# Lesson 5

# Physics

Nicolas Serf
serf.nicolas@gmail.com


PHYSICS


Physics
Collision meshes

# Physics

1. Collision meshes
2. Body
3. Interaction
4. Traces
5. Detection
6. Matrix


## Definition

A **mesh** is by definition a **collection** of **vertices** that form **triangles**. When it comes to physics, a mesh is a **complex piece** to handle as they form a complex **collision shape**.

## Vertices

If you want to strictly follows the **vertices** for a **collision**, you'll need to ensure that in **incoming** object to not **intersect** with **any triangles** formed by the vertices.

## Auto-generated

When it comes to **auto-generated** collider, engines will never makes a **collision detection** per **triangle**, it would be **impossible**. They'll most likely form **hull-convex shape** around your **mesh**.

# Physics

1. Collision meshes
2. Body
3. Interaction
4. Traces
5. Detection
6. Matrix







## Definition

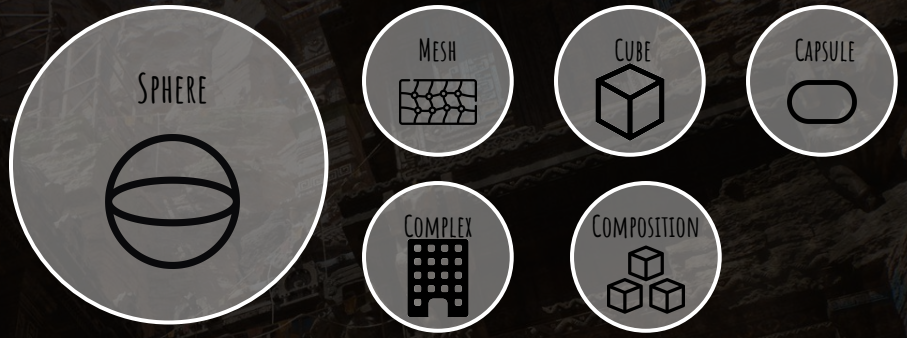
A cube is really **easy** for physics engine to be computed. Actually, it is not really a **cube** but a **rectangle**.

## Extents

For a **cube**, we defines **extents**, it means basically, from the **origin point**, how it **extends** in the 3 direction. It will basically form the **final rectangle** that will be used.

# Physics

1. Collision meshes
2. Body
3. Interaction
4. Traces
5. Detection
6. Matrix




## Definition

A sphere is also an easy **shape** to be **computed** by physics engine. In simulation, a **sphere-sphere** collision will be the **easiest** to compute as it is simply a **distance check**. Consider that a **cube** can be used also.

## Center

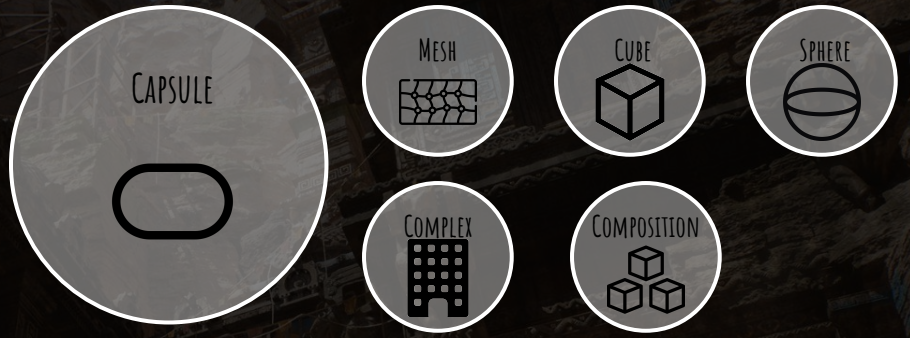
First **property** of a sphere is obvious the **center**, it is basically the **location** of the collider.

## Radius

From the center, we define a **radius** for the sphere, and it gives us the **final shape** that will be used in the physics simulation.

# Physics

1. Collision meshes
2. Body
3. Interaction
4. Traces
5. Detection
6. Matrix




## Definition

A **capsule** is a bit more complex than a **sphere** and a cube, but it actually can also be simulated by a cube. In essence, a capsule is the most suitable collision shape for humanoids.

## Center & Direction

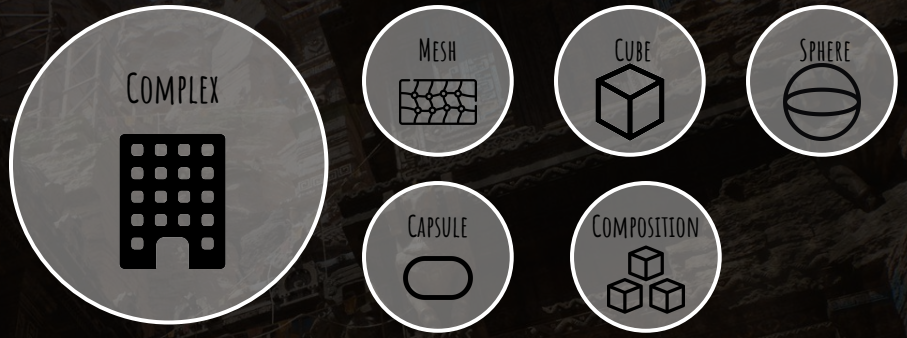
You define a **center** and a **direction** for a capsule. The center will be obviously the **collider position**, while the **direction** will dictate how **oriented** if the capsule.

## Height & Radius

Then, from the **center** and **direction**, you defines the **length** of the **capsule** (the **height**) and the **size** of the **capsule** (the **radius**). From there, you have the **final shape** that will be used.

# Physics

1. Collision meshes
2. Body
3. Interaction
4. Traces
5. Detection
6. Matrix

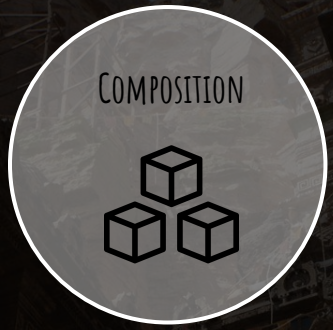



## Definition

There is multiple way to create a **complex collider** like said previously, with a **hull convex shape**, a **mesh collider**, **convex mesh collider**, etc... As a general rules, **never use them**.

# Physics

1. Collision meshes
2. Body
3. Interaction
4. Traces
5. Detection
6. Matrix









## Definition

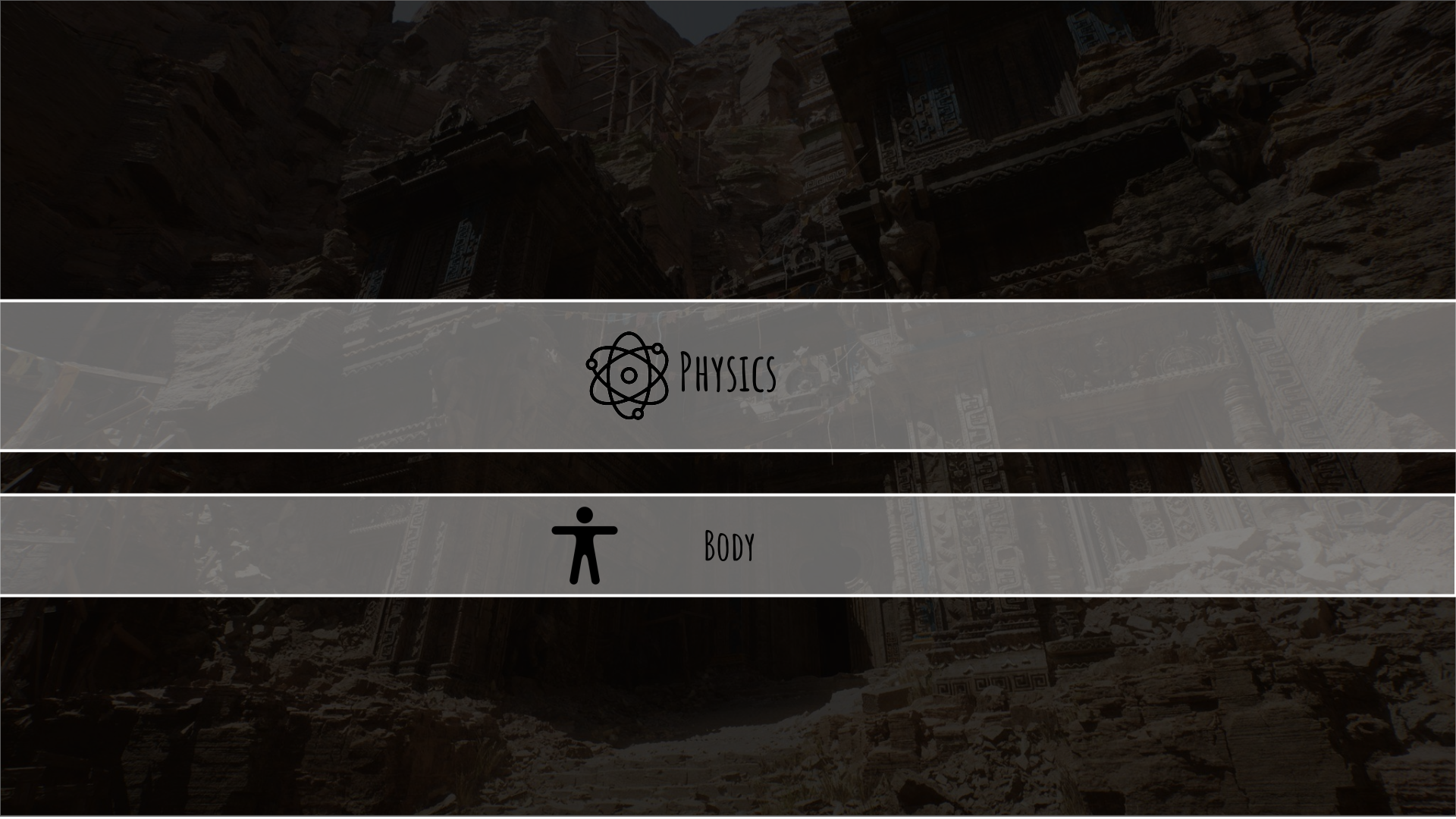
We've already talked a bit about **collider composition**. It is **absolutely necessary** to use that as much as possible. It will allows to have **finer collision detection** without having a complex collider.

## Performances

Even if you can think that **multiple collider** are **heavier** than a **simple** one, it is not. A **hull convex**, or even a **mesh collider** is extremely **expensive** on the simulation phase.

## Implication

On the implication side, you'll obviously needs to **properly defines** each **colliding area**. But is is also a **strength**, because you'll be able to defines **damage** from **area**, etc...


Physics
Body

# Physics







## Definition

A body has various **names** and **representation** over **different engine** but the idea is always the same. You apply **forces** on a **body**, which will be **processed** by the **physics engine**.

## Interface

A body is basically the **interface** you'll be **interacting** with in order to inform the physics engine of changes. You never directly call **physics engine code**, the body is the **proxy** with it.

## Forces and transform

When it comes to moving a body, you should **never use the transform directly**, as it will **teleport** the object and **bypass** the engine computation. Instead, **apply forces** to it with the various calls provided.

# Physics

1. Collision meshes
2. Body
3. Interaction
4. Traces
5. Detection
6. Matrix

Forces

Basis

Physical material

Parameters

## Definition

A force is basically a **vector** and a **strength**. You'll be feeding in which **direction** the force is applied, and the **quantity** of forces applied.

## Force type

You can distinct 2 **major force type** : **Force** or **impulse**. A force is supposed to be **continuous**, while an impulse you be **instantaneous**.

## Example

Let's say **FixedDeltaTime**=0.02 (FixedUpdate called 50x per second). **Strength** applied is 40 and gameobject **mass** is 2
Force - Strength * (Time) / Mass - 40 * (0.02) / 2 - 0.4
Impulse - Strength / Mass - 40 / 2 -20
Remark :20 = 0.4 * 50. We can conclude that addind a force in fixed update 50 times. So impulse is a instantaneous force

# Physics

1. Collision meshes
2. Body
3. Interaction
4. Traces
5. Detection
6. Matrix

Physical material

Basis

Forces

Parameters

## Definition

A **physical material** simply state how the object should **behave** regarding the **environment**. A **block of ice** will be **really different** than a **block of rock** on the **physic simulation**.

## Friction

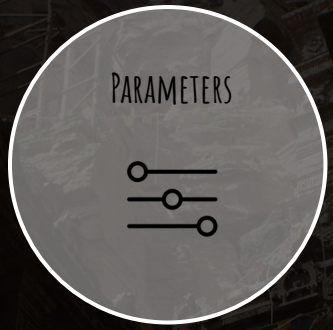
**Friction** is the 1st important element. It allows to configure how the object will be affected by **frictions**, which is basically **air**, other **surfaces**, etc...

## Bounciness

2nd important element : **Bounciness**. This allows to configure how the object will **interact** when it goes in **collision** with something. How the **force** will be **transfert**, how **re-apply** on the **body**.

# Physics









## Definition

There is plenty of parameter when it comes to configuring a body. We'll see in more details at the end of the lesson how to configure it on UNity.

## Mass

**Parameters** configuration vary based on engine but a general concept of **mass** will be present is everything. It is basically how **heavy** is your object in the **world**. It is an important **parameter** that you'll need to be **define** either **realistically** or based on the other **objects** in order to have a **coherence** in your **world**.


Physics
Interaction

# Physics

1. Collision meshes
2. Body
3. Interaction
4. Traces
5. Detection
6. Matrix

Basis

Overlap

Collide

Kinematic

## Definition

A **physic interaction** is something that will **trigger** an **event** when **two** objects are **interacting** with each others.

## Event based

As stated above, **collision** are **event based**, that's because we **cannot afford** to do a check on gameplay on **frame** to know if a **collision** happens. A method will be responsible for receiving the event

## Precision

When you play on **settings** in physics engine, you are actually affecting the collision detection, so the interaction indirectly. If you seen that there is incoherence of timing on the interaction.

# Physics

1. Collision meshes
2. Body
3. Interaction
4. Traces
5. Detection
6. Matrix





## Definition

An **overlap** is a **collision** event but within a **specific context**. An overlap is happening when there is no **real collision**. It means that the **objects** are passing through each one, but there is still an event

## Implication

Generally speaking, a **gameobject** is either **colliding** or **overlapping**. To get an overlapping event, at least 1 element should be on **overlapping** and **not colliding**.

# Physics

1. Collision meshes
2. Body
3. Interaction
4. Traces
5. Detection
6. Matrix





## Definition

A **collision** on the other hand will create a **real collision** with **bounces**, **forces**, etc... and an event. Objects will this time **not be able** to pass through.

## Implication

Just like the logic for **overlap**, the 2 **objects** that are **interacting** must be in **collision mode**. If one of another is kinematic or in overlap map, the collision will not happens, just like the event.

# Physics

1. Collision meshes
2. Body
3. Interaction
4. Traces
5. Detection
6. Matrix

Kinematic

Basis

Overlap

Collide

## Definition

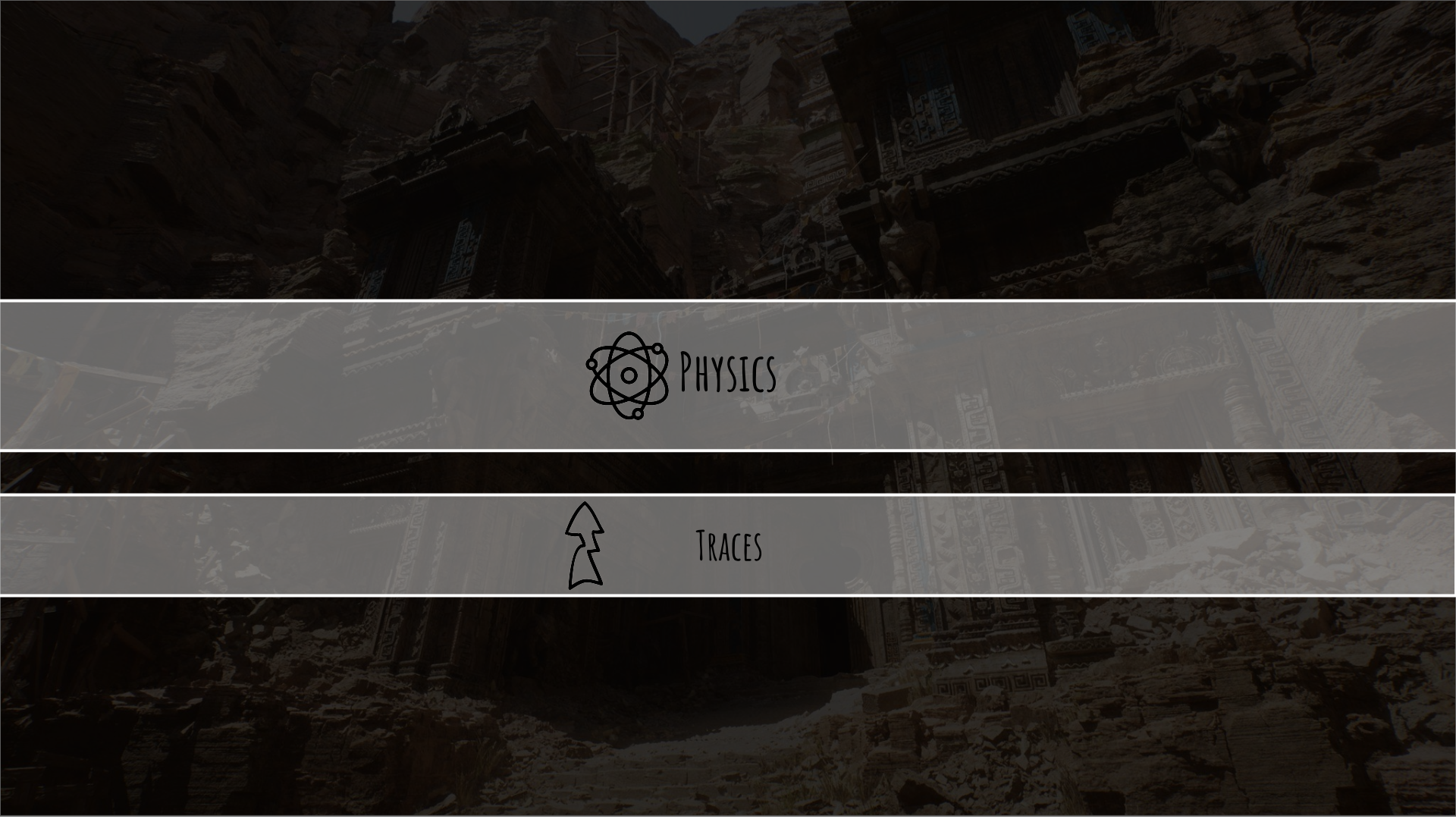
Sometimes, you **don't want the physics** to affect an object. The way of doing it depends from engine to engine, but the concept of **kinematic** is global. **Kinematic** is something **not affected** by **physics**.

## Implication

An object being kinematic means that it will not be computed at all by physics engine, so **no interaction** and **no event**. You must be **careful** when you are assigning kinematic to an object.

## Control

Being **kinematic**, you know have **full control** over the **transform** of the gameobject. It will now be controlled by the **animation** and the **.transform** you are setting in code


Physics
Traces

# Physics






## Definition

Before diving into the **specification** of a **trace**, the concept of **channel** is even more important. A **channel** is basically a way to tell for a trace **what will be blocking** that trace.

## Engine

There is **multiples channels** provided by **default** in **engines**. The most **common** one in all engine is the **visibility channel**. Basically, it is a channel that is consider **blocked** by **default** by **everything**

## Custom

Obviously, you'll be able to **create custom channel**, for the needs of your games. How to set them will be **per engine**, but the **concept** is that you'll be able to tell for each **object** if it **blocks**, **overlap** or **ignore**

# Physics

1. Collision meshes
2. Body
3. Interaction
4. Traces
5. Detection
6. Matrix




## Definition

We talked a bit about the **raycast** and the general concept is quite **simple**. A raycast is a **line** casted from an **origin**; in a **direction** with a **length**. All this parameters constructs the **raycast**.

## Single vs multi

An **important information** when you are **raycasting** is to know if you want a **single** or **multi result**. In **single**, the ray **will stop** at the **first blocking**, in **multi**, it will **continue** until the end

## Performances

**Raycast** are so **omnipresent** that it is something **heavily optimized**, but even with that, you must be **careful** when **raycasting**, especially when you want **UV information** which is impacting **performances**.

# Physics

1. Collision meshes
2. Body
3. Interaction
4. Traces
5. Detection
6. Matrix

Shapecast

Channel

Raycast

## Definition

From the **primitives** on **physics** (**cube**, **sphere**, **capsule**), you have in most **engines** a way to **shapecast** at **runtime** one of these **primitives**.

## Simple vs Multi

Even more important for **shapecast**, there is also the potential **issue** about the **shapecast stopping** at the **first blocking target**. In most cases, **shapecast** are used in **multi mode**.

## Performances

**Unlike raycast**, **shapecast** are really **expensive**. In most cases, you **are** better having a **collider around** an **object** to **detect things**, for example **aggro range**, **world interaction**, etc...


Physics
Detection

# Physics



Basis

Dimension

Discrete

Continuous

## Definition

**Detection** is the **major challenge** for a physics engine. It will be the **algorithm** running on a **frame basis** and **ensuring** to **detect** when 2 objects are interacting with each other.

## Collision boxes

**Physics engine** will only compute **detection** with the **collider boxes** that you've been **creating** on all object in your **world**. Obviously, **world physics** is **partitioned** in order to **not compute everything**.

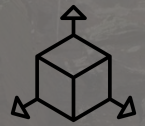
## Challenges

The topic is **challenging** because when a game is **dealing** with **fast moving objects**, the **detection algorithm** can be **misleading** or **missing collision**.

# Physics

1. Collision meshes
2. Body
3. Interaction
4. Traces
5. Detection
6. Matrix

## Dimension



Basis

Discrete

Continuous

### Definition

**Dimension** is really **important** when it comes to **complexity** for **collision detection**. That's also why some **engines** decides to create a **different physics framework** based on **2D** or **3D**.

### Differences

There is various **differences** between **2D** and **3D** but globally speaking, in **2D**, you simply need to check **overlapping** of objects in a **single plane**.**3D** is way **more complex** and requires more **structures**.

# Physics

1. Collision meshes
2. Body
3. Interaction
4. Traces
5. Detection
6. Matrix

## Discrete

Basis



Dimension

Continuous

### Definition

A first technique, which is way cheaper but also imprecise if **discrete detection**. It is the **default algorithm** used in most engine

### Principe

We don't need to go into **details** about how it works. **Discrete detection** means that it will **test** the **detection** on a **regular basis**, most likely at **each physics cycle**.

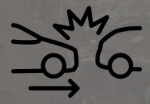
### Implication

Being a **detection per frame**, it means that it will be **inaccurate** on some cases, or even more, totally **miss collision** for **fast moving objects**.

# Physics



Continuous



Basis

Dimension

Discrete

## Definition

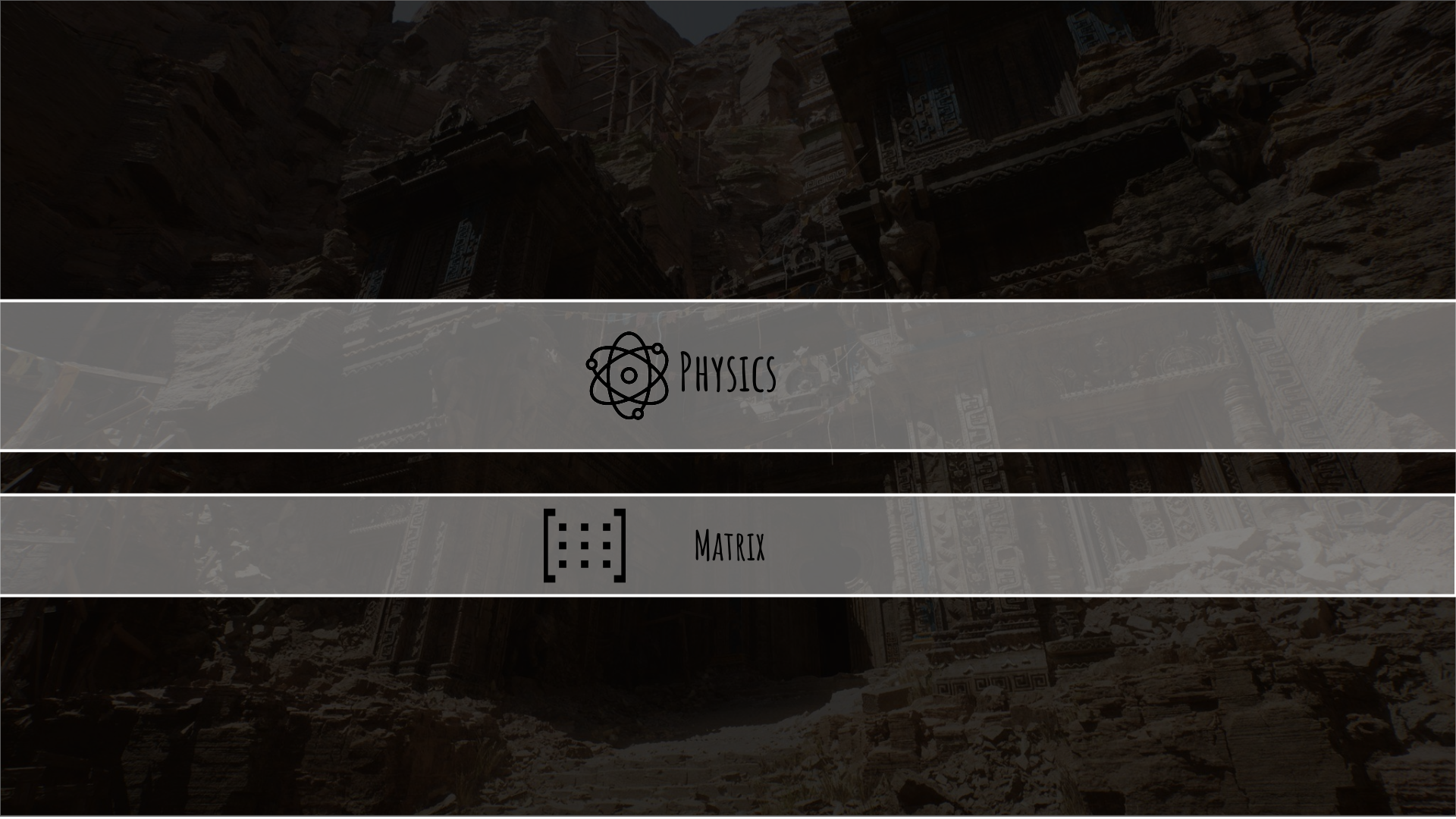
Second technique, which is way expensive but precise is if **continuous detection**. It is the collision detection used for FPS, action game, etc...

## Principe

Instead of detecting collision **per cycle**, **continuous detection** is done during all the **travelling** of the object.

## Implication

Being a **continuous detection**, it will be way more **accurate**, but also more expensive as the computation is done continuously.


Physics
Matrix

# Physics

1. Collision meshes
2. Body
3. Interaction
4. Traces
5. Detection
6. Matrix

## Basis

### Definition

**Implementation** of collision matrix is something really **specific** from engine to engine, so we'll just give the general **idea**.

### Principe

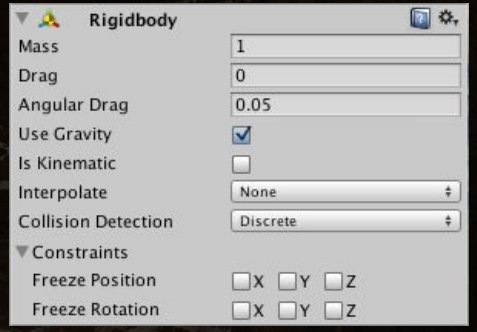
The concept is quite **simple**, the **collision matrix** allows users to **configure** how an **object type** is interacting with another **object type**. The same logic goes with **channels**

### Channel vs object type

A **channel** is used when it comes to **raytrace**. An **object type** is used for **object** themselves. It should not be **confused** are they are 2 **differents** aspects of the **physics interactions**.


UNITY

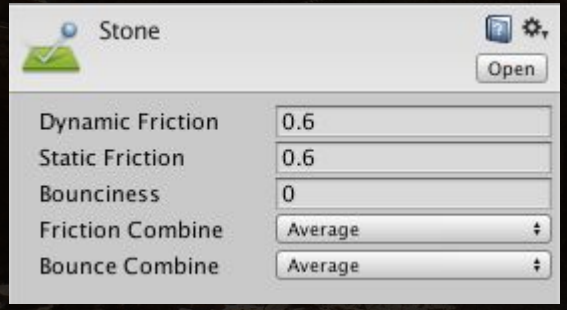
# Rigidbody




- **Mass** : Gameobject mass (In kilograms)
- **Drag** : How gameobject interact with air force while he's moving. A value of 0 means thair air has no effect. A big value will makes the gameobject stops immediately
- **Angular Drag** : How gameobject interact with air force while he's rotating. Value of 0 means that air has no effect. A big value will makes the rotation stops really fast, but never completely
- **Use Gravity** : Is gameobject affected by gravity ?
- **Is Kinematic** : Should the gameobject be excluded from physics calculation ? (PhysX)
- **Interpolate** : Physical interpolation about the movement. Use it if you see jerky movement.
  - **None** : No interpolation
  - **Interpolate** : Transform is move smoothly based on the last frame
  - **Extrapolate** : Transform is move smoothly based on the next frame
- **Collision Detection** : Used to specify the collision detection type to be used on this gameobject.
  - **Discrete** : Used by default, with discrete detection (Efficient but inaccurate)
  - **Continuous** : Used if you have issues to detect objects moving fast. (Accurate but costly)
  - **Continuous Dynamic** : Like continuous, but for multi-dynamic collision (accurate but really costly)
  - Comme le fonctionnement de Continuous, mais utilisé sur n'importe quel collider. Gros impact sur les performances.
  - **Continuous Speculative** : Like continuous, but less costly, but can be something inaccurate because of the speculative nature..
- **Constraints** : Restriction over the movement managed by physics engine.
  - **Freeze Position** : Constraints on position, so rigidbody cannot affect this field
  - **Freeze Rotation** : Constraints on rotation, so rigidbody cannot affect this field

- AddForce(Vector3, Forcemode)

# Physic Material




- **Dynamic Friction:** Friction used while the gameobject is moving. The value will most likely be between 0 and 1. A value near 0 will makes the gameobject looks like a block of ice, with little friction. A Value near 1 will makes the gameobject stop really fast except if the force is really big.
- **Static friction:** Friction used when the gameobject is not moving. The value will most likely be between 0 and 1. A value near 0 will makes the gameobject looks like a block of ice, really easy to move. A value near 1 will makes the game object really hard to move.
- **Bounciness :** How gameobject reacts to bounces. A value of 1 mean no bounce at all. A value of 1 will makes the object never lose energy from bouncing.
- **Friction combine :** How 2 gameobject colliding interacts and combines
  - **Average :** Takes the average of the 2 frictions
  - **Minimum :** Takes the minimum of the 2 frictions
  - **Maximum :** Takes the maximum of the 2 frictions
  - **Multiply :** Multiply the 2 frictions
- **Bounce Combine :** How 2 gameobject bouncing each others combines
  - Same logic as friction

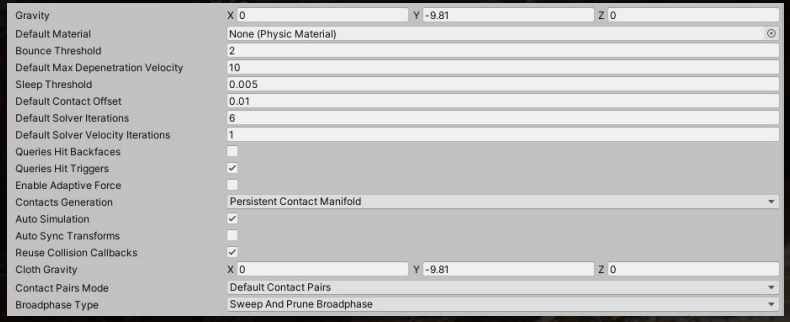
- Priority-wise combination are : Average < Minimum < Maximum < Multiply

# Collisions & Triggers

- OnCollisionX
  - OnCollisionEnter
  - OnCollisionExit
  - OnCollisionStay
- OnTriggerX : Possible with the IsTrigger checkbox.
  - OnTriggerEnter
  - OnTriggerExit
  - OnTriggerStay

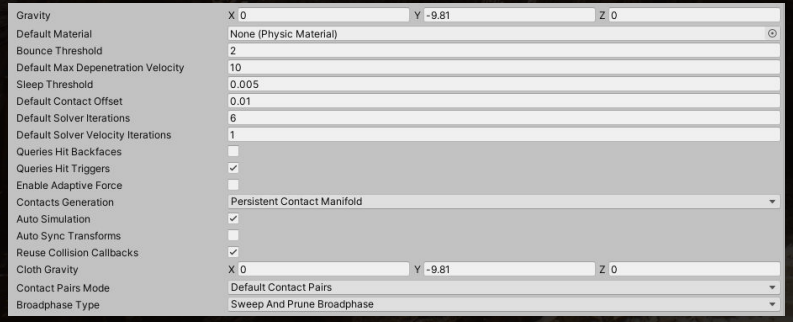
- Note
  - You must place the script with those callbacks on the same gameobject that contains the collider
  - If you want collisions, at least 1 object must has IsKinematic - false

# Physics settings - 1



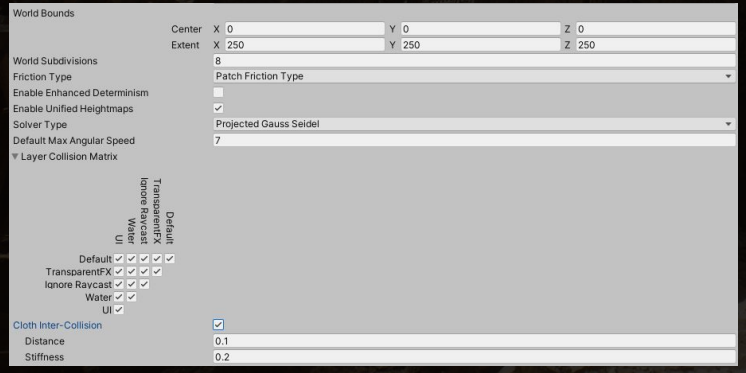

- **Gravity** : Defines for each axis how is the gravity force
  - Note : If you increase gravity, you may need to increase Default solver iterations.
- **Default Material** : Which physic material to be used if none is given for a collider
- **Bounce Threshold** : Defines a velocity. If 2 colliders has a velocity below that threshold, they will not bounce. 2, in most cases, is the minimum value to be used.
- **Default Max depenetration velocity** : Maximum velocity needed to move a collider outside a collision surface from another collider.
- **Sleep threshold** : Global threshold for energy. Below that threshold, a non kinematic gameobject will be asleep. A gameobject sleeping is not updated each frame by the physics engine
  - **Rigidbody energy / mass < Sleep threshold** - Candidate to fall asleep.
- **Default contact offset** : Defines the distance used to generate collision. A value near 0 could lead to jerky collisions.
- **Default solver iterations** : How many solver is used by physics engine
  - Solvers are tiny physics tasks. Increasing this number will allows to have clearer results but reduce performances.
- **Default solver velocity iterations** : How many velocity processes are performed each physics frame. The higher this number is, the higher the accuracy of the resulting exit velocity after a bouncing will be.
- **Queries hit backfaces** : Should backfaces of triangles be taking into physics requests ?
- **Queries hit triggers** : Should physics requests returns an hit when intersecting with a IsTrigger collider. Can be overridden at method's call.
- **Enable adaptative force** : Affects how forces are transmitted when there is a chain collision.
- **Contacts generation** : PCM - Nouveau projet, Legacy only for older versions
- **Auto simulation** : Physics is simulated automatically.
- **Auto sync transforms** : Synchronize transform with physics engine modification on transform.
- **Re-use collision callbacks** : Simply leave it to true, except if you plan to use the collision for later.
- **Cloth gravity** : Gravity apply to clothes

# Physics settings - 2




- **Contact pairs mode** : Contact type to be used
  - **Default Contacts pairs** : Collisions & triggers events except kinematic-kinematic & kinematic-static
  - **Enable kinematic kinematic pairs** : Collisions & trigger for kinematic-kinematic
  - **Enable kinematic static pairs** : Collisions & triggers events for Kinematic-static
  - **Enable all contact pairs** : Collisions & triggers events for everything
- **Broadphase type** : Which broadphase algorithm to be used
  - **SAP** : Useful when many physics objects are asleep. Performances will be really bad if a lot of gameobject are moving
  - **MBP** : Better performances overall than SAP, but worse if a lot of asleep gameobjects. Needs a worldbounds
  - **ABP** : Same as MBP with an automatic world bounds computation.

# Physics Settings - 3




- **World bounds** : Useful only with MBP. Define physical world with a grid
- **World subdivisions** : useful only with MBP. How many cell in the grid
- **Friction type** : Friction type during physics simulation
  - **Path friction type** : Most stable algorithm when iterator solver is low
  - **One directional friction type** : More expensive than PFT, but better results in some cases
  - **Two directional friction type** : Requires more iterator solver, but better results.
- **Enable enhanced determinism** : If actors are determinist, ensure that the simulation will be consistent.
- **Enable unified heighmaps** : Terrain collision uses the same way as meshes collisions.
- **Solver type** : PhysX solver to be used
  - **Project Gauss seidel** : Default solver
  - **Temporal gauss seidel** : Situational for some projects.
- **Default max angular speed** : Default max angular velocity (rotation) allowed on rigidbodies. Possible to be overridden per rigidbody
- **Layer collision matrix** : Collision matrix allowing to globally defines how layers interacts with each others.
- **Cloth inter-collision**
  - **Distance** : Diameter between each inter-collision particles. Ensure that spheres doesn't collides.
  - **Stiffness** : Forces of repulsion between inter-collision particles.

# Traces

public static bool **Raycast**(Vector3 origin, Vector3 direction, float maxDistance = Mathf.Infinity, int layerMask = DefaultRaycastLayers, QueryTriggerInteraction queryTriggerInteraction = QueryTriggerInteraction.UseGlobal);
public static RaycastHit[] **RaycastAll**(Ray ray, float maxDistance = Mathf.Infinity, int layerMask = DefaultRaycastLayers, QueryTriggerInteraction queryTriggerInteraction = QueryTriggerInteraction.UseGlobal);

public static bool **SphereCast**(Vector3 origin, float radius, Vector3 direction, out RaycastHit hitInfo, float maxDistance = Mathf.Infinity, int layerMask = DefaultRaycastLayers, QueryTriggerInteraction queryTriggerInteraction = QueryTriggerInteraction.UseGlobal);

public static RaycastHit[] **BoxCastAll**(Vector3 center, Vector3 halfExtents, Vector3 direction, Quaternion orientation = Quaternion.identity, float maxDistance = Mathf.Infinity, int layerMask = DefaultRaycastLayers, QueryTriggerInteraction queryTriggerInteraction = QueryTriggerInteraction.UseGlobal);

etc...

https://docs.unity3d.com/ScriptReference/Physics.html


Live demonstration


? Questions ?