VRUE Tutorial #2

13.10.17

Khrystyna Vasylevska

INTERACTIVE
MEDIA SYSTEMS
Vienna University of Technology

Topics

- Some organization remarks
- HTC Vive
  - How does it work
  - Setup and Calibration
  - Unity Integration
  - Unreal Integration
- What you need to know for Task #2
- Task #2

Assignments

- Implementing main task is **obligatory**!
  - Task #3 will be building on Task #1 and Task #2
  - Final project will build on Task #3

- Bonus is **optional**
  - Graded as some Extra points
  - Will be used to slightly improve the final grade, if the main task is not quite perfect
  - Make sure to indicate that you implemented bonus!
    - By submitting a screenshot and feature description txt file to bonus submission.
    - By adding "_bonus" to the name of your executable.

Submissions & Sharing

- Private repositories only! And at all times!
- Cheating will not be tolerated!

- Have troubles?
  - Go thought the slides of Tutorial #1 AGAIN!
  - Ask tutors!

- Task 1 submission deadline: **15.10.2017 23:55**
- Task 2 submission deadline: **27.10.2017 23:55**

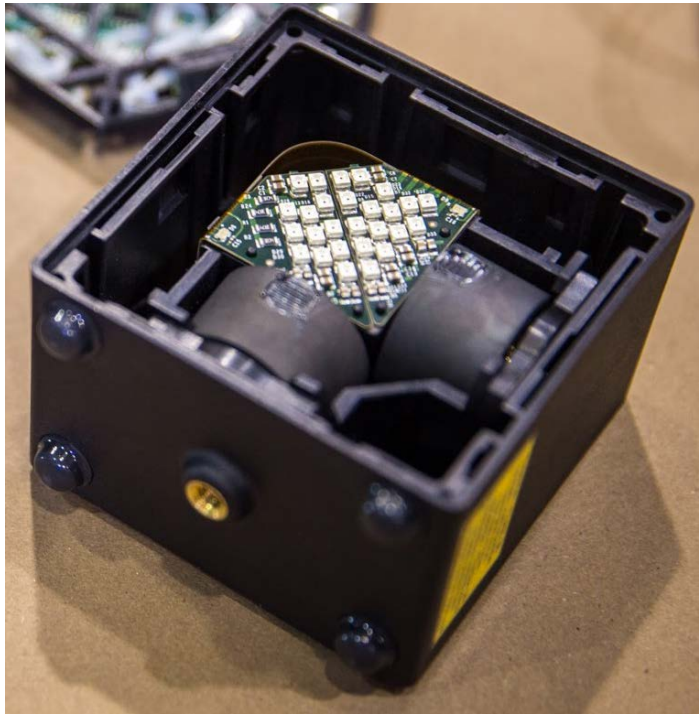
HTC Vive:
What's inside and how it works

INTERACTIVE
MEDIA SYSTEMS

Vienna University of Technology

LightHouse Base Station

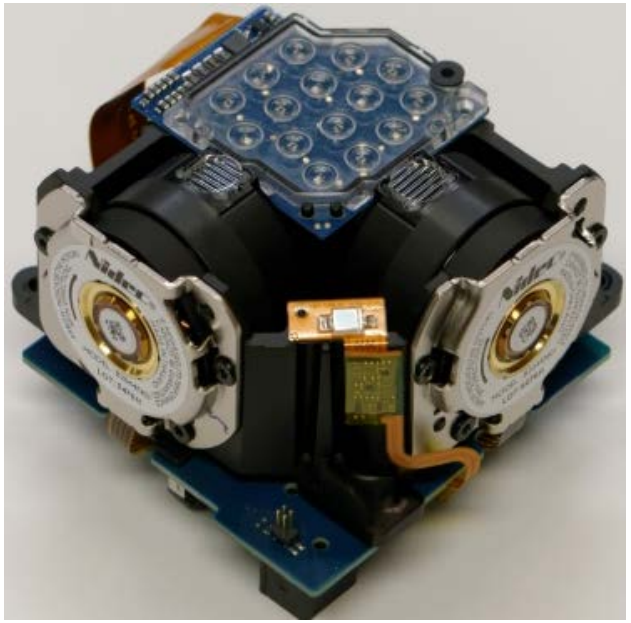
- Each base station contains two IR lasers with mirrors
  - One sweeps from bottom to top (horizontally, 8.333ms)
  - Second sweeps from left to right (vertical , 8.333ms)
- Lasers rotate at 3,600 rpm
- Each base station also contains a LED array
  - flashes a wide-angle synchronization pulse at the beginning of each 8.333ms laser sweep
  - to synchronize two base stations with each other
  - to give tracked devices a way to measure their relative angles to each base station



Source: http://www.pcper.com

LightHouse Syncronization

- LED array is used
  - to synchronize two base stations with each other
  - to give tracked devices a way to measure their relative angles to each base station
- Only be one laser sweeping the tracking volume at any time
- Four lasers of two linked base stations (A and B), are interleaved
  - A's two lasers sweep one after another and then B's two lasers
  - So LED array behaves like this
    - blink / X-sweep / blink / Y-sweep / blink / (none) / blink / (none)
    - See video https://youtu.be/avBt_P0wg_Y



Source: http://www.pcper.com

Do not drop!

INTERACTIVE MEDIA SYSTEMS
Vienna University of Technology

Tracking

- Any optical positional tracking has limits
- To have initial position and orientation (locked) a device must have:
  - at least 5 sensors ‘lit’ by a Base Station
  - or 3 if two Base Stations are in view
- Once locked, the device can use data from an IMU (Gyroscope & Accelerometer)
  - To update its location in real-time between the sweeps
  - Even if all sensors have been temporarily blocked from view

Vive Headset

Contains

- An eye relief adjustment
- An interpupillary distance adjustment
- Fresnel lenses
- AMOLED display panels
  - 91.8 mm diagonal, 447 pixel per inch
- A proximity sensor (is Vive on now?)
- A front-facing camera
- 32 photodiodes + IR filters on the outer shell
- IMU – Inertial Measurement Unit

Source: https://www.ifixit.com

Vive Controllers

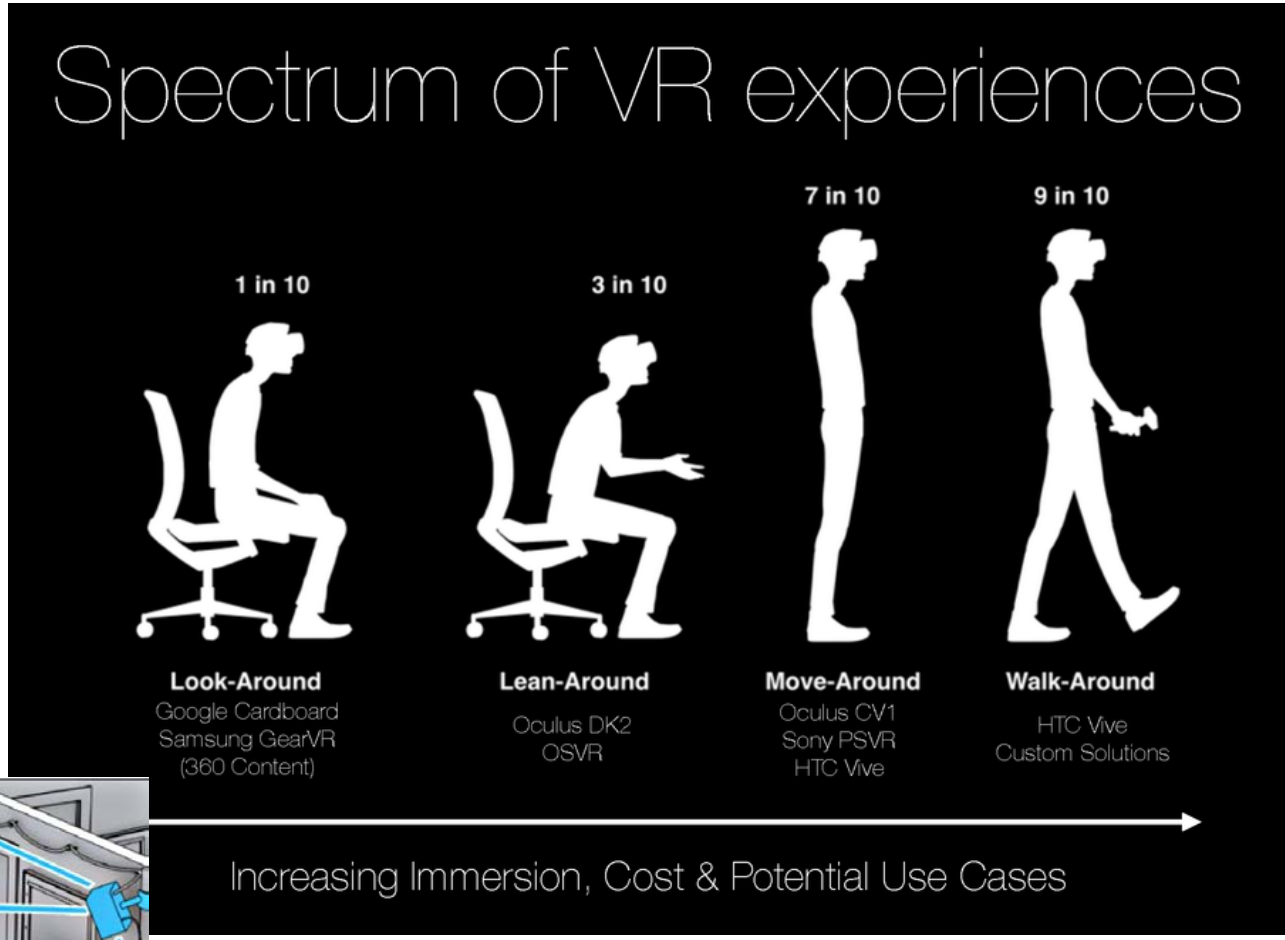
- Touchpad
- 2 small buttons (menu, On)
- Trigger button
- Grip button
- 24 sensors + IR filters
- IMU
- Battery

Sources:
http://www.pcper.com
https://www.ifixit.com

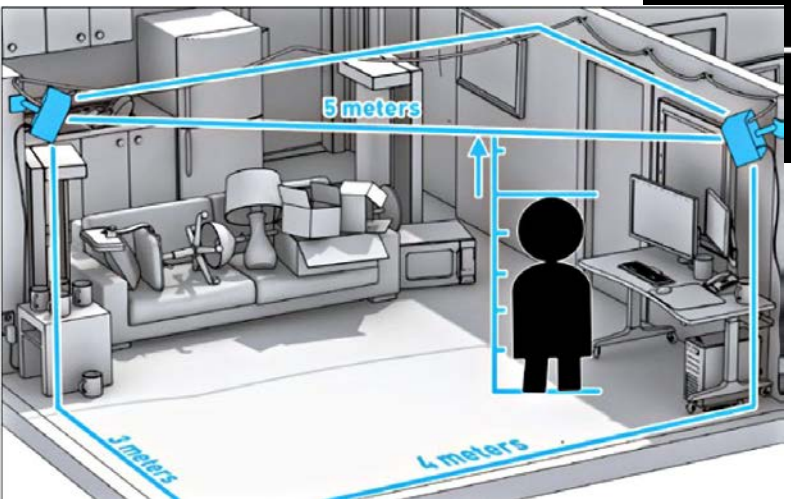
INTERACTIVE
MEDIA SYSTEMS
Vienna University of Technology

Possible Setups

- Seated
- Standing
- Walking



Source: http://www.bilawal.in

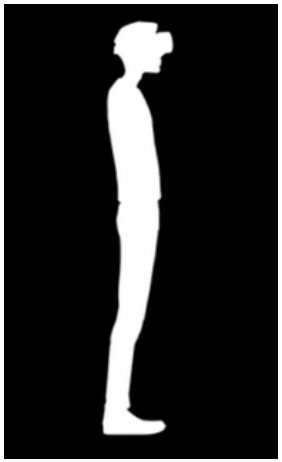


Limited Setups

Seated

- Assumes Origin in Player head
- Might be reset
  - in SteamVR via “VR Settings”
  - Using ResetSeatedZeroPose call through OpenVR API
- Calibrated Area: 1x1m

Standing



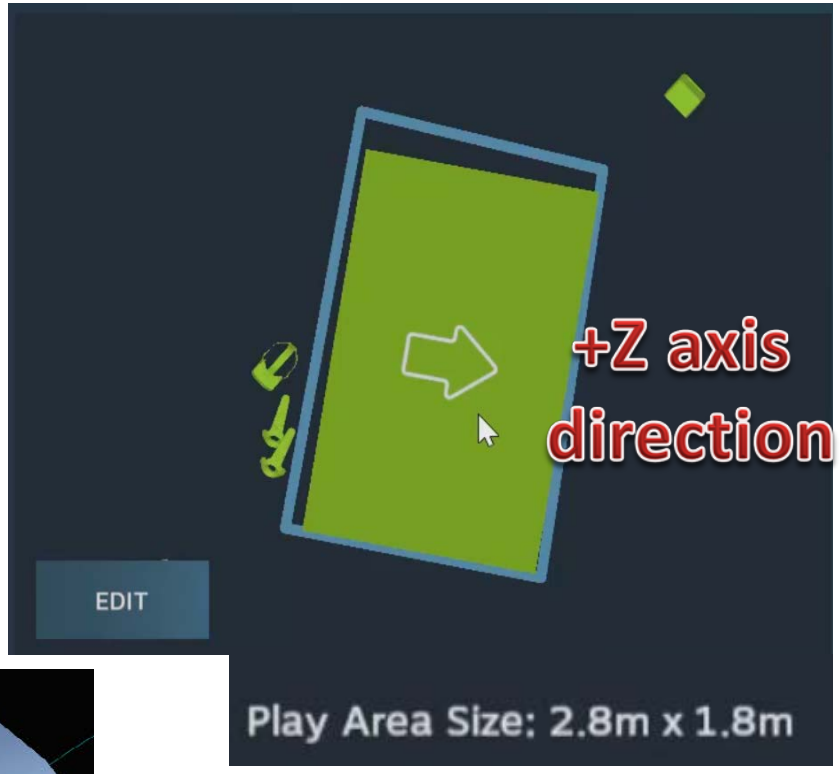
- Assumes Origin
  - centered in tracked volume
  - on the floor
- Tracked volume is set via Room setup

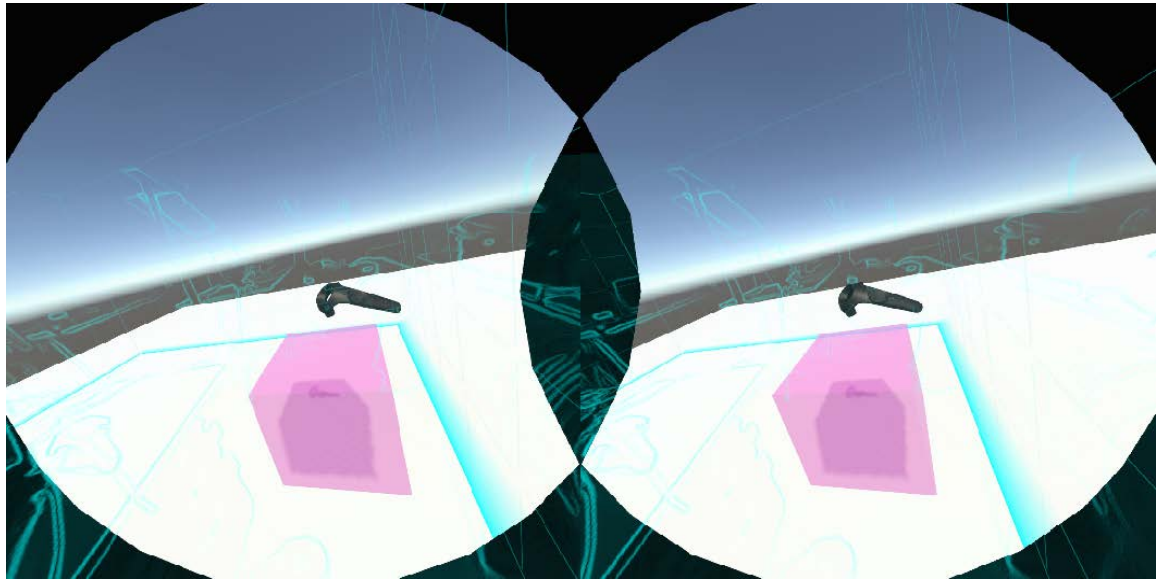
Installation

- Steam
  - http://store.steampowered.com/
- SteamVR and connect HTC Vive
  - see the installation Tutorial here http://www.htcvive.com/eu/support/
  - If something does not work as expected contact HTC Vive support and do as instructed before you report a problem to us
- Steam->Library tab
  - SteamVR can be found under Tools
    - Check it, if you can't find it try
      - Steam->Settings->Account
      - Beta participation -> Change-> Steam Beta Update
- Update the firmware when prompted
  - Base stations might require to do it via cable – check if it is needed before mounting the base stations

Rooms Setup and Calibration

- Room calibration
  - Use advanced mode if needed
- Note your play area size
  - Software doesn't allow areas exceeding 4.5x4.5m
- Chaperone is needed!
  - Camera view is controlled under camera settings in Vive menu

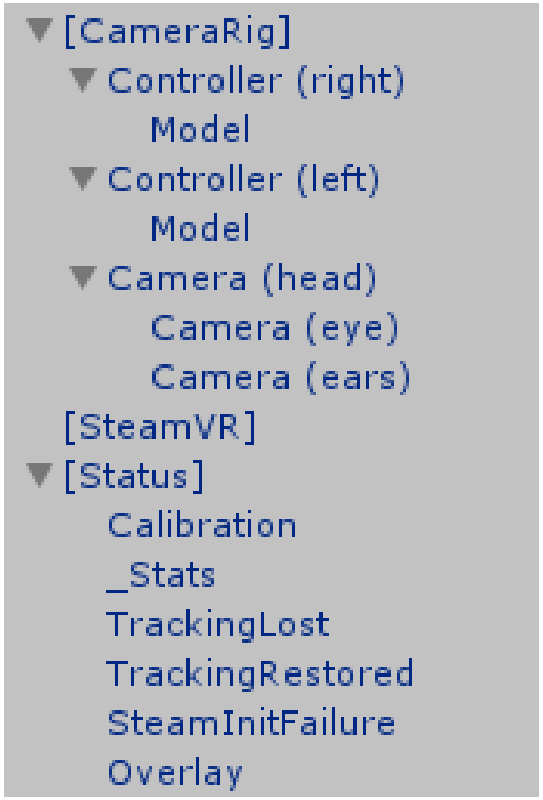





unity

Unity Integration

- Import Steam VR plugin to your project
  - Window->AssetStore
  - Search for SteamVR plugin
- Accept proposed Project settings



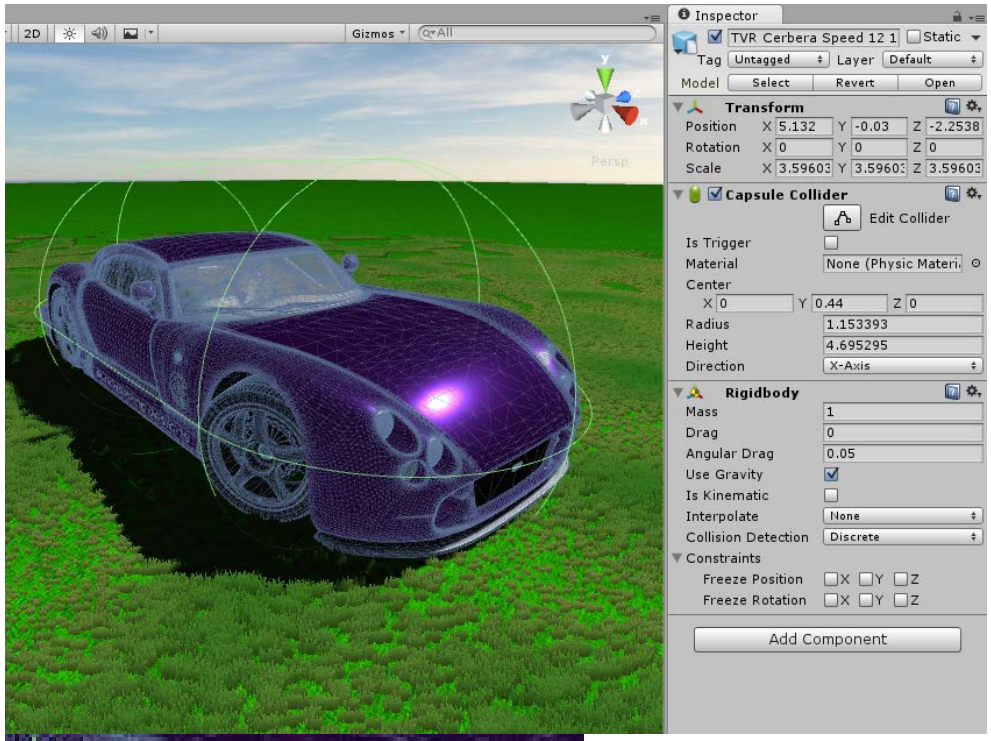
Prefabs

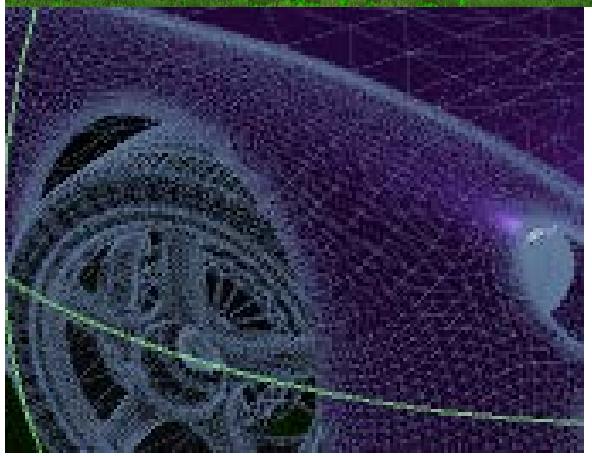
- Main - CameraRig
- Additional:
  - SteamVR
    - Configures game settings and setup (seated, standing, room scale)
  - Status
    - Calbigration tool and other statistics

Vive Prefabs

- [CameraRig] - origin
  - Defines and represents the play area
    - Steam VR_Controller Manager
    - Steam VR_PlayArea
- Camera (head)
  - Represents head motions in tracked volume
  - Renders screen view (not HMD View)
    - SteamVR_Tracked Object – Index = Hmd
- Camera (eye)
  - Follows the head motions
  - Renders HMD view (slightly different settings)
    - Steam VR_Camera
- Controller (left/right)
    - SteamVR_Tracked Object
  - Model
    - Steam VR_Render Model

Physics Basics Review





Collider

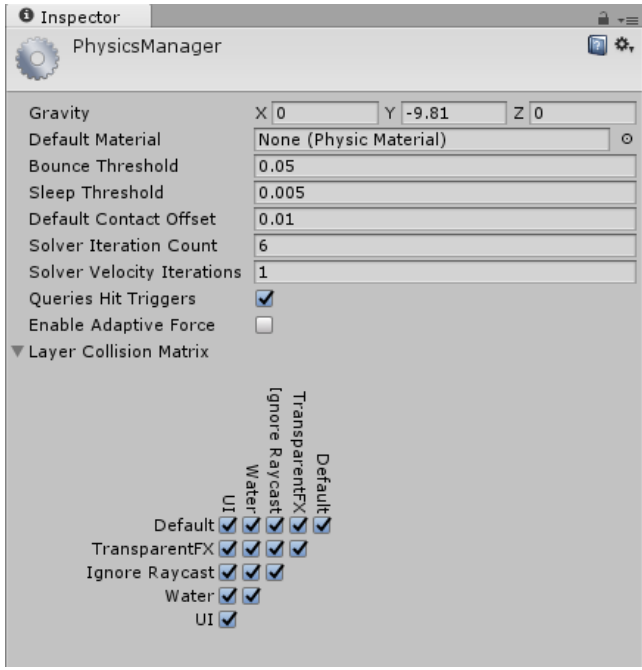
- Component that reacts on other Colliders
- Collider can be a trigger

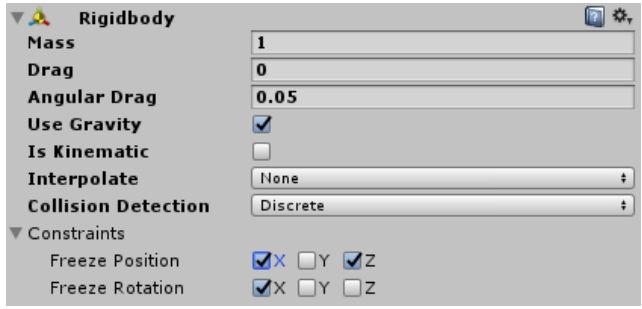
Rigidbody

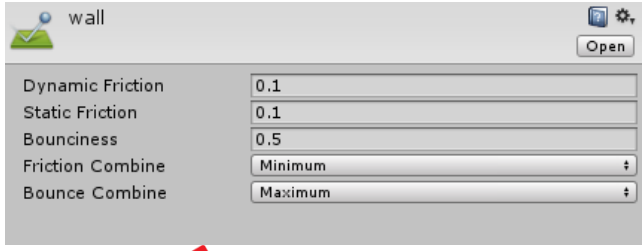
- Component that object to be effected by physics
- Must have a `Collider` Component to interact with other physics objects
- `isKinematic` disables physics interaction for the object
  - But it still affects the other objects

Physics Parameters

- Physics settings
  - General settings for the project
    - Edit->Project Settings ->Physics
  - For each individual object
    - Rigidbody properties
  - For a group or specific type of objects
    - Physics Material
      - Describes the surface/material physics
      - Controls how an object reacts to a collision







INTERACTIVE
MEDIA SYSTEMS
Vienna University of Technology

Physics & Interaction

- Physics collision needs
  - 2 Colliders
  - At least 1 Rigidbody
  - Physics Engine will use parameters of each Rigidbody and Physics material to calculate the right response to collision
- What if there are no second Rigidbody or it isKinematic, but you need some extra response?
  - you use AddForce(Vector3 force, ForceMode)
    - ForceMode uses mass (or not – see documentation)
      - Force
      - Impulse
      - etc.
  - Or you assign the response ignoring the physics engine
    - Rigidbody.velocity
    - Rigidbody.AddTorque()

    Note: ignoring the physics might cause unnatural behavior

Ways to Implement Interaction

- Use Collider.isTrigger
  - allows to detect collision (touch), but ignores the Physics of it
  - Trigger also calls for OnTriggerEnter(Collider) also [Stay, Exit] events
  - To ensure that you can control hold/release actions you will also need another condition, like is some button pressed or not.

Attachment of the GameObject to the controller:

- Parenting
  - Nest the object to the controller using transform.parent
- Track Position
  - Match the positions in Update()
  - Similar to parenting, but without breaking the hierarchy
- Physics Joint
  - Fixed Joint component relies on physics and setup parameters
  - You can setup Breaking Force and/or Torque

Vive Controller Class

`SteamVR_ControllerManager` from [CameraRig]

- Handles the controllers rendering, indexing, connections

`SteamVR_Controller`

- Handles input events
- defines :
  - `ButtonMask`: ApplicationMenu, Grip, Axis0-4, Touchpad, Trigger
    - Analog enum EVR`ButtonId` defined in openvr_api.cs
  - `Device` class has following properties and methods:
    - index, connected, hasTracking, outOfRange, uninitialized, etc.
    - bool GetPress(ulong buttonMask)
    - bool GetTouch(ulong buttonMask)
    - TriggerHapticPulse(duration in MicroSec, EVRButtonId)

Accessing Vive Controller Data

- Controllers in Unity have only `SteamVR_TrackedObject` component
  - `SteamVR_TrackedObject.index` – is the controller's deviceID
    - It changes every run and sometimes during run-time
    - New devices are dynamically added or removed
    - Reference to the component doesn't change and provides access to the updated index
- To get to the instance of the `Device` class
  - Use `SteamVR_Controller.Input(int deviceID)`
    - And from there you get your input and output (vibrations)
- Good practice:
  - Do not call `GetComponent<T>()` every frame, get it in `Start()` or `Awake()` once!
  - Similarly, avoid frequent `GameObject.Find()` calls, if possible.

Useful Things: Execution Order

- Awake
  - always called before any Start functions or just after a prefab is instantiated
  - Called even if Script is disabled!
- Start
  - is called before the first frame update
  - only if the script instance is enabled
- Update
  - Called based on frame rate
- FixedUpdate
  - time based and used typically for physics and other time related things
  - is often called more frequently than Update, fits only time dependent events
  - DO NOT use it instead of Update or there will be strange behavior!

There are also other functions. For further information see
https://docs.unity3d.com/Manual/ExecutionOrder.html

Other Useful Things

- To check what your user is seeing
  - SteamVR app -> Dropdown menu - Display Mirror
- To check what's around you
  - SteamVR app -> Dropdown menu Settings -> Camera -> Allow Camera for Chaperone Bounds Will show you the edges of surrounding objects.
    - Beware, that the scale is not quite right relative to the virtual world.
  - In game: Menu -> Settings -> Chaperone will allow you to change the grid view and color.
- Timing (Optional)
  - Edit->Preferences->Time
    - Default Fixed TimeStep = 0.02
    - If set Fixed TimeStep = 1/90 = 0.01111111 physics updates will match the Vive update rate
    - Disadvantage: Higher CPU usage

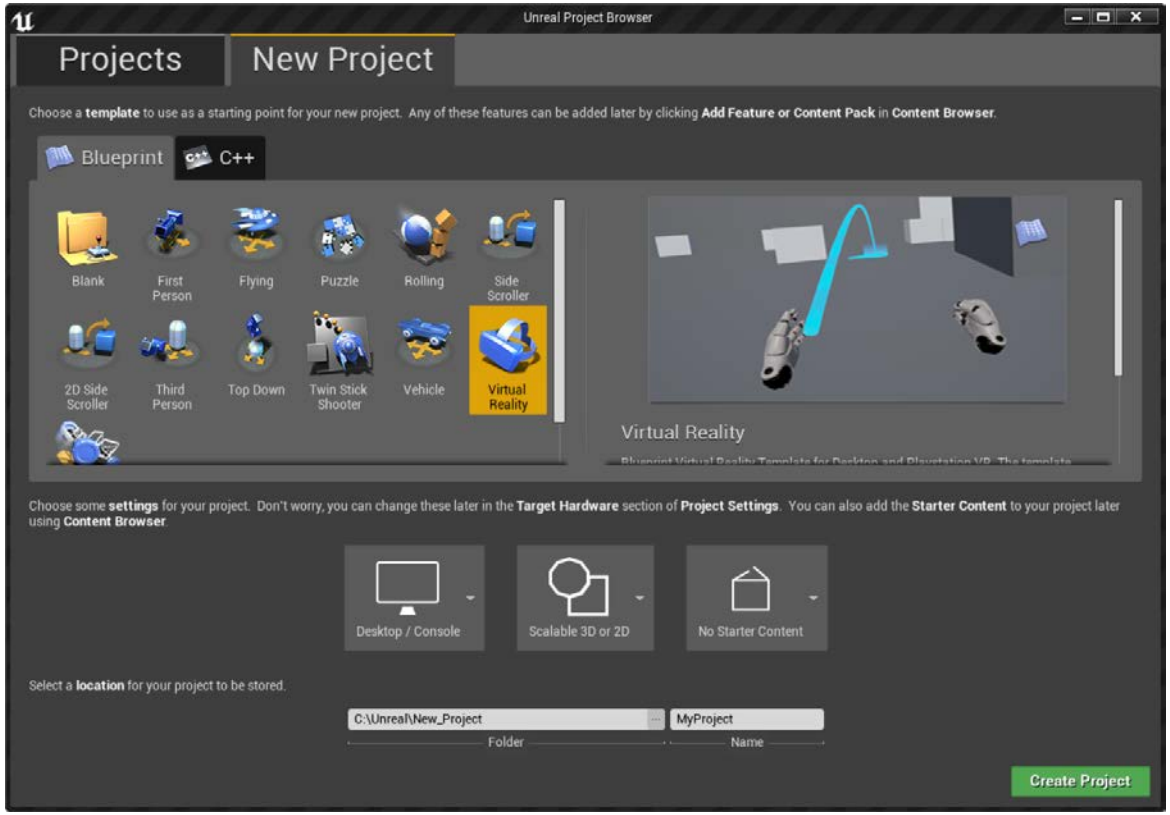
[Unity]Where to Look for Help

- SteamVR Developer Important Links and Documentation
  - https://steamcommunity.com/app/358720/discussions/0/613956964584902849/
- SteamVR at Valve Developer Community
  - https://developer.valvesoftware.com/wiki/SteamVR
- SteamVR Developer Forum
  - http://steamcommunity.com/app/358720/discussions/
- HTC Vive SteamVR SDK
  - https://www.htcdev.com/devcenter/opensense-sdk/htc-vive-steamvr-sdk
- OpenVR API Documentation
  - https://github.com/ValveSoftware/openvr/wiki/API-Documentation
- Unity Documentation
  - https://docs.unity3d.com/Manual/index.html

UNREAL
ENGINE

Unreal Integration

- Create a new Virtual Reality Project
  - Select HMD+ Motion Controller after creating Project
  - File->Open Level-> VirtualRealityBP-> MotionControllerMap for HTC Vive Test Environment



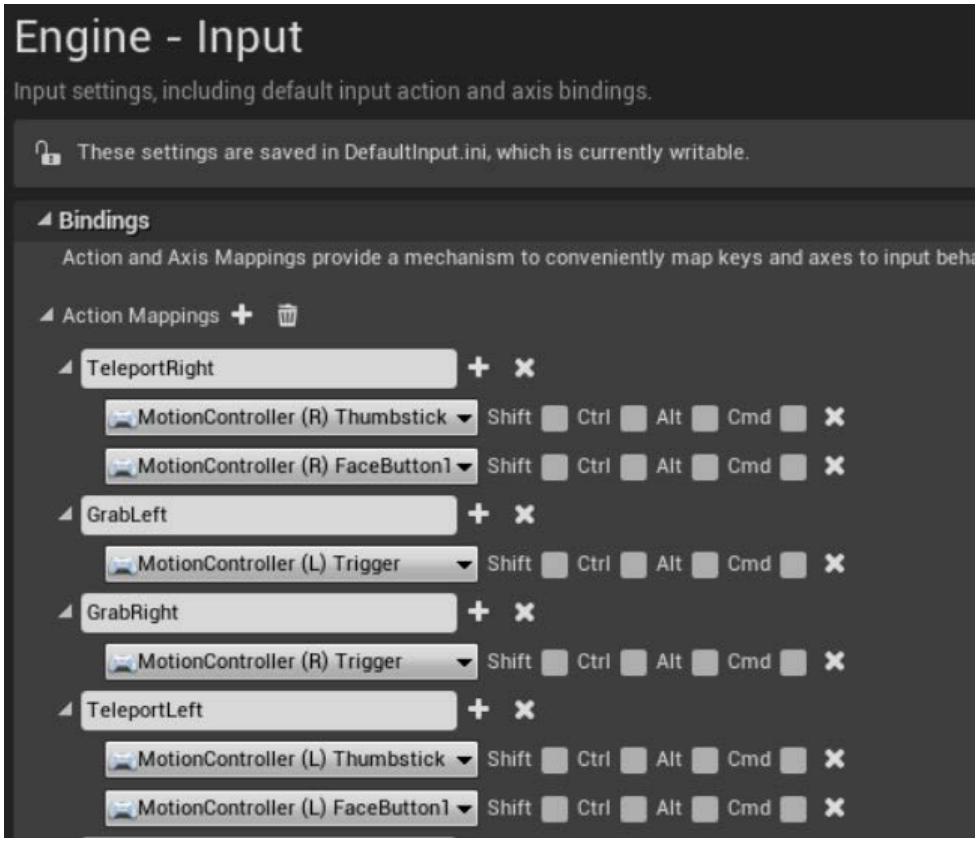
VR Pawn

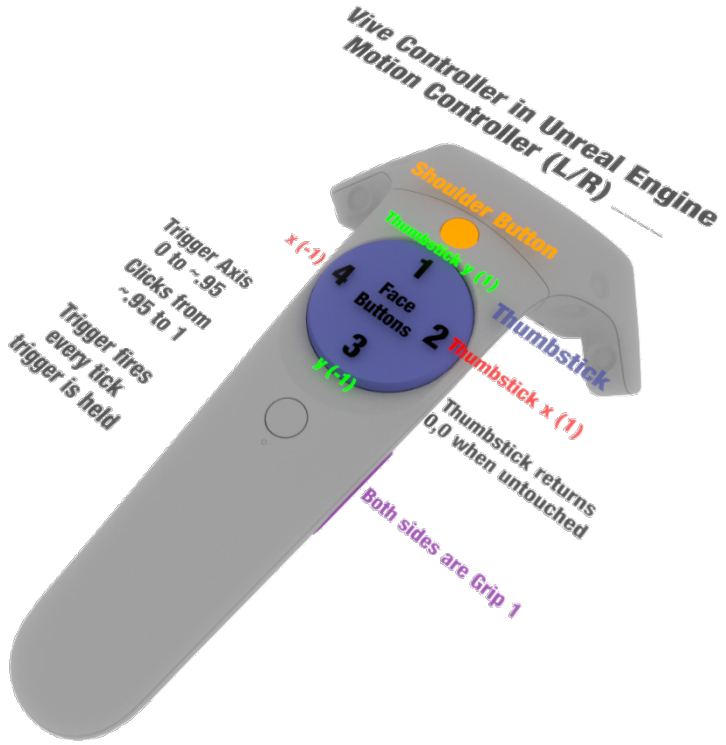
- VRPawn - origin
  - Defines and represents the Player (HMD)
- Controller (left/right)
  - Both Controllers are **spawned** from VRPawn (MotionControllerPawn in Content browser) in the Blueprint Event Graph
  - Controllers are attached Actors
  - Controller Behavior can be found in the Event Graph of the BP

VR Template info: http://www.tomlooman.com/vrtemplate/

Vive Controller Input

- **Settings-> Project Settings->Input ->Bindings**
  - here add Input Bindings for specific Buttons and Axes (Touchpad)
  - define the input keys in the settings first then you bind them in C++ to a method





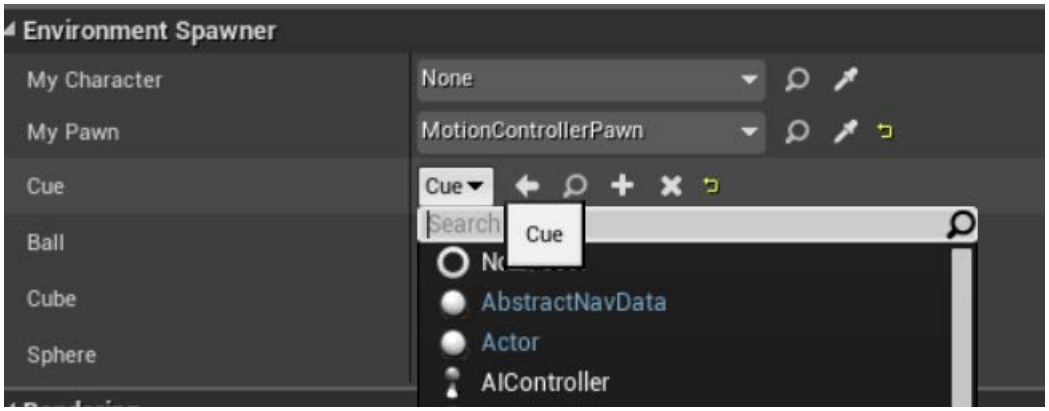
More info
https://docs.unrealengine.com/latest/INT/Programming/Tutorials/PlayerCamera/2/index.html

Accessing Vive Controller Data

- Using the mappings in C++
  - Bind the defined Action to a Method so the Method is called as soon as the Input Event occurs
    - `InputComponent->BindAction(“GrabLeft", EInputEvent::IE_Pressed, this,&AMyClass::AMyMethodToBind);`
  - For Axis(Touchpad) use this
    - `InputComponent>BindAxis("LeftTriggerAnalog",this,&AMyClass::AMyMethodToBind);`
  - Always check your Mappings in the **Settings** so they are assigned!

Exposing Objects in the Editor

- You can expose Data fields in your C++ scripts in your editor by using
  - UPROPERTY
  - Different Parameters can be given for different results:
    - `VisibleAnywhere`
    - `EditAnywhere`



- Example:

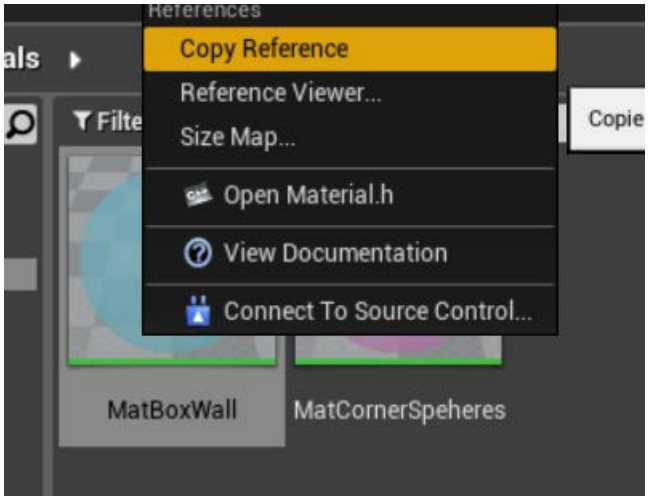
```
UPROPERTY(EditAnywhere)                    <- One line specificator
TSubclassOf<class AActor> variable;
```

- Properties Documentation for more information

VR Template info: http://www.tomlooman.com/vrtemplate/

Loading Resources from Disk

- You can acess different kinds of Resources directly from the **Content Browser** inside C++
- Example for a pre-created Material
  - `staticConstructorHelpers::FObjectFinder<UMaterial> MyMaterialName(TEXT("Material'/Game/Assets/Materials/MyMaterialName.MyMaterialName'"));`
- Loads the Material at runtime
- You can get the Path to the Resource by right-clicking it in your **Content Browser** and selecting **Copy Reference**



INTERACTIVE MEDIA SYSTEMS
Vienna University of Technology

Spawning Actors in the World

- You can spawn Actors into the world by calling this Method in C++
- `GetWorld()-> SpawnActor<AActor>(myActor, FVector location, FRotator rotation, FActorSpawnParameters params);`
- This Method returns an AActor* pointer
- Only Actors can be spawned like this. You can’t spawn Components (Meshes etc.) directly!

Organizing and Parenting

- You can organize Actors in the **World Outliner** by putting them in **Folders**

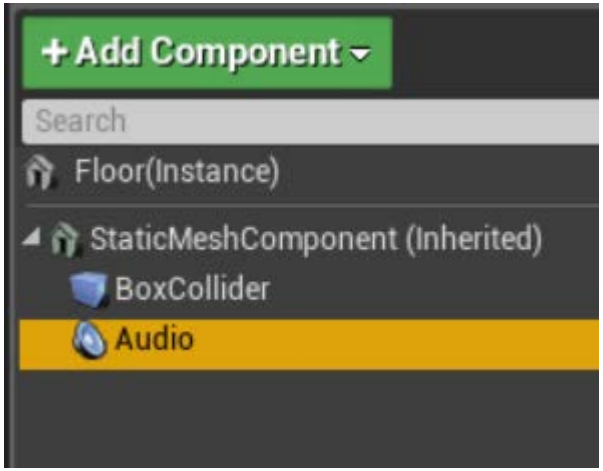
```
spawnedActor->SetFolderPath("/folderName");
```

- You can also parent AActors together
  - but not if they have their own Physics enabled. Use Folders instead.

```
childActor->AttachToActor(parentActor,
FAttachmentTransformRules::KeepRelativeTransform);
```

Components

- **Components** are a special type of Object designed to be used as **sub-objects** within **Actors** only
- Each specific Component has it‘s unique parameters and purpose
- You can create your own Components in C++
  - by defining a C++ (Component) class
  - use this Component instead of separate Actor
- And reuse this component later on other Actors as well



[Components Documentation]

Instantiating Components

- Components can be instantiated in a script and attached to an Actor.

```
UStaticMeshComponent* SphereVisual =
CreateDefaultSubobject<UStaticMeshComponent>(TEXT("VisualRepresentation"));
SphereVisual->SetupAttachment(RootComponent);
```

- After creating a Component you can load the data for it (Mesh in this case).

```
static ConstructorHelpers::FObjectFinder<UStaticMesh>
SphereVisualAsset(TEXT("/Game/StarterContent/Shapes/Shape_Sphere.Shape_Sphere"));
    if (SphereVisualAsset.Succeeded()){
       SphereVisual->SetStaticMesh(SphereVisualAsset.Object);
    }
```

Getting Components from Actors

- You can get Specific types of Components by creating a `TArray` and calling `GetComponents()` on an Actor

```
TArray<UStaticMeshComponent*> meshes;
myActor->GetComponents(meshes);
meshes[0]->SetMaterial(0, myMaterial);
```

- You can use this Method with different types of Components. The result depends of the `TArray` type.

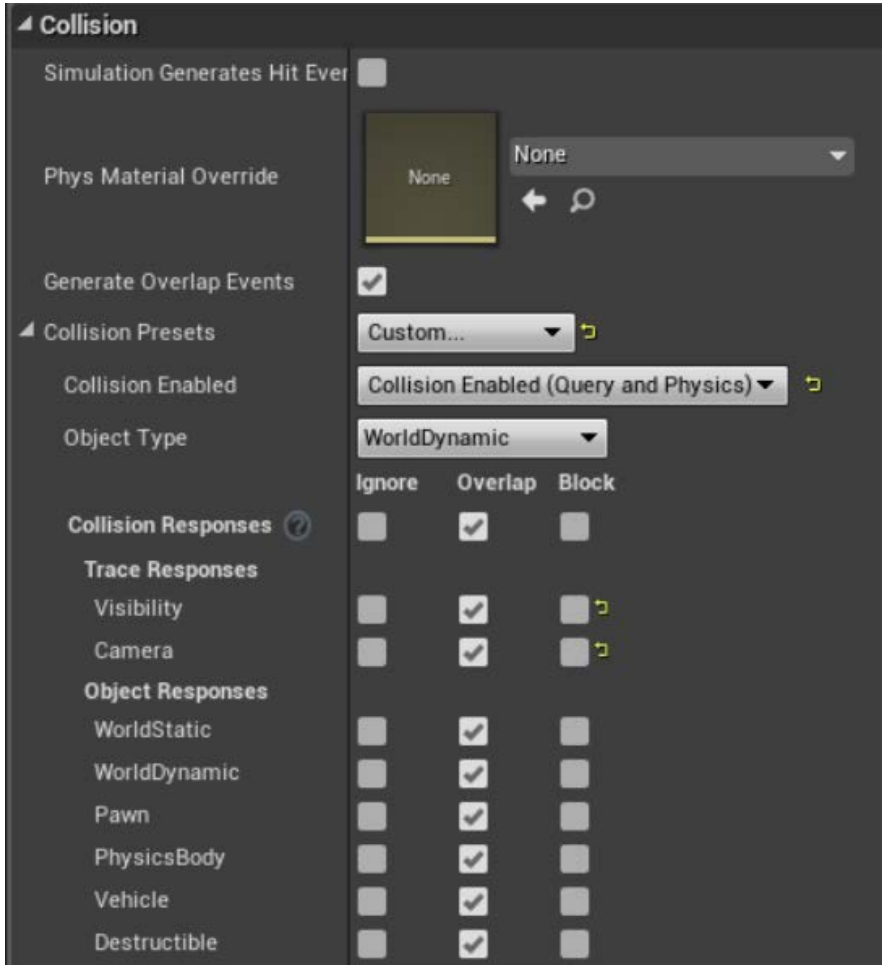
- You can also get an instance of a specific Component by calling `FindComponentByClass` on an AActor

```
UStaticMeshComponent* m = GetOwner()-
>FindComponentByClass<UStaticMeshComponent>();
```

Physics Basics Review

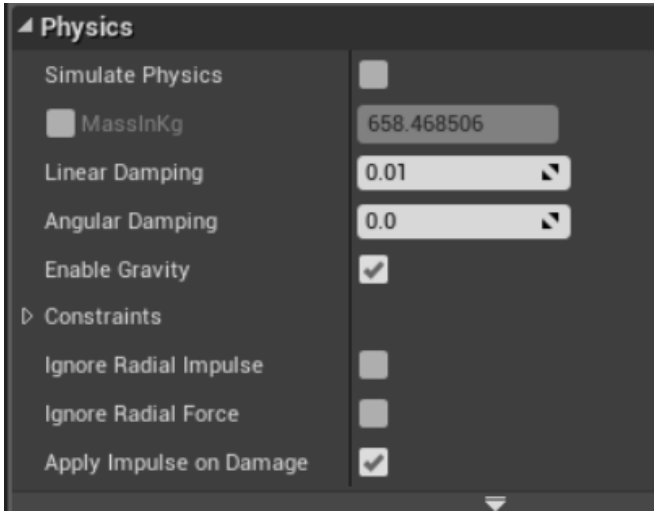
Collision

- Defines what the Actor **can collide with** and how the specific Collision should be handled by the



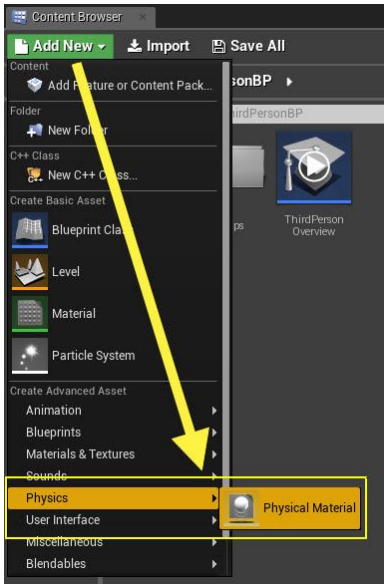
Physics

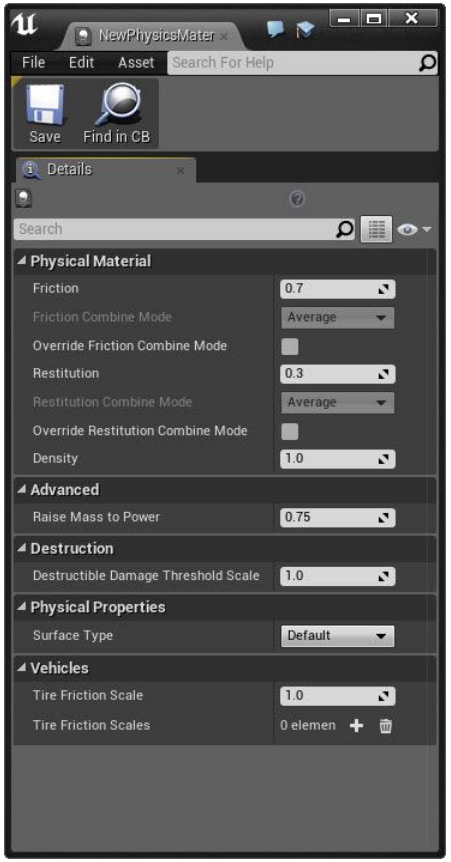
- Defines all **parameters** how the Actor is affected by physics



Physics Materials

- Physics material
  - Is used for a group or specific type of objects
  - Describes the surface/material physics
  - Controls how an object reacts to a collision





Trigger Events

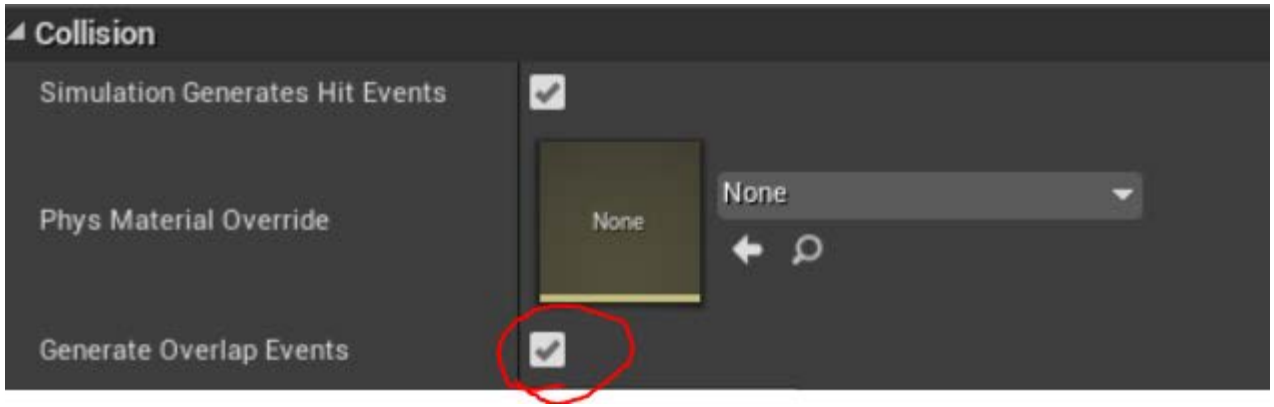
- You can add a dynamic Eventbased Method to a Component that is able to ***Generate Overlap Events*** (for exmple a StaticMeshComponent)

```
meshComponent->OnComponentBeginOverlap.AddDynamic(this,
&myClass::myMethod);
```

OR

```
meshComponent->AddForce(FVector::RightVector * 1000,NAME_None,
true);
```

- This Method will be trigerred as soon as an overlap Event happens



Checking for Overlapping Actors

- You can get all Overlapping Actors that overlap in a Volume by calling

```
TArray<Aactor*> overlappingActors;
overlappingVolume->GetOverlappingActors(overlappingActors);
```

  - This will Return a `TArray` of Actors that overlapped the volume. You can just iterate with a for loop over them to perform some operations on the actors.
  - Needs to be called every frame

Volume reference: https://docs.unrealengine.com/latest/INT/Engine/Actors/Volumes/

Useful Things: Execution Order

- **BeginPlay (Unity -> Awake)**
  - Overridable native event for when play begins for this Actor.
- **PostLoad (Unity -> Start)**
  - is called by serialized Actor after they have finished loading from disk
- **Tick (Unity -> Update)**
  - Function called every frame on the Actor
- **EndPlay**
  - Called in several places to guarantee the life of the Actor is coming to an end
- **Destroy**
  - is called manually by game any time an Actor is meant to be removed, but gameplay is still occurring

Also there are other functions.

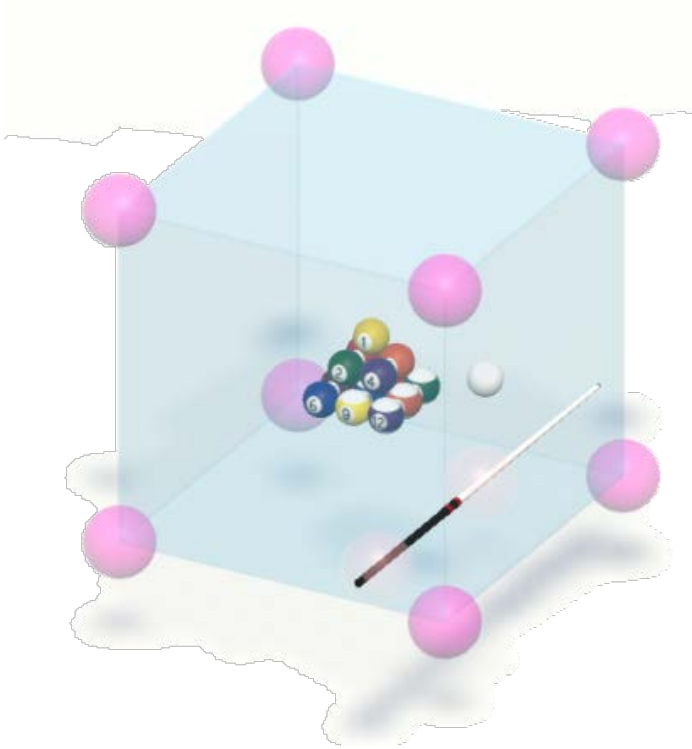
For further information see Actor Documentation or Actor Lifecycle

[Unreal]Where to Look for Help

- VRTeamplate Guide
  - http://www.tomlooman.com/vrtemplate/
- VRTemplate Getting Started
  - http://www.tomlooman.com/getting-started-with-vr/
- Accessing HTC Vive controllers from C++
  - https://docs.http://jonaskunze.com/accessing-htc-vive-controllers-from-c-in-ue4-4-13-and-4-14/.com/latest/INT/
- Unreal Engine Community Wiki(Tutorials and Reference)
  - https://wiki.unrealengine.com/Main_Page
- Unreal Engine Documentation
  - https://docs.unrealengine.com/latest/INT/

Task 2

3D Zero-Gravity Billiard



- More scripting! (almost no blueprints!)
  - Procedural generation of the gaming environment using scripts
  - Use primitives/meshComponents and a few prefabs/Actors
  - Materials
- Interaction
  - Hand interaction using Vive controllers
  - Physics interaction between different virtual objects with zero gravity