Game Engines in AI Exercise 03

Waldemar Zeitler

November 21, 2018

Note

This task was tested with the oculus rift. Fabian Kues helped me with the setup of the vr motion controllers in code, since he was working on the task at the same time as me and I had some problems there.
Thanks for allowing me to finish the task one day later!

Task 1

For the vr controlls a pawn was created. The created pawn handles the setup of the controllers, overlap events and the buttons. The following code shows the constructor of the pawn. First the root component and camera are setup, afterwards the motion controllers and the meshes for the controllers. It is important to give the controllers the correct motion source, so that both are usable correctly. The meshes of the hands are set to be able to go through objects and trigger overlap events. The collision can also be changed to not allow passing through objects but this is done here to allow an easier pickup of objects. To actualy set the meshes of the controller it is also necessary to find the appropriate controller meshes. To find them in the editor the view option of "Show Engine Content" needs to be activated. Furthermore the hand meshes receive overlap events, to get the right object for a pickup. The last thing to do was to auto posses the player as player0.

```
AVRPawn::AVRPawn()
{
	// Set this pawn to call Tick() every frame. You can turn
	    ↪ this off to improve performance if you don't need
	    ↪ it.
	PrimaryActorTick.bCanEverTick = true;

    // Create the root compnent and the camaera
    RootComponent = CreateDefaultSubobject<USceneComponent>(TEXT("
        ↪ RootComponent"));
    UCameraComponent* Camera = CreateDefaultSubobject<
        ↪ UCameraComponent>(TEXT("Camera"));
    Camera->AttachTo(RootComponent);

    // Setup for the motion controller hands. Important is the
        ↪ MotionSource, do differentiate the controllers
```

```
LeftHand = CreateDefaultSubobject<UMotionControllerComponent>(
    ↪ TEXT("LeftHand"));
LeftHand->SetupAttachment(RootComponent);
    LeftHand->MotionSource = FName("Left");
RightHand = CreateDefaultSubobject<UMotionControllerComponent
    ↪ >(TEXT("RightHand"));
RightHand->SetupAttachment(RootComponent);
    RightHand->MotionSource = FName("Right");

// Get the static controller mesh and set the overlap and
    ↪ physics parameters.
// Important to check for overlap events and allow the hands
    ↪ to go through objects
LeftHandMesh = CreateDefaultSubobject<UStaticMeshComponent>(
    ↪ TEXT("LeftHandStaticMesh"));
LeftHandMesh->SetupAttachment(LeftHand);
    LeftHandMesh->SetCollisionProfileName("OverlapAllDynamic")
        ↪ ;
    LeftHandMesh->BodyInstance.SetCollisionEnabled(
        ↪ ECollisionEnabled::QueryAndPhysics);
    LeftHandMesh->SetEnableGravity(false);
RightHandMesh = CreateDefaultSubobject<UStaticMeshComponent>(
    ↪ TEXT("RightHandStaticMesh"));
RightHandMesh->SetupAttachment(RightHand);
    RightHandMesh->SetCollisionProfileName("OverlapAllDynamic
        ↪ ");
    RightHandMesh->BodyInstance.SetCollisionEnabled(
        ↪ ECollisionEnabled::QueryAndPhysics);
    RightHandMesh->SetEnableGravity(false);

// Find the right static mesh of the controller and give it to
    ↪  the hand meshes
static ConstructorHelpers::FObjectFinder<UStaticMesh> Mesh(
    ↪ TEXT("StaticMesh'/Engine/VREditor/Devices/Vive/
    ↪ VivePreControllerMesh.VivePreControllerMesh'"));
if (Mesh.Succeeded())
{
    LeftHandMesh->SetStaticMesh(Mesh.Object);
    RightHandMesh->SetStaticMesh(Mesh.Object);
}


    // Overlap events to check when an object can be picked up
    RightHandMesh->OnComponentBeginOverlap.AddDynamic(this, &
        ↪ AVRPawn::BeginOverlapRight);
    RightHandMesh->OnComponentEndOverlap.AddDynamic(this, &
        ↪ AVRPawn::OnOverlapEndRight);
    LeftHandMesh->OnComponentBeginOverlap.AddDynamic(this, &
        ↪ AVRPawn::BeginOverlapLeft);
    LeftHandMesh->OnComponentEndOverlap.AddDynamic(this, &
```

```
            ↪ AVRPawn::OnOverlapEndLeft);

    // Player setup
    AutoPossessPlayer = EAutoReceiveInput::Player0;
}
```

The pawn is able to pick objects up and throw them, this is done by binding the trigger button of the oculus controller. If the controller triggers the overlap event and the trigger button is pressed and hold, the object will be attached to the controller. While the button is hold the object will follow the controller. If the button is released the object will drop down. Like this it is possible to throw objects.
The following code shows the "HoldRight" and "ThrowRight" functions, which are triggerd through the press and release of the trigger button (they are pretty much the same for the left hand). It is first checked if an overlap object is availible and that not another object is hold. Afterwards the objects gets cast to an "AStaticMeshActor" to remove the physics and snap it to the motion controller. For the release it is just checked if an object is hold and if so it gets detached and the physics rest.

```
void AVRPawn::HoldRight()
{
    // Attach the overlaping actor to the hand and make him
        ↪ moveable
        if (OverlapActorRight != NULL && RightHandObject == NULL)
        {
        // Cast the OverlapActor to a static mesh actor
                RightHandObject = Cast<AStaticMeshActor>(
                    ↪ OverlapActorRight);

                // Physics and gravite need to be romved for the
                    ↪ attachment
                RightHandObject->GetStaticMeshComponent()->
                    ↪ SetEnableGravity(false);
                RightHandObject->GetStaticMeshComponent()->
                    ↪ SetSimulatePhysics(false);
                RightHandObject->AttachToComponent(RightHandMesh,
                    ↪ FAttachmentTransformRules::
                    ↪ SnapToTargetNotIncludingScale);
        }
}

void AVRPawn::ThrowRight()
{
    // Throw the object out of the hand, when the button is
        ↪ released
        if (RightHandObject != NULL)
        {
                RightHandObject->DetachFromActor(
                    ↪ FDetachmentTransformRules::
```

```
                ↪ KeepWorldTransform);

            // Object is detached from the hand and physics can
                ↪  be reset to normal
            RightHandObject->GetStaticMeshComponent()->
                ↪ SetEnableGravity(true);
            RightHandObject->GetStaticMeshComponent()->
                ↪ SetSimulatePhysics(true);
            RightHandObject = nullptr;
        }
}
```

The overlap functions and binding of the buttons is done like usual and can be viewed in the code.

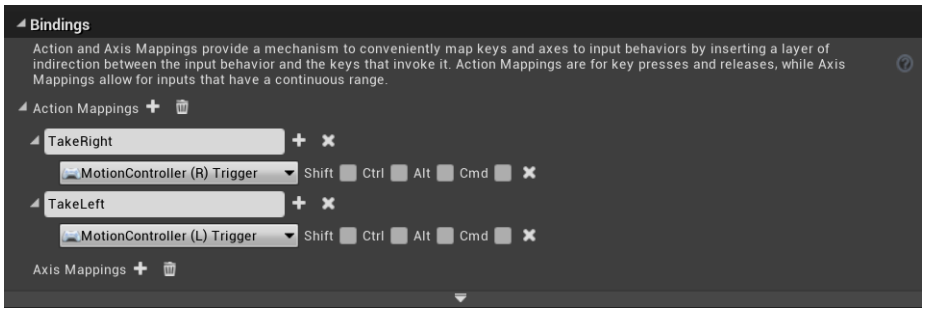

Figure 1: Input Binding

The pawn is spawned by the game mode, with the following code.

```
AEX3_Waldemar_ZeitlerGameMode::AEX3_Waldemar_ZeitlerGameMode()
{
    // The VRPawn shall be the default pawn and spawn with the
        ↪ player start
    DefaultPawnClass = AVRPawn::StaticClass();
}
```

Like this it is only necessary to place a player start into the world and to set the game mode in the world editor.

Note

The Tick function can probably be removed but I can't test the program without a vr system, so I left the function, before anything breaks.

Task 2

The following picture shows the motion controller with the collosion box set for the controller and the auto convex collision set. The collision box is for overlap

events, if the physics of the controller are set to not allow passing through objects, so that an overlap is still possible.

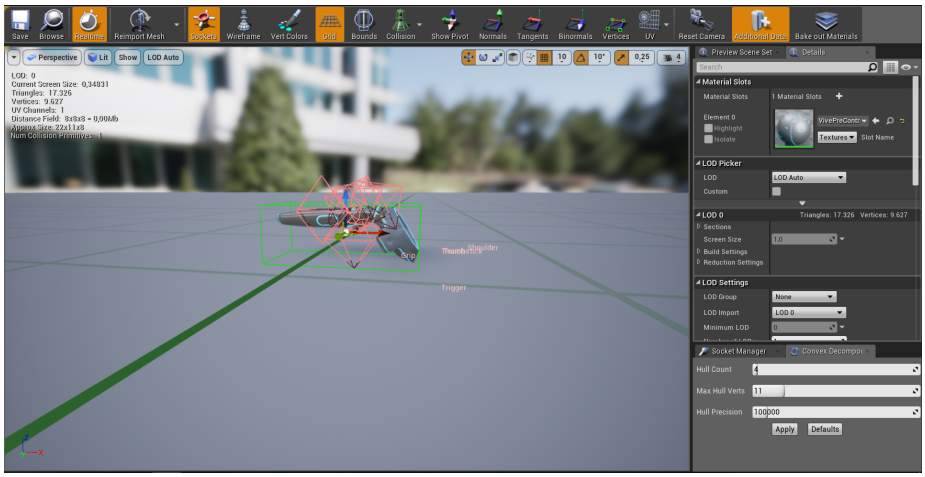

Figure 2: Convex Collision

The current map contains the player start, a cube and some "pins". When the game begins it is possible to pick the cube up and throw him into the pins.

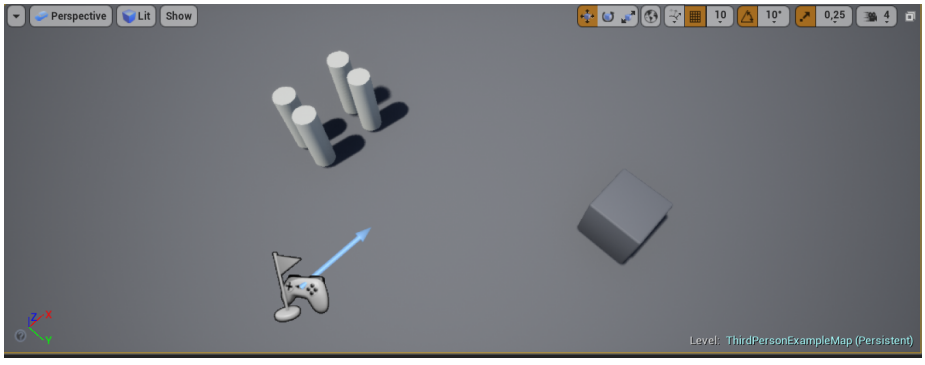

Figure 3: World

As already mentioned, the motion controller can go through the cube, for an easier interaction. While holding the button the cube follows the hand motion and if the hand has some velocity while releasing the cube, the cube will be thrown. The goal of this little setup is to take the cube and throw him against the pins.

Note

The pins and the cube also received a convex hull.
The program has to be started as VR-Preview.