VRUE Tutorial 3

Networking and Distribution
31.10.2017

Iana Podkosova

INTERACTIVE
MEDIA SYSTEMS
Vienna University of Technology

Organizational

- ✓ Assignment 1 — Basics/Leap Motion
- ✓ Assignment 2 — Physics/HTC Vive
- ✓ Assignment 3 — Networking and Distribution
  - due 21.11.2017
- ✓ Project idea
  - due 10.11.2017
- ✓ Project — VR Game based on all above
  - Mid-January 2018

Project Task

- VR game for 2 players
  - Can also be a serious game or a simulator/educational application
- Compulsory features (you have to implement them)
  - Leap player creates different objects that are picked up and used by both players
  - Throwing objects (correct and correctly synchronised physics)
  - LeapPlayer can move (but both players always see each other)
  - Interactive game space extention (see next slide): 2 methods
- Short description to be submitted on **November 10th**
  - Not graded but **obligatory**
  - Text file describing gameplay and abilities of each player

Game Space vs Tracking Space

- Tracking space is limited to a max of 5x5 meters
  - Game worlds are usually much bigger
- How to explore a large game world using a very limited tracking space?
- Teleportation? Magic portals?
  - Can provoke simulator sickness if used frequently in an HMD-based application
  - Can easily provoke disorientation
  - Breaks the feeling of the continuity of the motion
- Alternative: manipulate tracking data and/or the environment

Game Space Extention

- Multiple floors + elevators
  - Modify y-position of the camera
- Increased walking speed
- Flying with specific gesture input (not on button press!)
- Redirected walking: add rotation gains
- Redirected walking: modify geometry on the fly
- Use overlapping spaces
- Suggest your own method!

Knowledge for Assignment 3

- Setting up a networked project
  - In Unity
  - In Unreal
  - Steering a player by tracking input
- Enabling distributed interaction
  - Management of object manipulation rights
  - Correct handling of physics depending on interaction

Networking and Distribution

INTERACTIVE
MEDIA SYSTEMS
Vienna University of Technology

Basic Concepts

- Server/Client structure (both Unity and Unreal)
  - Single server – multiple clients, clients never communicate directly
  - Authoritative vs. Non-authoritative server (theory)
  - Clients control local players
- Applications can be configured as Server, Client or Host(Unity)/ListenServer(Unreal)
  - Host is a server that also has a local client instance
- Network communication
  - State synchronization/replication for objects and variables, remote actions

Client-Server Communication: Authoritative Server

- Server tasks
  - Calculations, physics (collision detection), game logic
  - Receive and process all input data from clients
  - Distribution of position updates to clients
- Client tasks
  - Send input directly to the server
  - Receive and apply updates (including position)
- Characteristics
  - Clients can‘t cheat
  - Server loaded with calculations
  - High network traffic
  - Prediction on the client side in case of delays

Client-Server Communication: Non-Authoritative Server

- Server tasks
  - Synchronization of clients
- Client tasks
  - Input and physics calculated and applied locally
  - Position updates (but not only) are sent to the server
- Characteristics
  - Game logic on the client side
  - No need in prediction
  - Smaller delay
  - Clients own their objects
  - Harder to achieve perfect synchronization

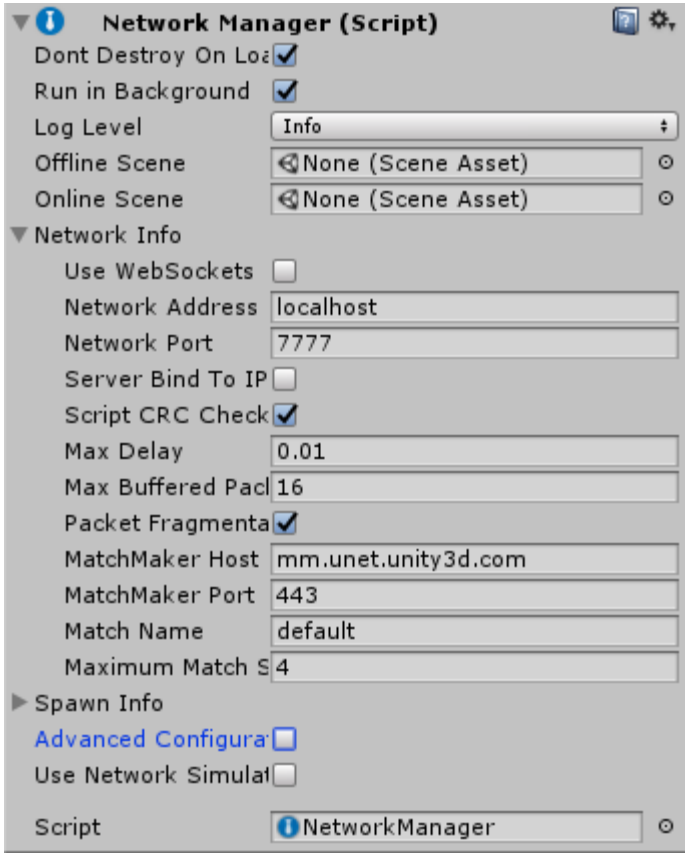
Client-Server Communication: VR, Unity and Unreal

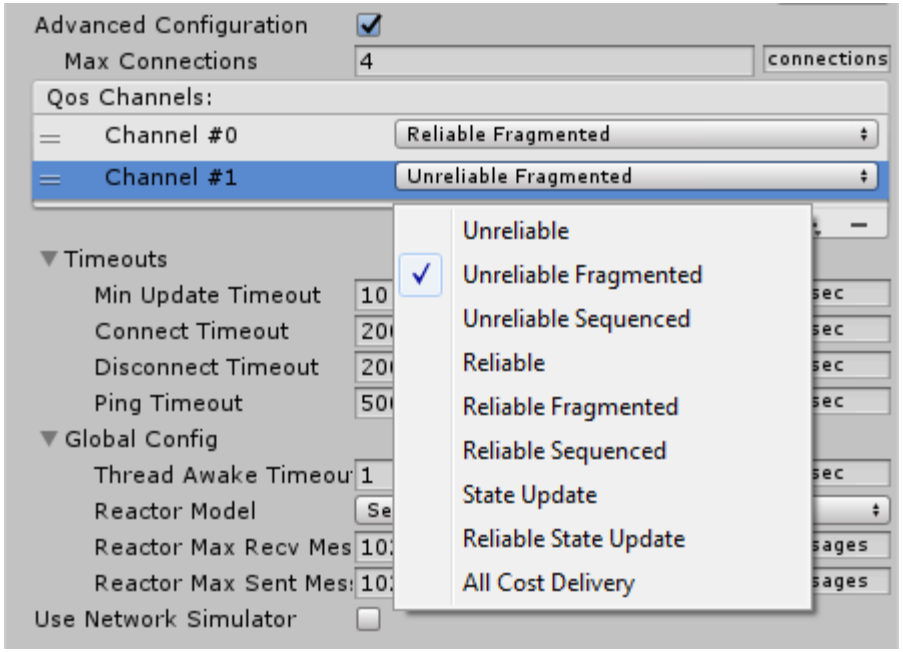
- What is better for VR?
  - Everything that directly influences a user should be processed on the client
    - Tracking input
    - Interaction with nearby objects
- How is it done in UNET (Unity)?
  - Player and player objects controlled by the client (client authority)
  - All the other objects controlled by the server (server authority)
    - But the authority can be passed to clients
    - Necessary if game objects should be manipulated by all clients
  - Server can easily invoke remote actions on clients, remote actions invoked on the server are more restricted
- How is it done in Unreal?
  - Designed to use authoritative servers but workarounds are possible
  - Strict definition of a gameplay
- Game logic can be split between the server and the client

Unity

Network Manager

- Needed to set up a networked scene
- Setting for a player prefab and all spawnable objects
- Network transmission settings
- StartServer(), StartClient(), StartHost()
- public override void OnStartServer(), etc.



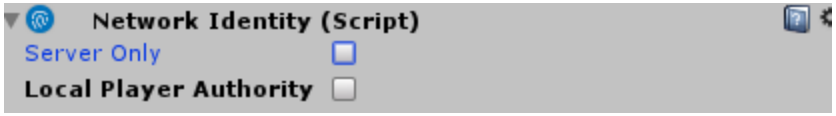


INTERACTIVE MEDIA SYSTEMS
Vienna University of Technology

Connection Management

- NetworkClient
  - NetworkConnection connection
- NetworkServer
  - Spawns distributed objects: Spawn, SpawnObjects, SpawnWithClientAuthority
  - Spawned objects should be in RegisteredSpawnablePrefabs
  - List<NetworkConnection> connections
- NetworkConnection
  - One for client, multiple for server
  - Can „own“ distributed objects
- NetworkClient, NetworkServer, NetworkConnection also implement low-level messaging
  - Out of scope for the exercise but could have been used

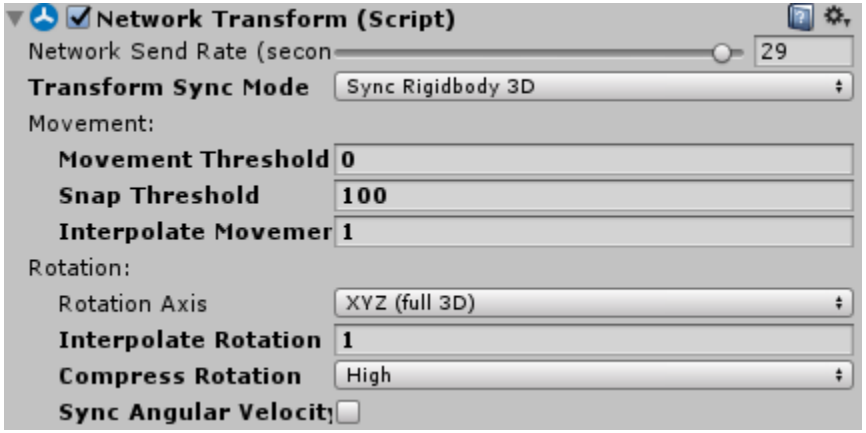
NetworkIdentity



- Identifies objects accross the network
  - NetworkInstanceId is primary property, issued by the server and assigned to clients
- Objects with it should be only created by the server
- For complex objects, must be only on the root of the hierarchy
- ServerOnly to instantiate only on the server
- localPlayerAuthority to enable client control

NetworkTransform

- To synchronize position and rotation of a networked game object
- Server-client for server objects and client-server for an owning client



- Also need a NetworkIdentity component
  - With localPlayerAuthority checked for client-owned objects
- TransformSyncMode
  - Only the position is synchronized with every setting
- NetworkTransformChild
  - To synchronize transforms in the hierarchy
  - Should always be attached to the parent game object

NetworkBehaviour

- Derives from MonoBehaviour
- Allows for UNET functionality: networked actions, callbacks, variable state synchronization from server to client
- Requires a NetworkIdentity on the game object
- A game object can have multiple NetworkBehaviours
- All scripts containing networking functionality should be NetworkBehaviours
- Callbacks OnStartClient and OnStartServer
  - Called when a NetworkBehaviour is activated
  - Custom functionality can be added
- OnStartAuthority, OnStopAuthority
  - Called on behaviours with localPlayerAuthority when authority is passed to a client and removed from it

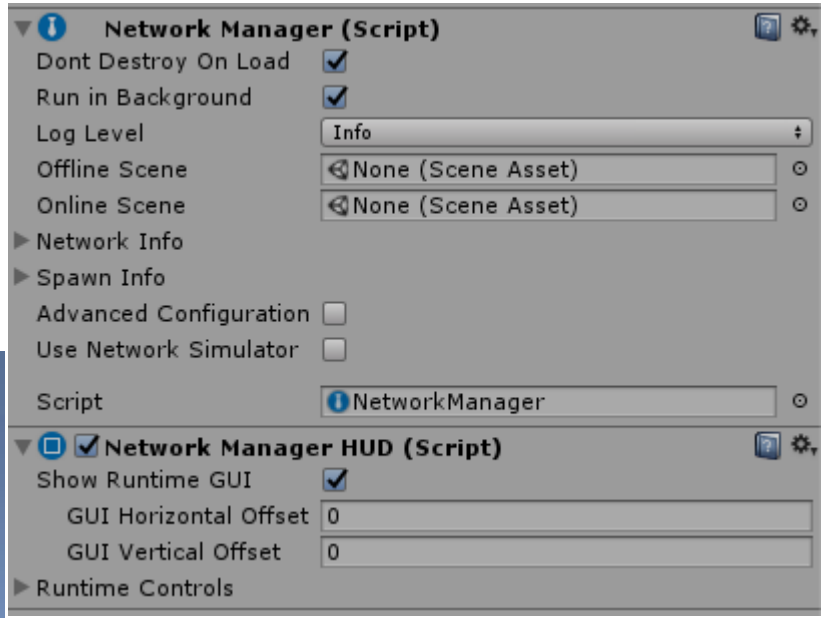
Remote Actions and Attributes

- From server to client
  - [SyncVar]
    - To synchronize variables of simple types
  - [ClientRpc]
    - Called on the server, invoked on all clients
    - Function must begin with the prefix Rpc
  - [TargetRpc]
    - Called on the server, invoked on a particular client
    - Function must start with the prefix Target
- From client to server
  - [Command]
    - called on a client, invoked on the server on the player object associated with the connection
    - Function must start with the prefix Cmd

- Arguments of remote actions are serialized over the network
- Other attributes
  - [Client]
    - For a function to only run on a client
  - [Server]
    - To only run on the server
  - [NetworkSettings]
    - To configure the transmission mode of a

Network Manager HUD

- Addition to Network Manager
- Enables Runtime GUI
- Allows to setup as server or client with adress and port
- Way easier to use in an .exe



Unreal

UE4 Networking

- UObject
  - Object derived from UObject
  - Objects, Game Objects, Actors, Pawns, Characters
- Server
  - The computer process with an instance of a UWorld that contains Actors that are replicated to clients
  - The state of the world contained here is considered to be the "real" or "correct" one
- Client
  - The computer process with an instance of a UWorld that contains Actors that were received from a Server over a network connection
  - The state of a client is an approximation of the state of the Server. It is not considered to be accurate or correct.

UE4 Networking: Connection

- No need in placing anything special in the map, can just start the game as a server or a client
  - Open command line at the location of the project
  - “UE4Editor.exe” “ProjectName.uproject” MapName?listen -game -log
  - “UE4Editor.exe” “ProjectName.uproject” MapName -server -log (-nosteam)
  - “UE4Editor.exe” “ProjectName.uproject” IPaddress -game -log
- Cannot connect one editor instance to another!
  - It is possible to start a listen server in the editor and connect a client started from command line
  - Possible to run a dedicated server and a client on the same machine (ip 127.0.0.1)
- UE4Editor provides in-built multiplayer testing
  - Gets difficult with actual input devices (Vive, Leap)
  - Make sure to start testing on two machines as soon as possible!

UE4 Gameplay Classes

- Important for the project!
- **GameMode** (or GameModeBase)
  - Only exists on the server
  - Rules for connecting to the game
  - Good place for other game rules
- **GameState**
  - For clients to monitor the state of the game
- **GameInstance**
  - On the server and replicated to clients (one on each)
  - Good for keeping information relevant to each player
- **PlayerController**
  - On the server for each player and on the Role_AutonomousProxy once
  - Meant to process input
  - Steers the player pawn
- **PlayerPawn** (or Character)
  - Exists everywhere for each player
  - Gets spawned as the main player
- **PlayerState**
  - For each player on the server and replicated to all clients
  - Good for monitoring players

UE4 Networking:

- Network Roles
  - rules that control how information flows between Server and Client
- Spawning Rule
  - Only the server can spawn Actors that will replicate to clients
- UPROPERTY Replication Rule
  - A UPROPERTY that is tagged with Replicated is owned/controlled by the server
  - If a client changes the value of a replicated variable locally, that value will be overwritten the next time the server changes it
- Rules For Calling Functions
  - call functions on objects on the server version or client version of objects

Network Roles

- Rules that control how information flows between Server and Client
- **ROLE_Authority**
  - authoritative version of the object and its state represents what is considered to be the only "real" state of the object
  - An object with ROLE_Authority can execute function calls on any object on the server
  - an object with ROLE_Authority will also replicate any changes to UPROPERTY fields to any client versions of the object
- **ROLE_None**
  - object has no networking role and isn't replicated
- **ROLE_SimulatedProxy**
  - Simulates the state of the server object on a client
  - Client has no authority to change the state of the object
- **ROLE_AutonomousProxy**
  - Client can execute function calls on this object that execute on the server
  - Role_AutonomousProxy is the owning client for pawns
- **IsLocallyControlled**
  - Not a network role but useful to check if the game instance is a local player (client or a listen server)

Actor Replication

- Always from server to clients only**!**
- **bReplicates**
  - Must be true
- **bAlwaysRelevant**
  - Actor is always relevant to all clients and will always be replicated
- **bOnlyRelevantToOwner**
  - Will only replicate to the player represented by the owning Pawn or PlayerController
- **bNetLoadOnClient**
  - If true the Actor will load from a level file on a network client. This should be set to true for Actors you place in a map that you want to exist on a client (typically most Actors want this)
- **bTearOff**
  - If the Server sets this to true all clients will take authoritative control of their locally replicated versions of the actor and changes and function calls on the actor will no longer be replicated over the network. It will be as though it was a locally spawned actor.
- **bReplicateMovement**
  - Set to true if you want to be able to move the Actor and have its position be updated on clients automatically
  - Pawns have this one default, **however it is not sufficient to see moving Vive controllers and Leap hands**
- Actor must be "network-relevant"
  - When it is within the viewing range of a client
  - AActor::NetCullDistanceSquared controls the culling distance

Property Replication

- Each Actor can maintain a list of replicated UPROPERTY()
- From server to clients only
- UPROPERTY(replicated)
  - Can be conditionally replicated: SkipOwner, SimulatedOnly etc.
  - RepNotify: a user-defined function will be called on the client every time the property changes
- Properties always replicate reliably

Component Replication

- Still very ambiguous in Unreal!
- Components are not designed to replicate
  - However component replication is useful and needs to be used in the assignment
- Components replicate as a part of their owning Actor
  - Static components (the ones created in the Actor constructor script as Default Subobjects) will always spawn on clients
  - Dynamic components need to be marked as replicated to be spawned on clients
  - Clients can also spawn locally-owned non-replicated components, depending on the required functionality

Function Call Replication

- Any UFUNCTION() can be set to replicate and execute on client or server instances of the object
- Possible to choose reliable or unreliable replication
- Types of remote function calls
  - Server
    - function will execute on the server version of the object ONLY
    - object must have ROLE_Authority or ROLE_AutonomousProxy to execute this method
    - Functions marked as Server must also be marked as WithValidation and implement a validate function
  - Client
    - function will execute on the client that has a version of the object with ROLE_AutonomousProxy (client that "Owns" this object)
    - The only ROLE_AutonomousProxy objects that exist in Unreal are client instances of PlayerControllers
  - NetMulticast
    - function will execute on all clients that have an instance of the object. Only an object with ROLE_Authority can execute this function, any other networking role won't do anything

Replication Example

EXAMPLE for an Actor owned by Client 1

- *Actor_Server* - ROLE_Authority
- *Actor_Client1* - ROLE_AutonomousProxy
  - any calls to UFUNCTION() methods marked as Server will execute on Actor_Server
- *Actor_Client2* - ROLE_SimulatedProxy
  - any calls to replicated functions will be ignored

- changes to any **UPROPERTY** on the *Actor_Server* instance will get **replicated** to both *Actor_Client1* and *Actor_Client2* **automatically**

- any **UFUNCTION()** methods that are replicated as **Client** or **NetMulticast** that are called on *Actor_Server* by the server will execute on *Client1* in the case of Client or both *Client1* and *Client2* in the case of NetMulticast

Task 3:
Distributed Interaction

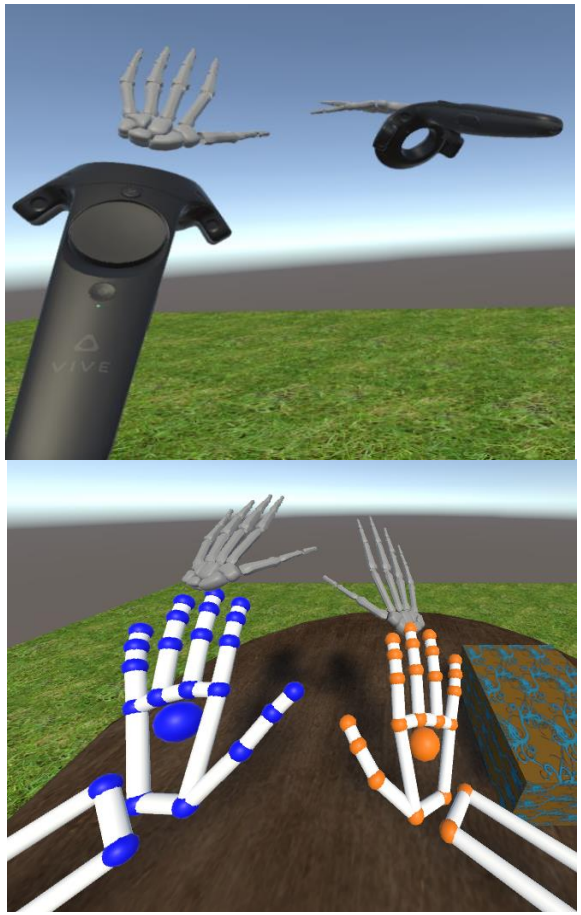
INTERACTIVE
MEDIA SYSTEMS
Vienna University of Technology

Required Functionality

- A Leap-controlled player and a Vive-controlled player can connect to the same game and see each other (hands only)
  - Pre-implemented in Unity
  - Synchronisation of hands positions needs to be done in Unreal
- Both players can manipulate a server-created object
  - The object is visible to both players: needs to be replicated (replication checked in Unreal, NetworkIdentity in Unity)
  - Both players can pick up an object with a special combination of actions and move it around, correctly seen by the other player
  - One player can pass the object to the other player without dropping it
  - The object follows Physics rules when not being manipulated by a player: make sure Physics properties are coordinated on the server and the clients!

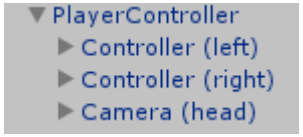
Unity Resources

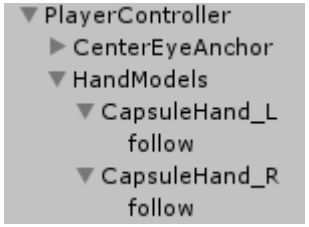
- Network set up
  - Modify Network Address
- Vive Player
  - Controlled by HTC Vive input
- Leap Player
  - Controlled by Leap Motion input
  - Fixed camera
- Only skeletal hands are visible to the other player
- Need two computers to test
  - LAN connection is better
  - A dedicated server for the task implementation
  - Can run the server and the Leap client on the same machine



Player Control

- `LocalPlayerController.cs`
  - Runs locally on PlayerController object
  - Steers the local copy of networked hands





- Player prefab:
  - Spawned as the main player on every machine
  - Transform is empty, the actual character depends on what is set in the Actor component

- `Actor.cs`
  - Is a NetworkBehaviour, attached to the Player prefab
  - Runs on every machine
  - Initializes a correct Character
  - Commands can be invoked on the server in this script!
  - Will be used in distributed interaction

Synchronize Server Object

- NetworkIdentity
  - localPlayerAuthority
- NetworkTransform
  - Find parameters providing the smoothest sync
- Can only be manipulated by the server!
  - Good to test with a host and a client: move on the host and observe on the client
  - Not active if a client is started but not connected to the server

Allow Clients to Manipulate Server Objects

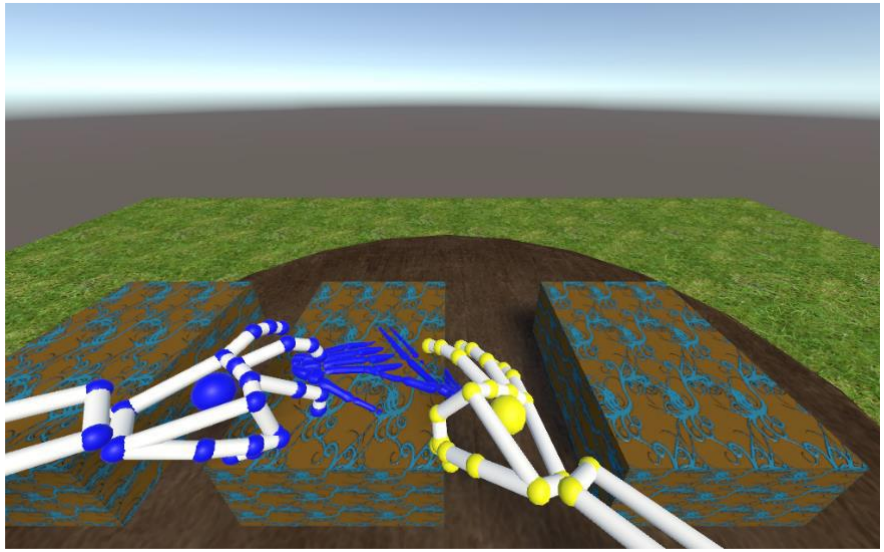
- NetworkIdentity.AssignClientAuthority(NetworkConnection conn)
  - To allow a client to manipulate server objects
  - Objects become „belonging“ to the client
  - Will disappear if to disconnect the owning client
- NetworkIdentity.RemoveClientAuthority(NetworkConnection conn)
  - To revert the authority back to the server
  - A distributed object should always have an owner (authority cannot be removed from the server, only passed to the client)
- NetworkIdentity.hasAuthority
  - To check who currently has it
- Need some conditions to invoke these functions
  - For example, when a user touches a distributed object
  - Can test authority exchange with key presses

Conditions for Client Object Control

- What are the conditions
  - User „grabs“ an object with both hands
- Where to check them
  - Server?
    - Avatar (hands) is synchronized unreliably -> possible package loss can delay interaction
  - Client?
    - Need to communicate the event to the server -> reliably by remote actions
    - Can also get slow, but reliable messages are sure to be delivered

How to Grab an Object

- With HTC Vive Controllers
  - Both triggers down – local condition
  - Both netwoked hands touching the object – localPlayer condition
- With Leap Motion – controlled hands
  - Both Leap hands in Pinch – local condition
  - Both networked hands touching the object – localPlayer condition



Notify the Server

- [Command] attribute
- Can only be invoked on the player object of a corresponding NetworkConnection
- Server grants the authority over an object in response
- What if two players grab an object simultaneously?
  - The first one gets it
  - Think about a situation where both users are „holding“ an object and then the current „owner“ releases the grab
    - Make sure the object does not fall

Unreal Resources

- GameMode_BP.uasset
  - Put it in your project and set as the used GameMode in ProjectSettings->Modes&Maps
  - Pawns to be spawned as the 1st and 2nd connecting player can be chosen inside the blueprint code
- Leap Player
  - Use your pawn from the first assignment
  - Only grabbing functionality is necessary, can remove gesture-related functionality if it collides with the replication
  - Movement of the Leap-controlled hands needs to be made visible to the Vive-controlled player (only position and rotation, replicating individual fingers is not required)
- Vive Player
  - Use the MotionControllerPawn from the VRTemplate
  - Movement of the Vive-controlled hands needs to be made visible to the Live-controlled player (only position and rotation, replicating the hand animation is not required)
- C++ classes with implemented functionality available as a reference
  - Vive-controlled pawn in C++
  - Custom PlayerController
  - Custom Actor class for the shared object

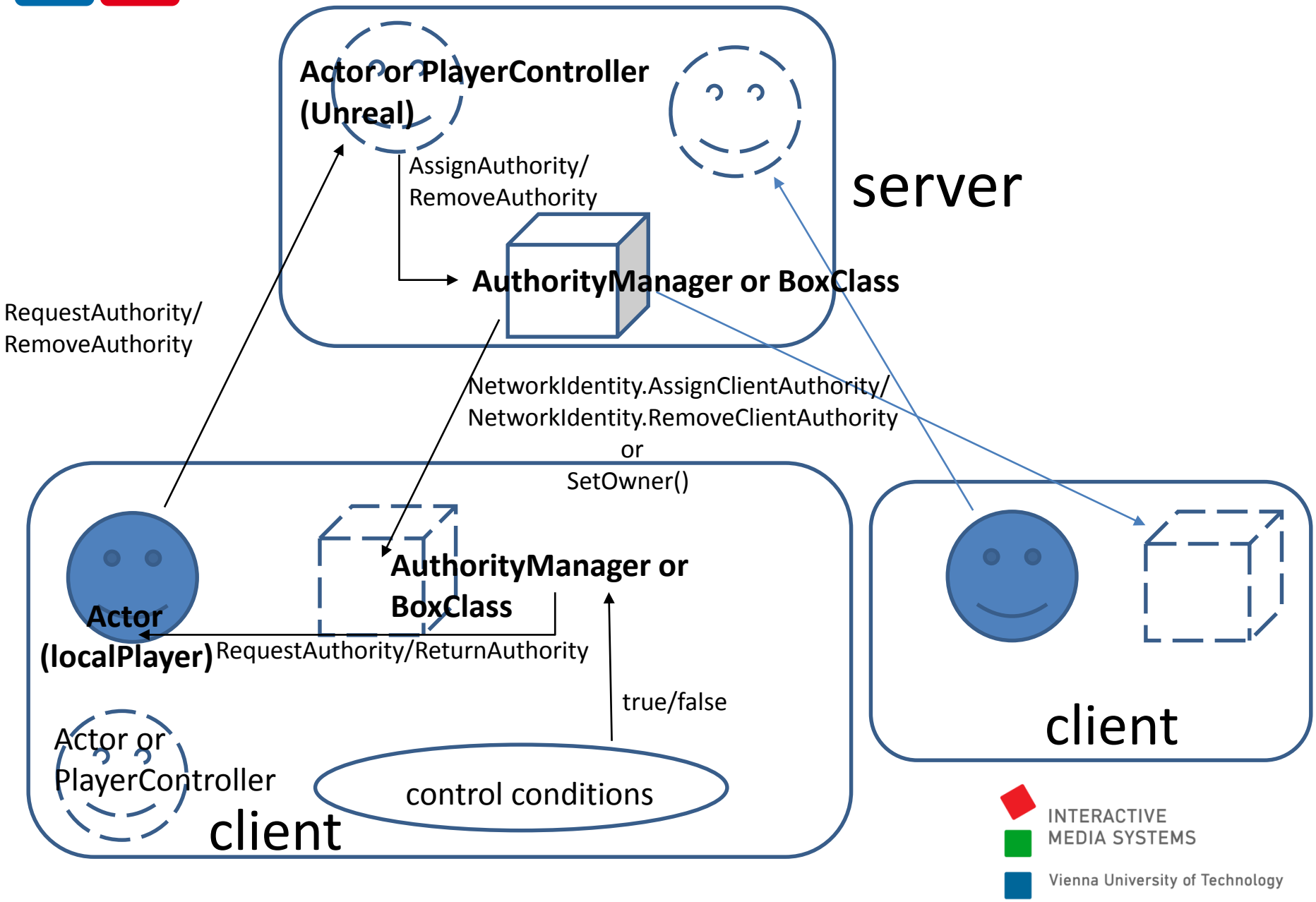
How to Replicate Hands

- Replicated UPROPERTY for each hand‘s position and orientation as a variable in the Pawn class, with SkipOwner condition
- Each tick, check the pawn‘s network role
  - Set each hand‘s mesh position and rotation to the replicated values if Role_SimulatedProxy
  - Send position and rotation obtained from the tracking data to the server if IsLocallyControlled, set replicated position and rotation to those values in the server function
- Check the C++ implementation in the reference Pawn class

How to Replicate the Box

- Box needs to be able to switch owner when grabbing conditions are fulfilled (similar to authority management in Unity)
- When one of the PlayerControllers on the server is the current owner (the box is being held by a player) the replication mechanism is the same as for the hands
- Use C++ example
- Grabbing conditions to use: both hands are grabbing the object (both triggers down for Vive and both hands in grabbing gesture for Leap)
- If both Leap hands grabbing is difficult to achieve use an arbitrary gesture (but the same on both hands)

Overall Communication Flow



INTERACTIVE
MEDIA SYSTEMS
Vienna University of Technology

Structure

- Plan scripts/blueprints organization
- Try to keep logical functionality separately from the particular implementation
  - Authority exchange
  - Conditions for the authority
    - Can be different for different shared objects and input devices!
  - Implementation of these conditions
- Allow multiple shared objects!
- Shared object manipulation should also work if to use two Vive-controlled or two Leap-controlled players

Online Help

- Unity Networking Guide
  - https://docs.unity3d.com/Manual/UNet.html
  - https://unity3d.com/de/learn/tutorials/topics/multiplayer-networking
- Unreal
  - https://docs.unrealengine.com/latest/INT/Gameplay/Networking/
  - https://wiki.unrealengine.com/Replication
  - https://wiki.unrealengine.com/Network_Replication,_Using_ReplicatedUsing_/_RepNotify_vars
  - https://docs.unrealengine.com/latest/INT/Gameplay/Networking/Actors/Properties/index.html
  - https://docs.unrealengine.com/latest/INT/Gameplay/Networking/Actors/Components/
  - https://www.unrealengine.com/en-US/blog/network-tips-and-tricks
  - https://docs.unrealengine.com/latest/INT/Gameplay/Networking/Blueprints/index.html
  - https://docs.unrealengine.com/latest/INT/Gameplay/HowTo/Networking/ReplicateFunction/Blueprints/index.html
  - https://docs.unrealengine.com/latest/INT/Gameplay/HowTo/Networking/ReplicateVariable/Blueprints/index.html