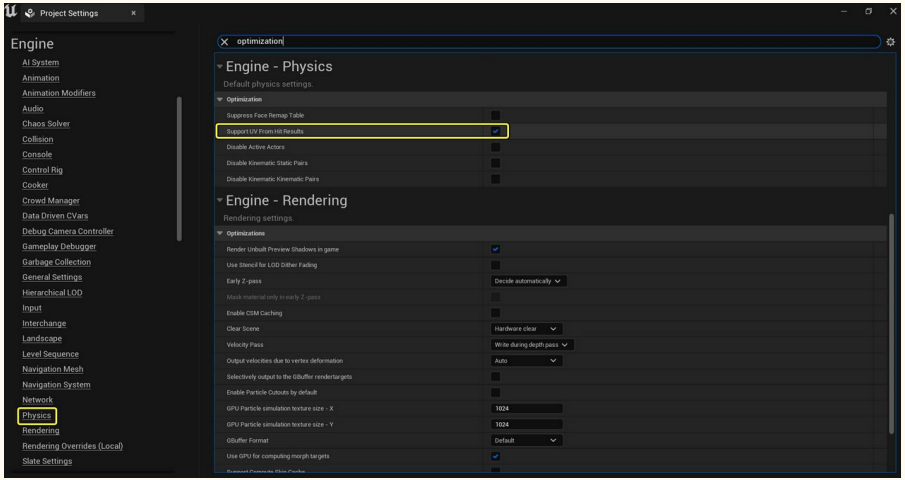
- Sometimes, **Line Trace** are not **enough**
- Shape trace are **based** on line trace with more **complex computation**
- They function like **Line Traces**, giving an **HitResult** and having **multiple variants** for **single** or **multi** trace and **For Objects** or **By Channel**
- They have an **added layer of checking** obviously as they are using a **shape** as a **volume** in the **tracing**





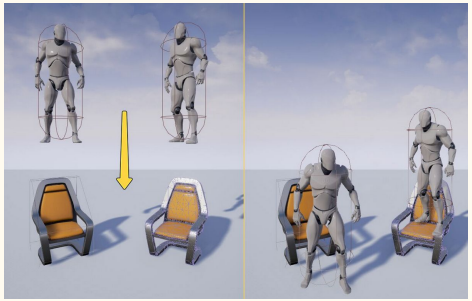
Getting UV Coordinates from Trace

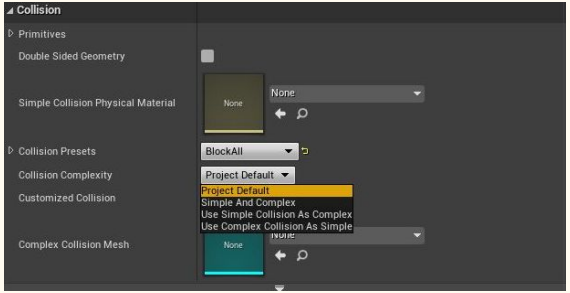
- Let's talk about a last point that may be useful in come cases, getting **UV coordinates** from trace
- It assume that **trace complex** is **enabled** in the trace settings
- It assume that **Support UV From HitResult** is **enabled** in **Project Settings**
- It has **some limitation** like only working on **Static Mesh Components** and **Procedural Mesh Component.**
- It increase **CPU memory usage** as the engine needs to **keep an additional copy** of vertex positions and UV coordinates in main memory
- It **allows** in HitResult to get **UV coordinate** of the **hit**, which may be useful for example in **creating visual effect** precisely on the impact point directly on the **material texture**



Simple VS Complex Collision

- You have access to **simple** and **complex collision shapes**. **Simple Collision** are **primitives** like **cubes**, **spheres**, **capsules**, and **convex hulls**. Complex Collision is the **trimesh** of a given object. By default, Unreal Engine creates **both simple** and **complex** shapes, then, based on what the user **wants** (complex query versus simple query), the **physics solver** will use the **corresponding shape** for scene queries and collision tests.
- There is 4 types of collision complexity
    - **Project Default** : Using the **project's physics settings,** this will cause **simple collision** requests to use **simple collision**, and **complex** requests to use **complex collision**; the "default" behavior.
    - **Simple and Complex** : This flag **enables** the creation of **simple** and **complex shapes**, using s**imple shapes for regular scene queries and collision tests**, and using **complex** (per poly) shapes for **complex scene queries**.
    - **Use Simple Collision As Complex** : This means that if a **complex query** is requested, the engine will still **query against simple shapes**; basically ignoring the **trimesh**. This helps **save memory** since we don't need to **bake** the trimesh and can **improve performance**
    - **Use Complex Collision As Simple** : This means that if a **simple query is requested**, the engine will query **against complex shapes**; basically ignoring the simple collision. This allows us to use the **trimesh** for the physics simulation collision. Note that if you are using **UseComplexAsSimple** you **cannot simulate** the object, but you can use it to **collide** with other **simulated** (simple) **objects**.





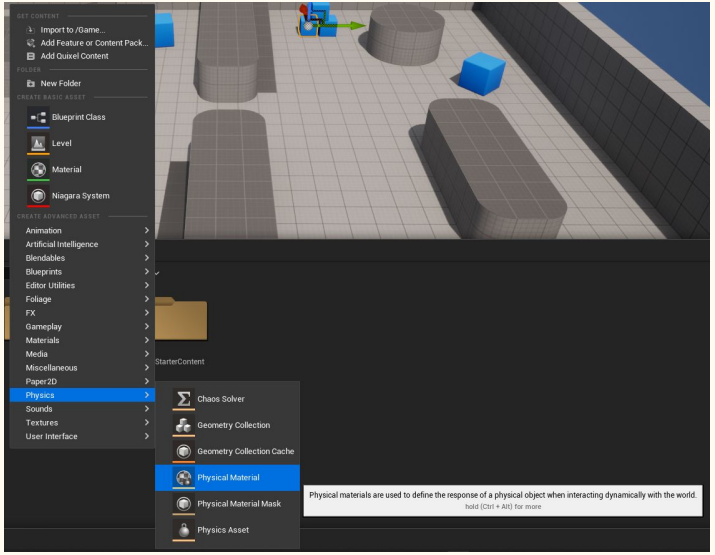
Physical Material

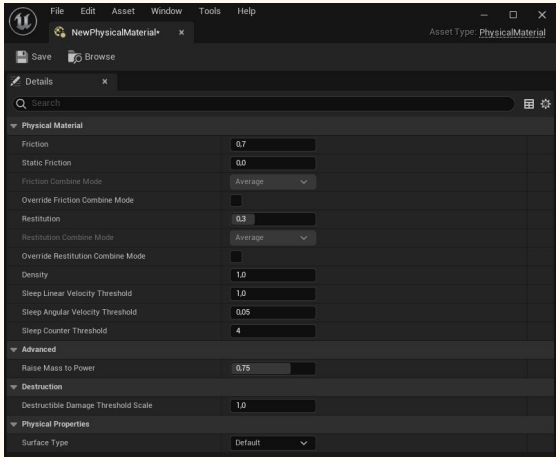
- **Physical Material** are **.uasset** element
- You can choose from which **class** you want your **Physical Material to inherit from**, the **default** one being the most **common**

- You can then **adjust** the settings of your **physical material**.
- You can see that the **last member** of the PM is **Surface Type**. It can be whatever you **want** from all surface type available in your project. You can **create new ones** in **Project Settings > Physics > Physical Surface**.
    - If can be useful to specify for example that your physical material represent grass or sand

- There is multiple ways to assign a PM to a material
    - **To a material** : In **detail panel** directly in **Material**
    - **To a material instance** : In **detail panel** of your **Material Instance**
    - **To a Physics Asset** : From a **Physics Asset**, open the **Physical material dropdown** and select your asset
    - **To a Static Mesh** : In the **Simple Collision Physical Dropdown**, you can select the desired PM

- For a breakdown of the properties, check PM Properties





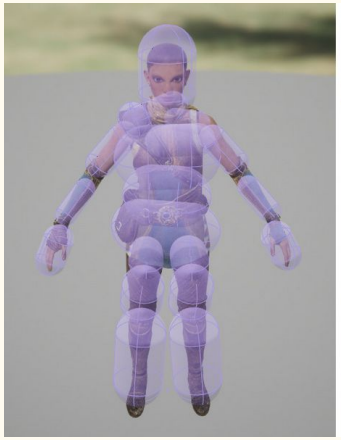
Physical Asset

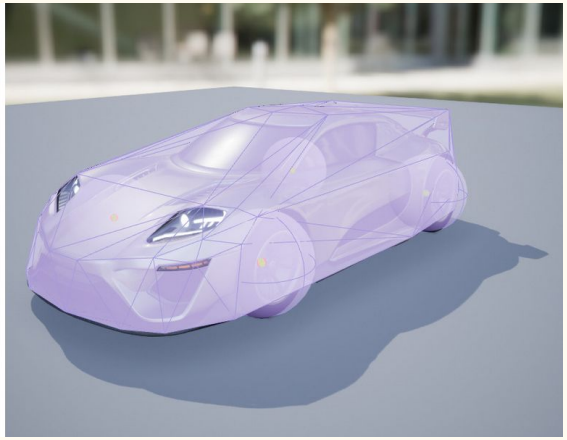
- **Physic Asset** are .uasset element
- They can be set up for any skeletal mesh for simulation
  - It can be a **Character**
  - It can be a **Vehicle**

- **Physic Asset** are used to defined the **physics** and **collision** used by a **Skeletal Mesh**. These contain a **set** of **rigid bodies** and **constraints** that make up a single **ragdoll**.
- They are basically the **physic representation** in the physical engine.

- You can create a **PA** by enabling the option while **importing**
- You can create a PA from the **content drawer** Physics > PhysicAsset

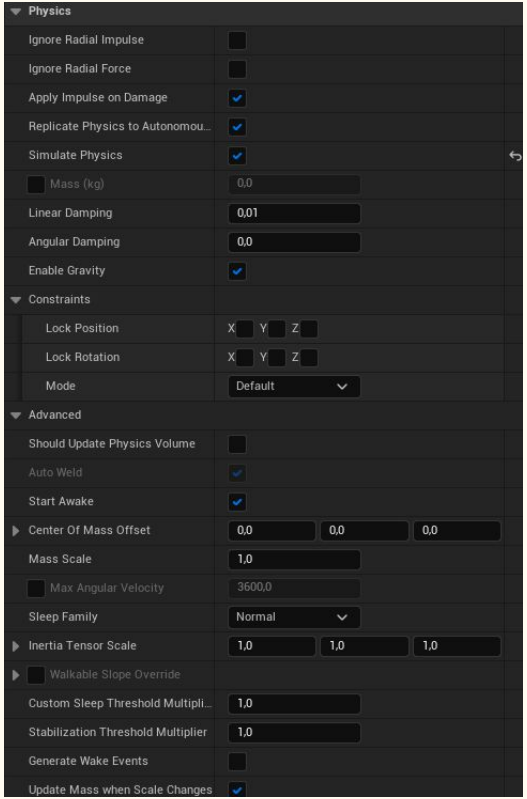
- It is more likely that this **kind of setup** will not be made by you but from **artist**, so we'll not go any further in this, but keep in mind that this is the thing **dictating** how your **Skeletal Mesh** will behave.





Physics bodies

- **Physics bodies** is the complex system that dictate how an **entity** behave in **physic engine**.
- It obviously cover the **collision responses**, but also all **parameters** that **specify** the **mass**, the **linear damping**, the **mass scale** etc...
- This value are set by **default**, but it is obviously **something** that needs to be **tweak** to match with the game feels looked for
- Again, it is more likely that this values will be setup by **artists** or **designer**, but it important to know where to find this informations if there is an **issue** in the **physics simulation** of the game



Add forces

- When you want to make your character moves, you should use the Character Movement Component way to do so, by calling for example
  - **AddMovementInput**
  - **SimpleMoveToLocation**
  - **Etc...**
- In some cases, you may want to add a physical force to your character, that interact correctly based on the physic system, in order to do so, you have access to two main calls
  - **AddImpulse** is designed to be used **only once** to add a **burst** at a location. E.g hitting a golf ball.
  - **Add force** is designed to be used **multiple times** to gradually **move objects**. E.g Player pushing a heavy cube

Time to.... highlight a concept

Prepare your job interviews

Practice

- **General**
  - Place 4 cubes in your level which needs to interact differently with the player
    - 1st : It completely ignore the player just like a non-interactive decor element
    - 2nd : It doesn't trigger any physical change, the object remains in place BUT it display a message on the screen
    - 3rd : It interact physically with the player, making the cube moved upon collision
    - 4th : It interact physically with the player, making the cube moved upon collision + it display a message on the screen
  - Create a Physical Material for your 3rd cube in order to make it heavier and much more difficult to move
  - Create a method that check if an object is inside a cone in front of your player, like a sword slash, do it mathematically

- **Follow-through project**
  - Ensure that your guards are able to detect your player and make it easy to tweak the value like search distance, etc...
    - Feel free to manage that as you want, but try to make a double test like a detection cone, then a trace
  - Make your player interact with gold ingot in order to collect it.
    - Upon collision, destroy the gold ingot
    - Upon collision, trigger a Niagara System to inform player that it has collected a gold ingot
    - Upon collision, increase the number of ingot collected
  - Create an ability that is triggered when you press a Key, for example E
    - It uses the scan detector which detect every guards around the player, even if they are behind a wall
    - It create 3 pulses separated by 3 seconds, and cannot be used for 1 minutes after that
    - For every guard affected, it change the material of its outfit
  - Create an ability that is triggered when you press a Key, for example R
    - It uses an IEM system, starting at mouse location, it cast a cube around it and cancel every electronics of environment
    - Guard / Lamp / etc... are disable