DEVELOPING VIRTUAL REALITY WITH **UE4**

UNREAL® ENGINE VR Cookbook

Mitch **McCAFFREY**

Praise for *Unreal® Engine VR Cookbook*

"I've been a fan of Mitch's work for quite some time. Back in early 2014, Mitch was exploring new locomotion mechanics in Unreal Engine for VR characters and sharing his findings, as well as numerous sample files, on the www.unrealengine.com forums in massive ongoing threads. Even in the early days of VR exploration, his work was helpful to many newcomers working to understand the issues of designing for a comfortable virtual experience. With his YouTube channel, *Mitch's VR Labs*, he helped thousands of people understand the foundations of locomotion and interaction mechanics with clear and concise UE4 videos. I'm thrilled that he has taken the time to bring all his knowledge and experience in working with Unreal Engine and virtual reality to the *Unreal® Engine VR Cookbook*. With the current attention and appetite for understanding how best to work in the exciting medium of virtual reality with Unreal Engine, I think Mitch is uniquely qualified to share this book with the world."

—**Luis Cataldi**, Unreal Engine Education, Epic Games, Inc.

This page intentionally left blank

Unreal® Engine VR Cookbook

Unreal® Engine VR Cookbook

Developing Virtual Reality with UE4

Mitch McCaffrey

EPIC GAMES

Addison-Wesley

Boston • Columbus • Indianapolis • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Library of Congress Control Number: 2016958916

ISBN-13: 978-0-13-464917-7
ISBN-10: 0-13-464917-6

1 17

Editor-in-Chief
Mark L. Taub

Executive Editor
Laura Lewin

Development Editor
Sheri Replin

Managing Editor
Sandra Schroeder

Full-Service Production Manager
Julie B. Nahil

Project Editor
Anna Popick

Copy Editor
Barbara Wood

Indexer
Dick Evans

Proofreader
Anna Popick

Technical Reviewers
Luis Cataldi
Jeff Wilson
Clinton Crumpler

Editorial Assistant
Olivia Basegio

Cover Designer
Chuti Prasertsith

Compositor
The CIP Group

Contents

Preface xiii

Acknowledgments xvii

About the Author xix

Part I: Getting Started **1**

1 Terminology and Best Practices 3

Terminology 4

Devices 4

Software 6

Unreal Engine 9

Best Practices 10

Summary 11

2 Head Mounted Display Setup 13

Gear VR 14

Gear VR Project Setup 14

Gear VR Global Menu Setup 20

Gear VR Global Menu Progress Material 26

Rift and Vive 32

Rift and Vive Project Setup 33

Rift and Vive Tracking Origins 39

Summary 42

3 Toolkit 43

Generic Function Library 44

Oculus Function Library 45

Steam VR Function Library 47

Summary 47

Part II: Recipes 49

4 Trace Interaction 51

Understanding Trace Interaction 52

Understanding Interfaces 55

Setting Up Trace Interaction 57

Basic Project Setup 57

Interaction Interface Setup 58

Interaction Component 61

Interaction Pawn Setup 73

Setting Up a Basic Interactive Object 75

Summary 79

Exercises 80

5 Teleportation 85

Setting Up Teleportation 86

Parabolic Tracing 86

Visualizing the Teleport 92

Visualization Material 92

Visualization Actor 94

Simple Teleportation Volume 95

Summary 101

Exercises 101

6 Unreal Motion Graphics and 2D User Interfaces 103

Challenges with 2D UI in VR 104

History and Compatibility of UMG 105

Basic VR Menu 105

Menu Actor 108

Menu Pawn 111

Custom Menu Interaction 113

Implementing Custom Menu Interaction: Approach 1 113

Implementing Custom Menu Interaction: Approach 2 118

Summary 123
Exercises 123

7 Character Inverse Kinematics 125
Introduction to Inverse Kinematics 126
Setting Up Head IK 127
Mirror Creation 128
IK Pawn 130
Head IK Animation Blueprint 132
Setting Up Hand IK 137
Adding Motion Controllers to Your Pawn 137
Hand IK Animation Blueprint 138
Summary 144
Exercises 145

8 Motion Controller Interaction 147
Why Motion Controller Interaction Works 148
What to Look Out For: The Importance of Affordance 148
Shared Input of the Current Generation of Motion Controllers 149
Setting Up the World Interaction Project 149
Interacting with Objects 151
Creating the World Interaction Interface 151
Creating the Interactor Component 153
Adding Interaction to the Interaction Pawn 161
Creating the Interactive Objects 163
Creating an Interactive Static Mesh Actor 163
Creating an Interactive Button 166
Creating an Interactive Lever 176
Summary 187
Exercises 188

9 VR Locomotion 189

Simulator Sickness 190

Locomotion Types 190

Natural 191

Teleportation 191

Vehicle 192

Physical 192

Artificial 193

Locomotion Implementation 194

First Person Template for Snap Turning 194

First Person Template for Running in Place 201

Summary 205

Exercises 205

10 VR Optimization 207

Requirements of VR Rendering 208

Latency Mitigation 209

Performance Improvements 215

Forward versus Deferred 215

Instanced Stereo 220

Hidden Area Mesh Optimization 221

VR Project Settings 221

Summary 229

Exercises 229

Part III: Appendices 231

A VR Editor 233

Enabling the VR Editor 234

Controlling the VR Editor 235

Navigating the World 237

Interacting with Objects 238

Interacting with Menus 239

Summary 241

B Resources 243
Epic 244
Oculus 244
Valve 244
Google 245
Community 245
Physical Meetups 245
Conferences 246

Index 247

Preface

The resurgence of virtual reality (VR) is upon us, and so an explosive demand for new and compelling VR experiences is coming. A growing billion-dollar medium, VR brings new and exciting challenges to the world of game development, while bringing previously uninitiated industries into the exciting realm of real-time rendering. Whether it's creating a classic-style arcade game or visualizing an exquisite villa in the grass-green hills of Tuscany, VR brings an unparalleled level of immersion to any experience. This immersion, however, brings with it a slew of challenges. When you develop for VR, you write the rulebook as you go.

In this book is a set of best practices and common interaction paradigms that have emerged over the past few years. Whether from the big players in the space or from a one-person team, the entire VR community is contributing to the compendium of VR knowledge at an astonishing pace. This book not only shows you how to implement these paradigms in Unreal Engine, it also shows you how to choose which one will suit your project.

Presented in cookbook style, this book takes a practical approach to learning the nuances of VR development. Each recipe shows you how to build a common system that is used in many VR games/experiences today. Whether you are building a first-person shooter or a relaxation simulator, each example keeps the content abstract enough that it applies to any genre, yet also mentions specific approaches that may work well for certain game types.

Who Should Read This Book?

This book is designed for people already familiar with navigating Unreal Engine 4 (UE4) and Blueprints. If you have little experience with either, visit the Unreal Engine documentation before reading this book. However, I explain most things when it comes to actual coding, and most of the mathematics is covered in sidebars and the main content. Therefore, a deep level of coding is not required.

How This Book Is Organized

This book is split into three parts:

- **Part I, "Getting Started"**: Chapters 1 through 3 contain an introduction to some of the terminology used within this book and the VR industry. This part also contains instructions for creating basic projects for various VR headsets.

- **Part II, "Recipes"**: Chapters 4 through 10 contain the main recipes for the book. This part covers everything from motion controller interaction to VR movement schemes.
- **Part III, "Appendices"**: This auxiliary information on the VR Editor and resources will help you on your journey in VR development.

Conventions Used in This Book

The following typographical conventions are used in this book:

- `Monospaced` text indicates blocks of code.
- *Italicized* text indicates a new key word or phrase.

note

A Note signifies a tip, suggestion, or general note.

SIDEBAR

A sidebar contains auxiliary information for the main text, such as an explanation of the mathematical principles used or work related to the main content.

warning

A Warning indicates a warning or caution.

Why Blueprints?

When programming in UE4, there are two main ways to implement logic in your games/experiences: the visual scripting language, Blueprints, and the more traditional coding language, C++.

Compared to Blueprints, C++ can be a little obscure because the required syntax can take a while to learn; however, C++ gives you greater access to some of the more hidden features of the engine. This, however, won't be a problem in this book; most of what is taught will be at a high enough level that Blueprints will suit your needs.

Blueprints also offers an easier method for migrating your work from one project to another, which will allow you to take any work done in this book and easily apply it to your existing content.

Companion Website

https://github.com/mitchemmc/UE4VRCookbook holds the various source files for every chapter that needs them. This website allows you to check your work for each recipe.

Acknowledgments

I am grateful to Laura Lewin, executive editor at Addison-Wesley, and Luis Cataldi, education evangelist at Epic Games, who gave me the opportunity to publish this book with Addison-Wesley. I thank the team from Addison-Wesley. Sheri Replin, the development editor, reviewed all the chapters and gave me valuable suggestions on the content presentation. Olivia Basegio helped me coordinate with the team at Addison-Wesley and organize the necessary paperwork.

This book could not have been published without feedback from technical reviewers Clinton Crumpler and Marty Murphy. They identified errors and provided valuable suggestions. Thank you.

Finally, I thank my significant other, Nicola, for being understanding and supportive while late nights were spent working on this book.

About the Author

Mitch McCaffrey is an independent game developer and creator of many community resources for Unreal Engine VR developers. He teaches game development best practices with his popular YouTube channel *Mitch's VR Lab* and demonstrates these in his community-driven VR Content Examples. His website is http://mitchvr.com.

This page intentionally left blank

PART II

RECIPES

This page intentionally left blank

CHAPTER 7

CHARACTER INVERSE KINEMATICS

VR allows players to inhabit a virtual character's body, and thus replicating players' real-world movements to their virtual characters can be an immersion multiplier if done right.

This chapter discusses techniques built into Unreal Engine for interpolating the user's current pose from known information about the player's location.

Introduction to Inverse Kinematics

As opposed to forward kinematics, where you define each bone's rotation to get your desired output, inverse kinematics (IK) allows you to define an end effector goal and let the system interpolate what the bone rotations need to be to get to that goal.

Many VR headsets give you access to precise head tracking so you know exactly where the user's head is in your virtual world. Without expensive motion capture equipment, however, you do not necessarily know where any other bone is (aside from hands in the case of motion controllers).

In these situations, inverse kinematics lets you extrapolate where the other bones in the skeleton are located because of the predictable way in which bones rotate. The simplest method of IK is two-bone IK (see Figure 7.1), because the bone rotation can be calculated analytically with some basic trigonometric identities; however, note that once a third dimension is introduced, as is the case in VR, there are an infinite number of possible solutions to a two-bone IK setup. To deal with this, Pole Vectors/Joint Targets are introduced. These allow developers to create a preference for how the bones move. In UE4, Joint Targets define a point in space that, along with the direction from the root bone to the IK target point, generates a plane (the normal of this plane is the cross product of these two directions). This plane is then used to simplify the IK problem back down to two dimensions.

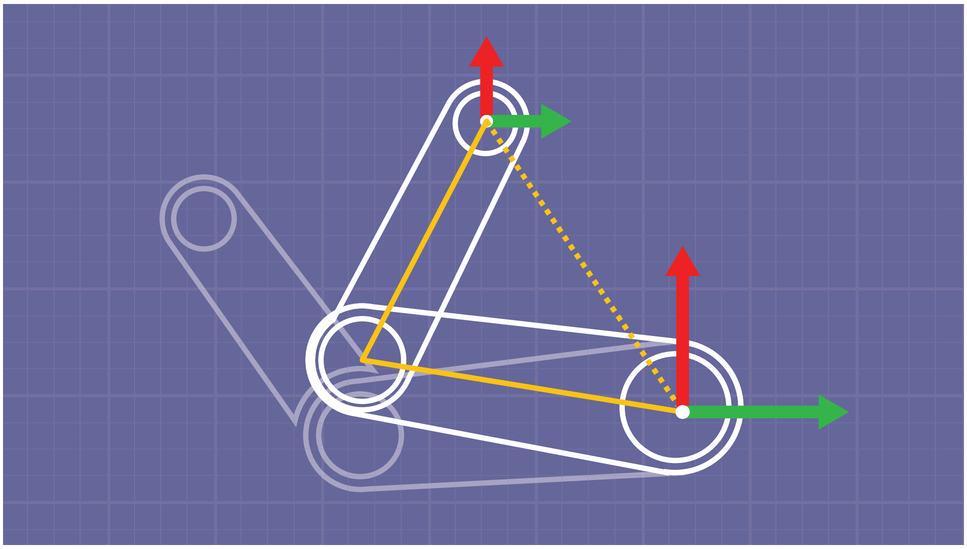


Figure 7.1 Two-bone inverse kinematics. Because the bone lengths are known, the delta from the previous bone rotations can be calculated through trigonometric identities.

UE4 offers another method for inverse kinematics that comes out of the box. This is Forward and Backward Reaching Inverse Kinematics (FABRIK). Unlike the two-bone IK implementation, FABRIK does not restrict the number of bones that can exist in the inverse kinematic chain. To do this, FABRIK takes a slightly different approach: rather than being analytical, like two-bone IK, FABRIK must be calculated iteratively by traversing up and down the chain of bones and converging toward a solution.

FABRIK first sets the last bone in the chain to match the position of the end goal, then works its way back up the chain, moving each bone toward the solution while attempting to maintain the bone's length and a straight line to the previous bone (see Figure 7.2). Once this loop reaches the very first bone, it will be moved out of place; however, because this is the root of the chain and shouldn't move from its original position, the algorithm is then run again, this time in reverse (hence Forward and Backward IK). By default, in UE4 this process is repeated ten times, but this iteration count can be changed on any FABRIK node in an animation graph.

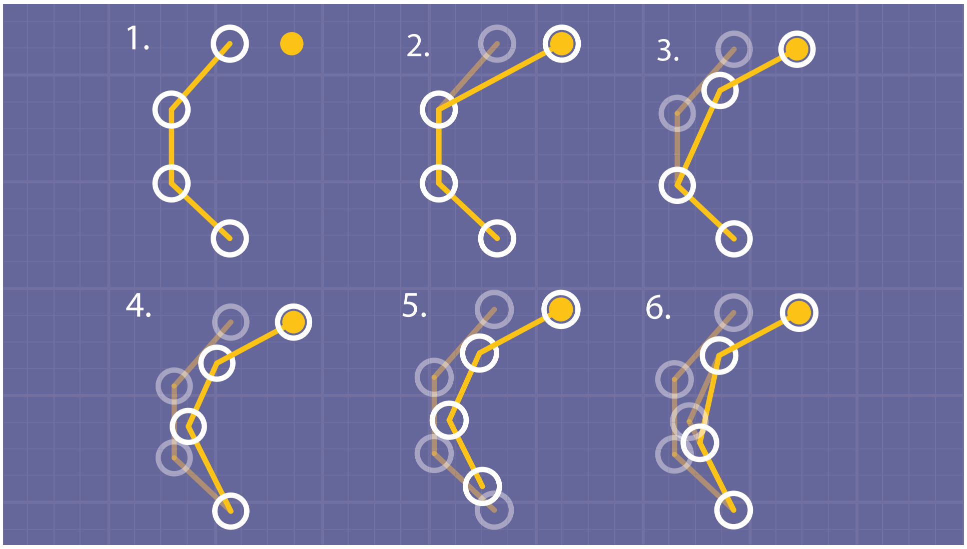


Figure 7.2 Forward and Backward Reaching Inverse Kinematics. 1: The initial bone state. 2: The end bone is moved toward the goal (yellow circle). 3–5: The next bone in the chain is then moved along the line connecting this bone to the previous bone while maintaining the initial bone length. 6: Because the root bone has been moved, it is moved back to its original position and the process is done in reverse.

Setting Up Head IK

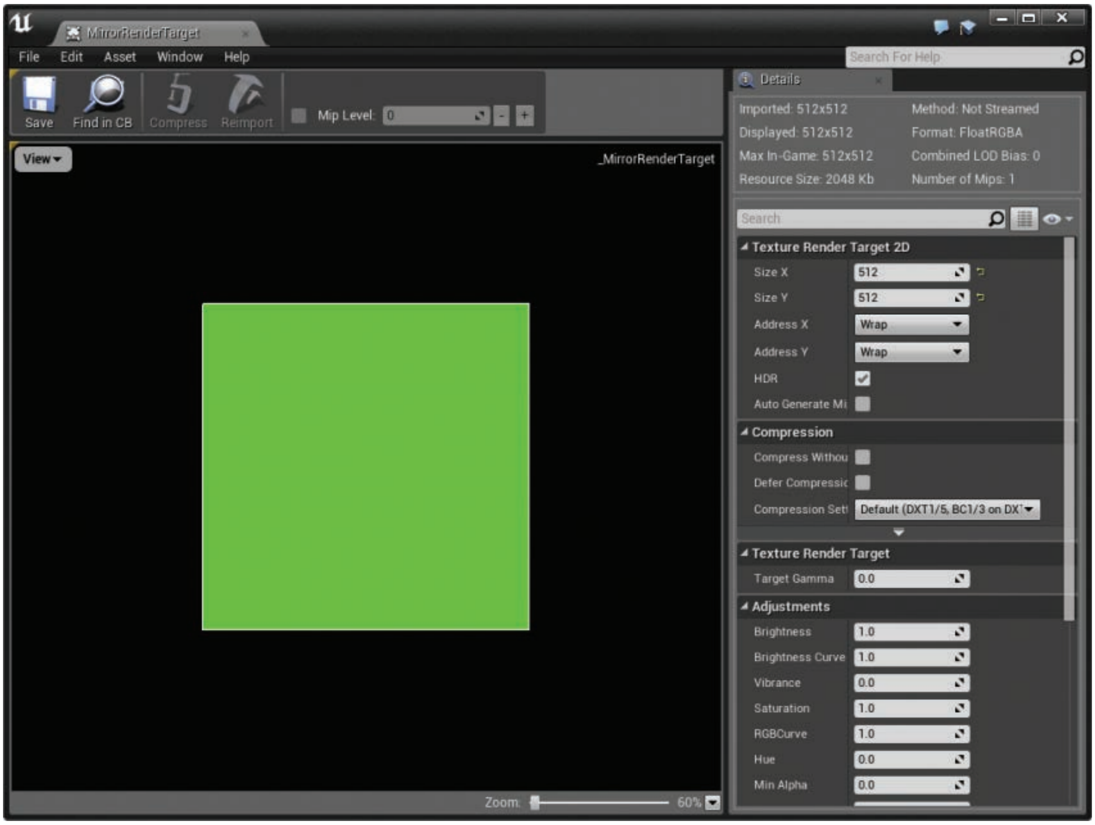
To test out UE4's IK system, we will create a basic head IK system that allows the in-game character to position its head bone at the player's in-world head location as well as bend its torso to match. Note that this works only if a player is standing still and not walking around the room, because the player's torso is not tracked.

Before starting, create a new project based on the Third Person Template with no starter content. This allows you to use the player model that comes with that template to test your IK Blueprints.

Mirror Creation

To know if your IK system is working, it would be nice to see what your in-game character looks like while playing in VR. To do this, you can create a simple mirror Material and render Texture:

1. Create two new folders in the Content Browser, the first named Materials and the second named Textures.
2. Inside the Textures folder, add a new Render Target called MirrorRenderTarget (Add New → Materials & Textures → Render Target).
3. Set both Size X and Size Y to 512 (see Figure 7.3).



Imported: 512x512
Method: Not Streamed
Displayed: 512x512
Format: FloatRGBA
Max In-Game: 512x512
Combined LOD Bias: 0
Resource Size: 2048 Kb
Number of Mips: 1

Texture Render Target 2D

Size X	512
Size Y	512
Address X	Wrap
Address Y	Wrap
HDR	✓
Auto Generate Mi	

Compression

Compress Withou	
Defer Compressic	
Compression Sett	Default (DXT1/5, BC1/3 on DX1

Texture Render Target

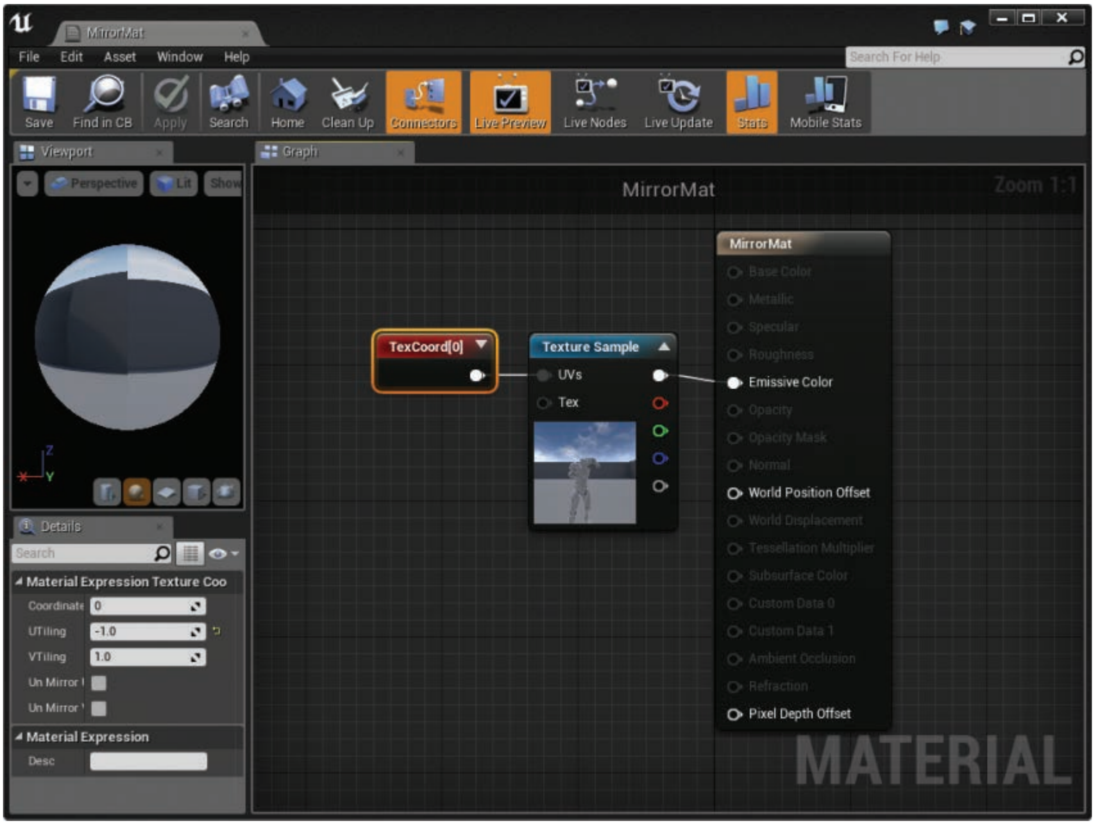
Target Gamma	0.0

Adjustments

Brightness	1.0
Brightness Curve	1.0
Vibrance	0.0
Saturation	1.0
RGBCurve	1.0
Hue	0.0
Min Alpha	0.0

Figure 7.3 Mirror Render Target

4. Inside the Materials folder, create a new blank Material called MirrorMat.
5. Open this Material and set its Shading Model to Unlit.
6. From the Content Browser, drag the MirrorRenderTarget into your MirrorMat.
7. Create a new TextureCoordinate node (U + click the graph) and set its UTiling to –1.
8. Pass this TextureCoordinate into the UVs input of your TextureSample (see Figure 7.4). This flips your Mirror Texture to make it more mirrorlike.



Material Expression Texture Coo

Coordinate 0

UTiling -1.0

VTiling 1.0

Un Mirror U

Un Mirror V

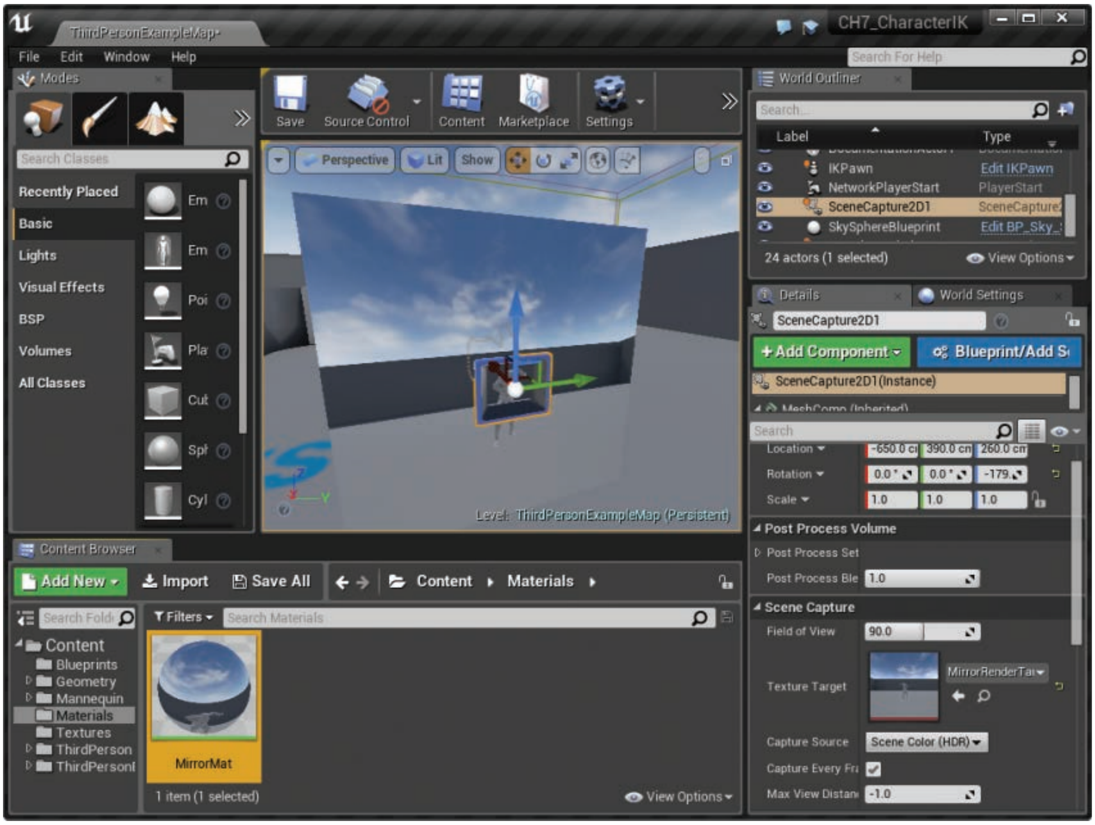
Material Expression

Desc

Figure 7.4 Mirror Material

9. Save and close this Material. Head back to the main engine window.
10. From the Modes tab, search and drag in a SceneCapture2D Actor, setting its Location to (X = –650.0, Y = 390.0, Z = 260.0) and Rotation to (Pitch = 0.0, Yaw = –180.0, Roll = 0.0). This positions it in front of the player.
11. Set this SceneCapture2D's Texture Target to the MirrorRenderTarget Texture. This puts whatever this camera captures into this Texture.

12. From the Modes panel again, drag in a basic Cube and set its Location to (X = –640.0, Y = 390.0, Z = 260.0) and Scale to (X = 0.01, Y = 2.0, Z = 2.0); see Figure 7.5.



Label	Type
IKPawn	Edit IKPawn
NetworkPlayerStart	PlayerStart
SceneCapture2D1	SceneCapture2D
SkySphereBlueprint	Edit BP_Sky_

24 actors (1 selected)

SceneCapture2D1

SceneCapture2D1(Instance)

Location	-650.0 cm	390.0 cm	260.0 cm
Rotation	0.0°	0.0°	-179.
Scale	1.0	1.0	1.0

Post Process Volume
- Post Process Set
- Post Process Ble: 1.0

Scene Capture
- Field of View: 90.0
- Texture Target: MirrorRenderTa
- Capture Source: Scene Color (HDR)
- Capture Every Fra: ✓
- Max View Distan: -1.0

Figure 7.5 Scene capture and mirror mesh

13. Apply the MirrorMat to the Cube's Material Element 0.

IK Pawn

Before the IK is set up, you need a Pawn to hold the Components you reference in the Animation Blueprint:

1. Create a new folder in the Content Browser named Blueprints.
2. Inside the Blueprints folder, create a new Animation Blueprint (Add New → Animation → Animation Blueprint), then select the UE4_Mannequin_Skeleton option in the

Animation Blueprint creation dialog (see Figure 7.6). This will hold all the IK logic that will be used later.

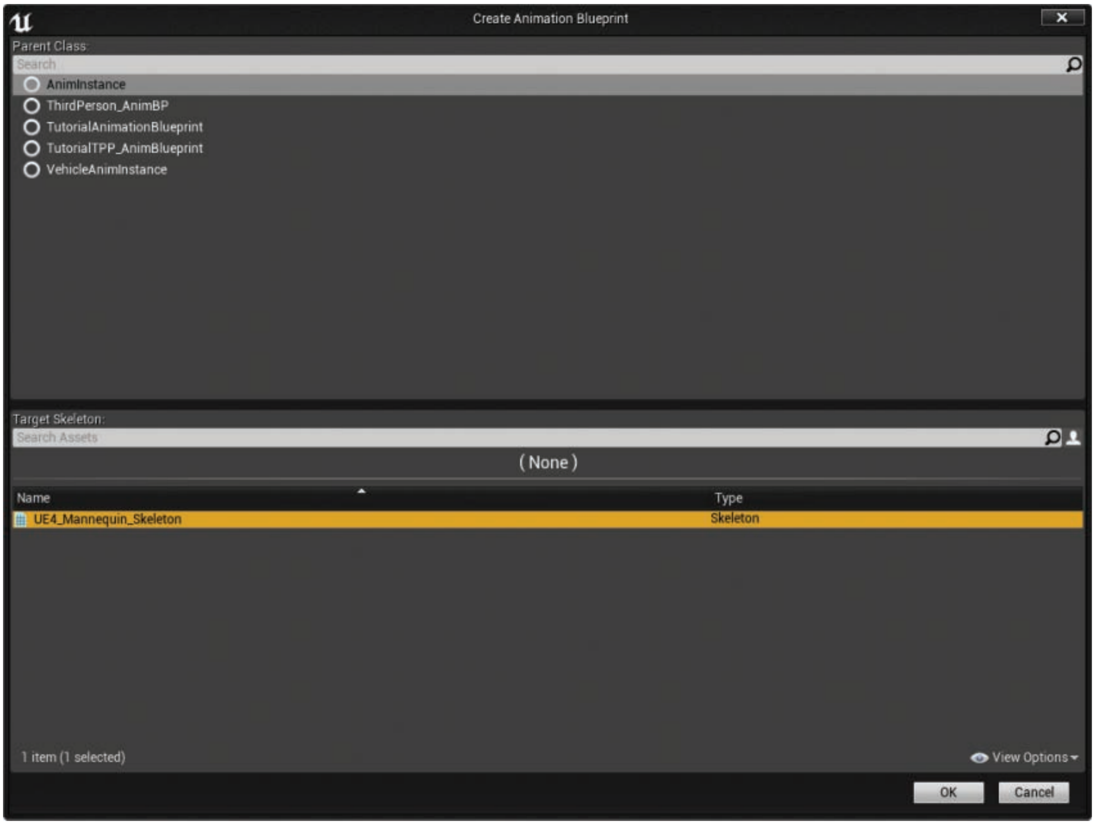
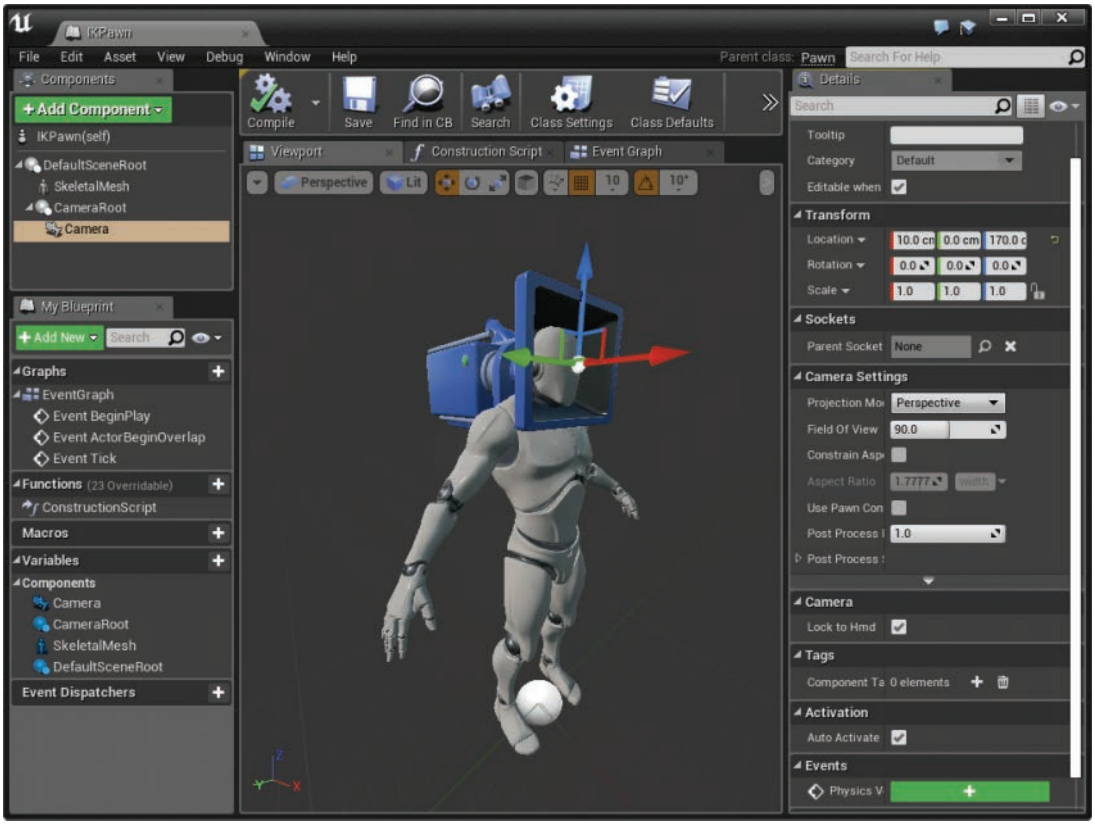


Figure 7.6 Animation Blueprint creation

3. Name this Animation Blueprint IKAnimBP.
4. Inside the Blueprints folder again, create a new Pawn named IKPawn.
5. Open this new Pawn and add three new Components: a SkeletalMesh, Camera, and Scene.
6. Name the Scene Component CameraRoot and attach the Camera to it.

7. Set the Camera to be at 170 Z height and 10 units forward in the X-axis (see Figure 7.7). This will affect the height of the Camera for testing with a monitor; however, because the Camera's position is reset to its root component when a VR game is launched, the Camera will be in the appropriate position for VR.



Tooltip
Category: Default
Editable when: ✓
Transform
Location: 10.0 cm, 0.0 cm, 170.0 c
Rotation: 0.0, 0.0, 0.0
Scale: 1.0, 1.0, 1.0
Sockets
Parent Socket: None
Camera Settings
Projection Mo: Perspective
Field Of View: 90.0
Constrain Asp
Aspect Ratio: 1.7777
Use Pawn Con
Post Process: 1.0
Post Process
Camera
Lock to Hmd: ✓
Tags
Component Ta: 0 elements
Activation
Auto Activate: ✓
Events
Physics V

Figure 7.7 IKPawn simple Camera setup

8. Rotate the SkeletalMesh –90 on the Z-axis. This ensures that the mesh faces down the X-axis.

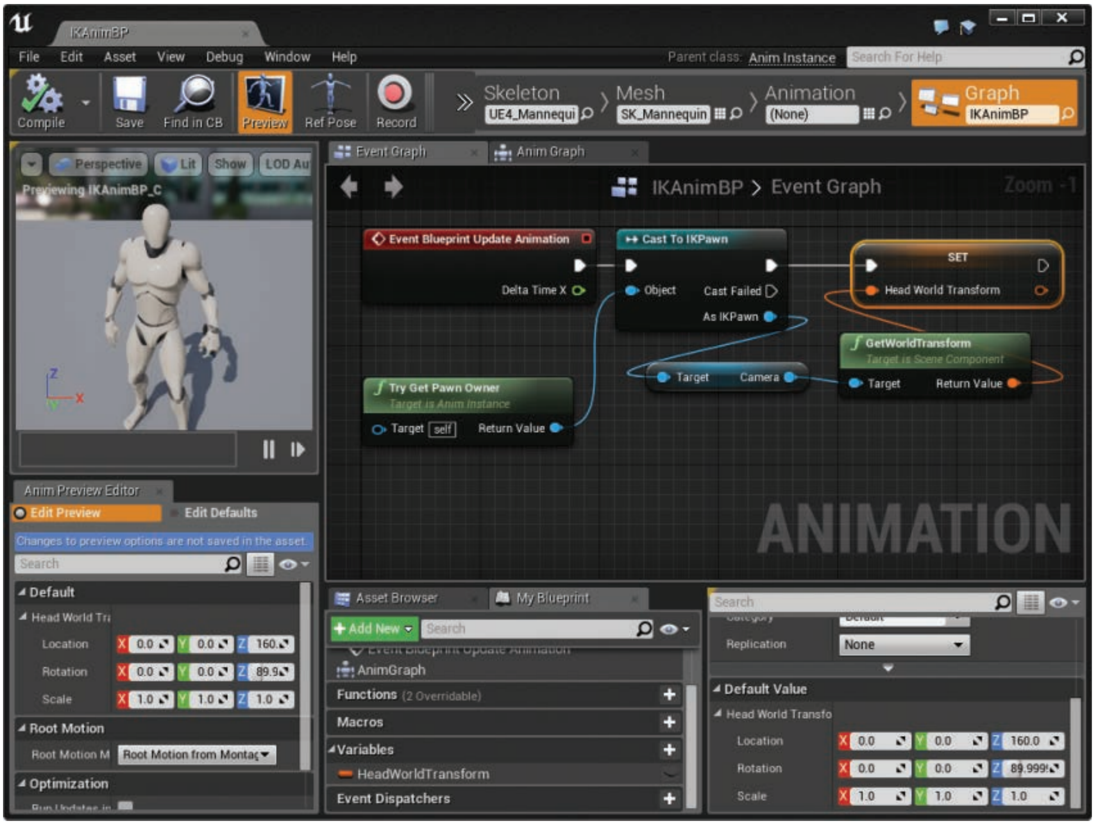
9. Set the SkeletalMesh's Skeletal Mesh property to the SK_Mannequin from the Third Person Template.

10. Set the SkeletalMesh's Anim Class to IKAnimBP.

Head IK Animation Blueprint

To apply the IK to the skeleton of the template character, you will use the FABRIK system because there are more than two bones from the head to the pelvis in this skeleton. Although you can chain two-bone IK nodes to get a similar effect, this can be a pain; the FABRIK node will work well for the head IK.

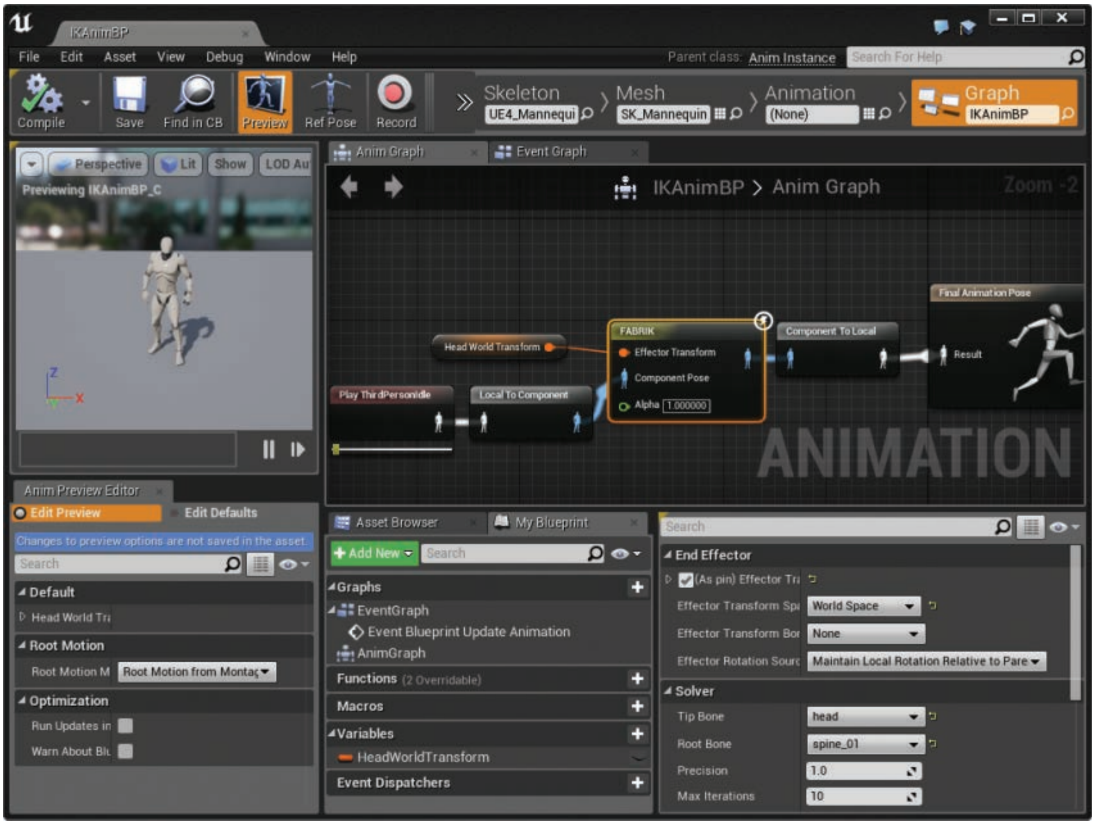
1. Open the IKAnimBP Animation Blueprint and create a new variable named HeadWorld-Transform of type Transform.
2. Set the HeadWorldTransform's default Location value to 160 units in the Z-axis and the default Rotation to 90 degrees in the Z-axis. This ensures that the head faces forward in preview windows.
3. Drag off of the TryGetPawnOwner node in the Event Graph and cast to the newly created IKPawn.
4. Drag off of the As IKPawn blue output pin of the cast and get the Camera Component from the Pawn.
5. Create a setter for the HeadWorldTransform variable and attach it to the Cast Succeeded pin of CastToIKPawn.
6. Drag off of the Camera Component getter and call GetWorldTransform, passing in the output to the setter for HeadWorldTransform (see Figure 7.8). You are using world space transforms in this example because they are easier to work with.



Head World Transform
Location X 0.0 Y 0.0 Z 160.0
Rotation X 0.0 Y 0.0 Z 89.999
Scale X 1.0 Y 1.0 Z 1.0

Figure 7.8 Animation Blueprint: head transform setup

7. Head into the Anim Graph and create a new PlayThirdPersonIdle node.
8. Create a new FABRIK node and connect the HeadWorldTransform variable to the Effector Transform input pin.
9. Select the FABRIK node and set the Effector Transform Space to World Space. This is because the Camera transform you are using is in world space.
10. Set the Tip Bone to “head” and the Root Bone to “spine_01” (see Figure 7.9). The FABRIK node will handle IK on all the bones between the two you just selected.

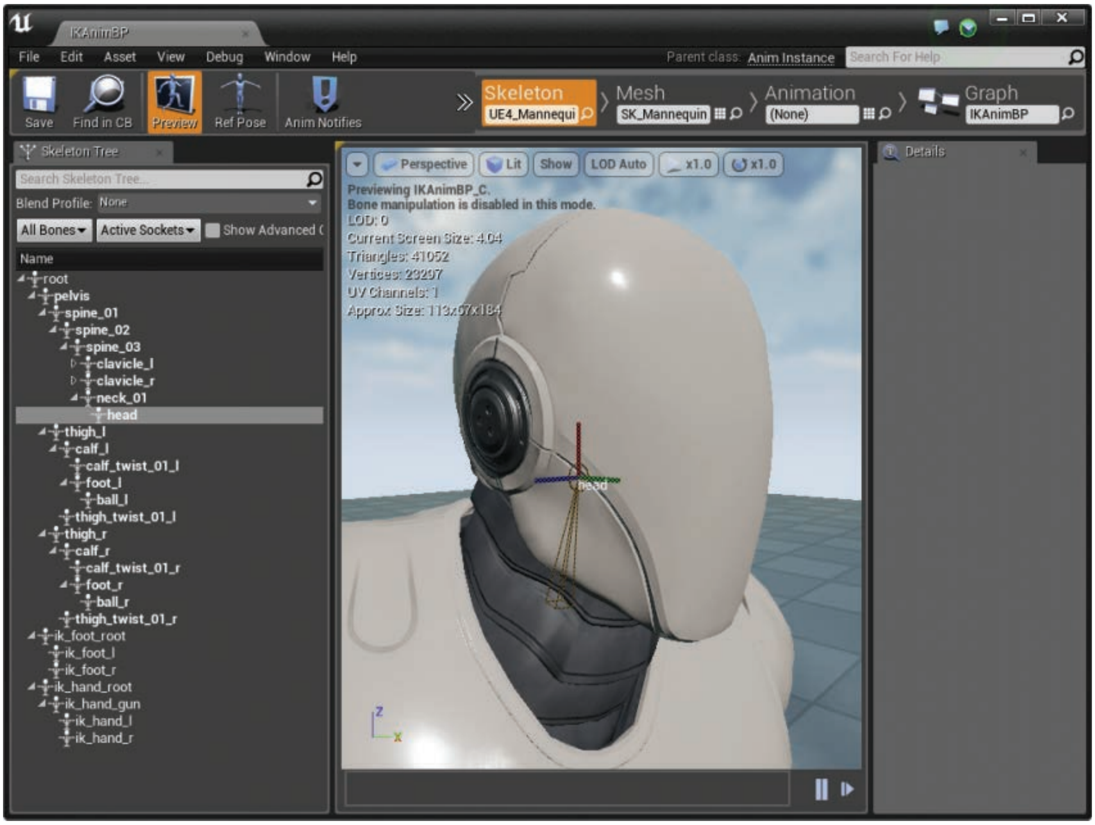


End Effector
(As pin) Effector Tr
Effector Transform Spa: World Space
Effector Transform Bo: None
Effector Rotation Sourc: Maintain Local Rotation Relative to Pare
Solver
Tip Bone: head
Root Bone: spine_01
Precision: 1.0
Max Iterations: 10

Figure 7.9 Animation Blueprint: simple head IK

11. Drag the output of the PlayThirdPersonIdle animation into the input of the FABRIK node. UE4 will automatically convert from Local to Component space for you.

12. Drag from the output of the FABRIK node into the input of the FinalAnimationPose node. Again, UE4 will automatically transform the coordinates for you. After this step the IK will work; however, it will not account for the rotation of the headset.
13. A quick note before the head rotation is added: In Figure 7.10, notice that the head bone's forward axis (red) is facing up, which means that you need to take this into account when setting this bone's rotation. Create a new Transform (Modify) Bone node in the Anim Graph.

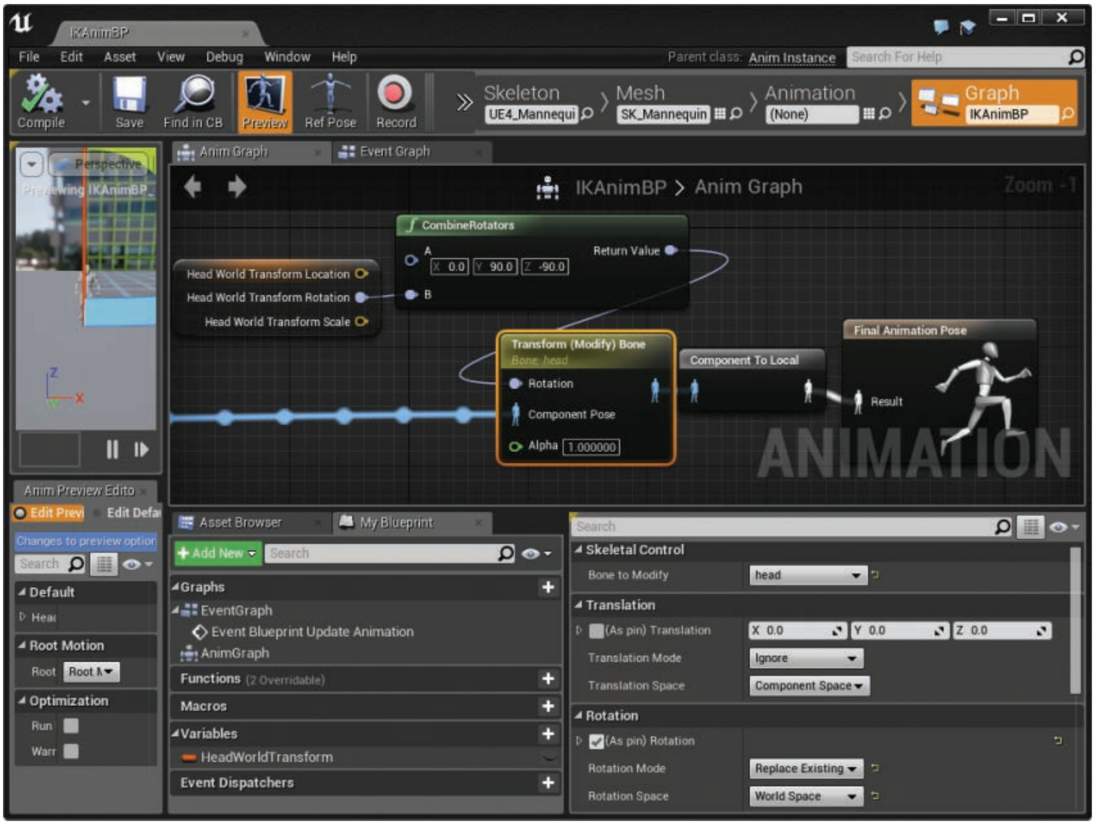


Previewing IKAnimBP_C.
Bone manipulation is disabled in this mode.
LOD: 0
Current Screen Size: 4.04
Triangles: 41052
Vertices: 23297
UV Channels: 1
Approx Size: 113x67x184

Figure 7.10 UE4 default skeleton head rotation

14. Set the transform node's Bone to Modify attribute to "head."

15. Set the Rotation Mode to Replace Existing and the Rotation Space to World Space (see Figure 7.11).



CombineRotators A: X 0.0, Y 90.0, Z -90.0

Alpha 1.000000

Bone to Modify	head
(As pin) Translation	X 0.0 Y 0.0 Z 0.0
Translation Mode	Ignore
Translation Space	Component Space
(As pin) Rotation	✓
Rotation Mode	Replace Existing
Rotation Space	World Space

Figure 7.11 Animation Blueprint: head bone rotation

16. Create a new CombineRotators node, passing in the value (Pitch = 90.0, Yaw = –90.0, Roll = 0.0) in the first input pin to account for the initial bone rotation.
17. Create a variable getter for the HeadWorldTransform variable and pass its rotation to the CombineRotators by splitting its output pin, then passing in the blue Rotation pin as the second argument (see Figure 7.11).
18. Connect the output of CombineRotators to the Rotation pin of Transform (Modify) Bone.
19. Connect the Transform (Modify) Bone between the FABRIK and ComponentToLocal node.

Now that the Pawn and Animation Blueprint are complete, test out your IK by deleting the default Pawn that is in the Third Person Example Map and dragging in your custom IKPawn. Then, set Auto Possess Player to Player 0 (this allows you to use this Pawn in your level without having a game mode set up) and hit Play. If you are using an Oculus Rift, tracking will default to Eye Level, so you may need to set the tracking level origin to Floor (see Chapter 2, “Head Mounted Display Setup”).

Setting Up Hand IK

If you tested the Pawn and IK from the previous section, you noticed that although the head IK works fairly well, if you have motion controllers, the temptation to move your hands is quite strong.

To set it up so that your motion controllers control the hands of the UE4 mannequin, you will use the simple two-bone IK system of UE4. This is because it is nice to have experience with both the FABRIK and two-bone systems, and the IK between the player’s hands and shoulders works quite well with the two-bone system.

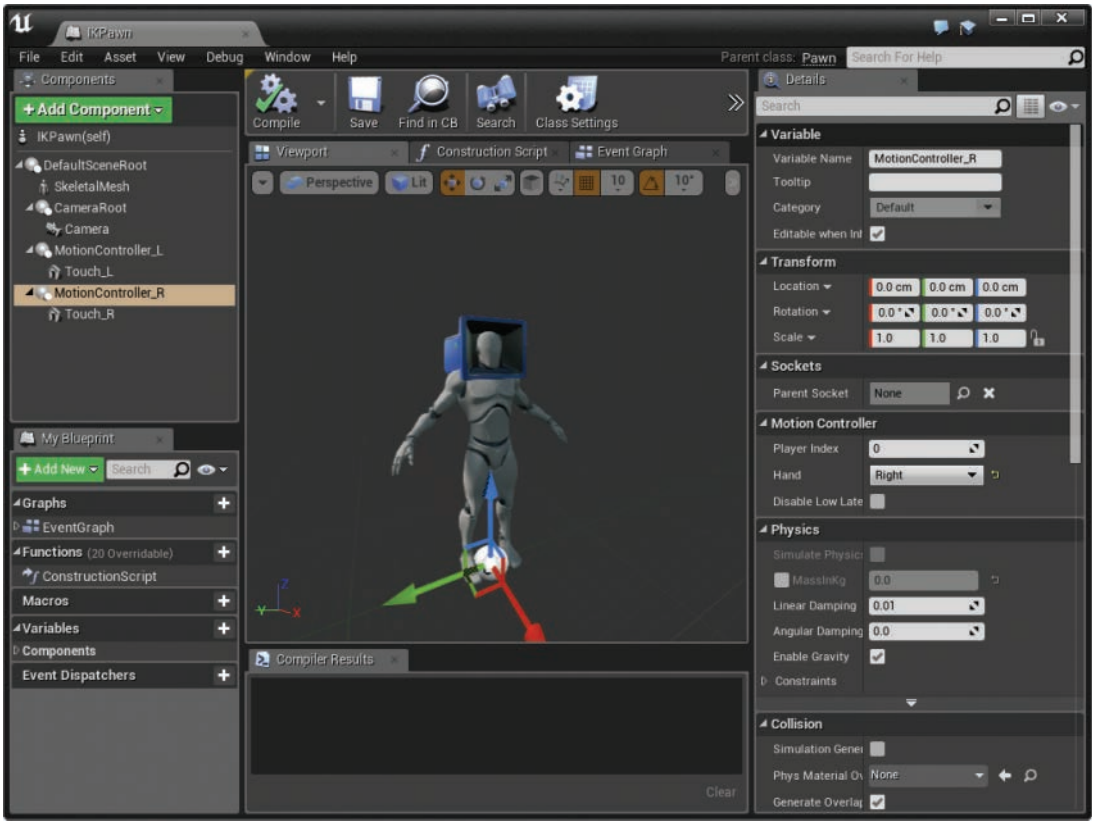
Before starting, we will work off of the previous section, so if you haven’t completed that, please do so.

Adding Motion Controllers to Your Pawn

Before you set up the IK for your Pawn, you need to add the Motion Controller Components that will be used to track the player’s hands:

1. Open the IKPawn from the Blueprints folder.
2. Add two new Motion Controller Components, the first named MotionController_L and the second MotionController_R.
3. For MotionController_R, set the Hand variable to Right.
4. Create two new Static Mesh Components to represent the controllers. These are mainly for debugging purposes.
5. If you are using Unreal Engine 4.13 or higher, you will have access to the motion controller meshes built into the engine (simply select Show Engine Content in the View options). If this is the case, select the appropriate Static Mesh property for your device for these two newly created Components. (For example, I am using the Oculus Touch, so I selected the OculusControllerMesh Static Mesh.) If you are not using 4.13 or above, simply select a sphere to represent your controllers or import the controller meshes from an external source.

6. Name these meshes appropriately for your controller (in this case, Touch_L and Touch_R were chosen) and attach these to the appropriate Motion Controller Component. Be sure to invert the Y scale on the right mesh if necessary (see Figure 7.12).



IKPawn(self)
DefaultSceneRoot
SkeletalMesh
CameraRoot
Camera
MotionController_L
Touch_L
MotionController_R
Touch_R

Variable
Variable Name: MotionController_R
Tooltip:
Category: Default
Editable when Inh: ✓
Transform
Location: 0.0 cm 0.0 cm 0.0 cm
Rotation: 0.0° 0.0° 0.0°
Scale: 1.0 1.0 1.0
Sockets
Parent Socket: None
Motion Controller
Player Index: 0
Hand: Right
Disable Low Late:
Physics
Simulate Physics:
MassInKg: 0.0
Linear Damping: 0.01
Angular Damping: 0.0
Enable Gravity: ✓
Constraints
Collision
Simulation Gener:
Phys Material Ov: None
Generate Overlap: ✓

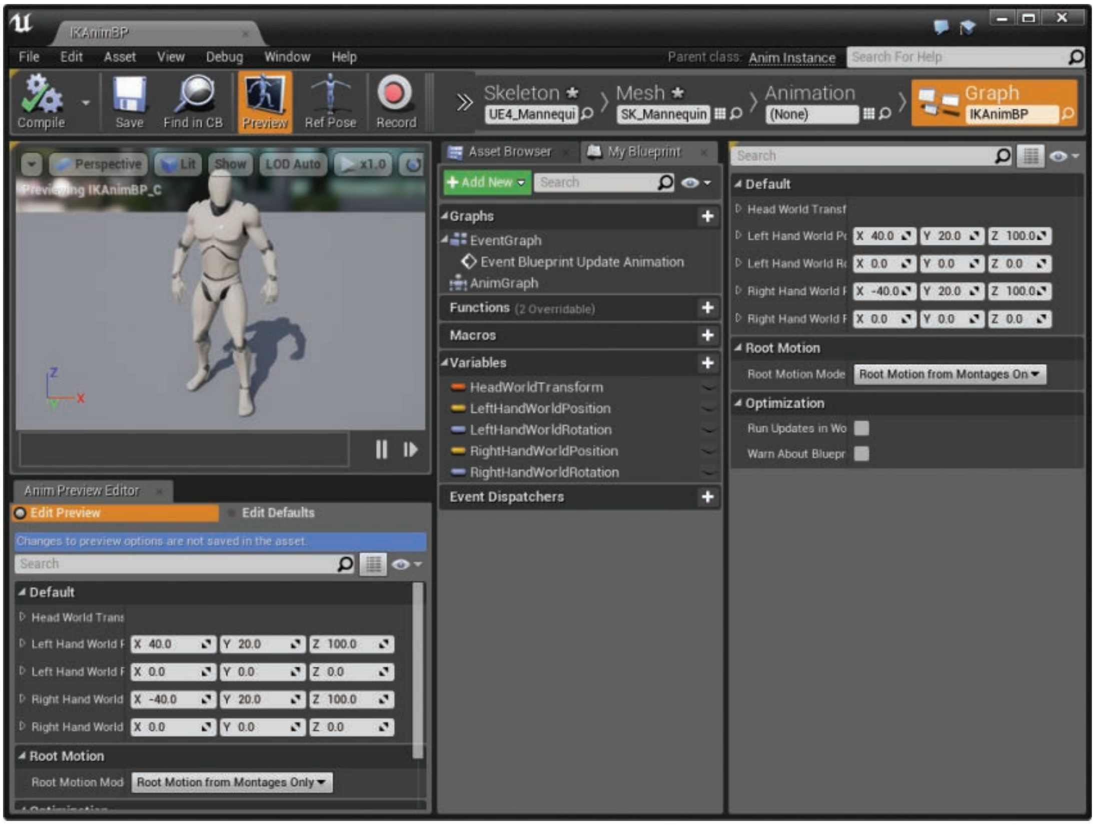
Figure 7.12 Adding motion controllers to the IKPawn

Hand IK Animation Blueprint

Your IKAnimBP class already implements a FABRIK system for the head and spine IK, so now it is time to implement a two-bone IK system for both arms of your skeleton:

1. Open the IKAnimBP Animation Blueprint and create two new variables, LeftHand-WorldPosition and LeftHandWorldRotation, of types Vector and Rotator respectively. For LeftHandWorldPosition, set a default value of (X = 40, Y = 20, Z = 100). This ensures that the mesh has an appropriate preview pose.

2. Create two more variables named RightHandWorldPosition and RightHandWorldRotation of types Vector and Rotator again. For RightHandWorldPosition set a default value of (X = –40, Y = 20, Z = 100) this time (see Figure 7.13). These will hold the world position and rotation of both motion controllers.

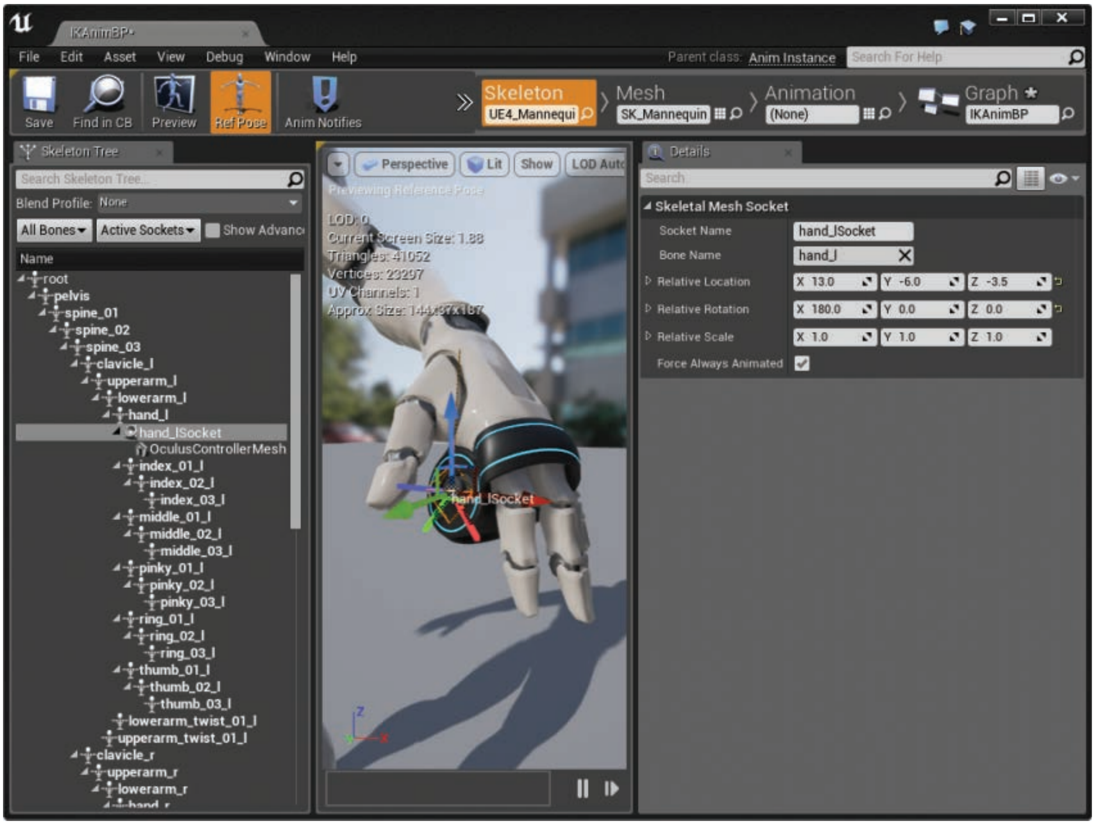


Left Hand World Pos: X 40.0, Y 20.0, Z 100.0
Left Hand World Rot: X 0.0, Y 0.0, Z 0.0
Right Hand World Pos: X -40.0, Y 20.0, Z 100.0
Right Hand World Rot: X 0.0, Y 0.0, Z 0.0
Root Motion Mode: Root Motion from Montages Only

Figure 7.13 IKAnimBP: adding the hand IK variables

3. Before you start adding the IK setup, you will add two Sockets to the Skeleton. This is because the origin point of the motion controllers does not line up with the hand bone location. This means that you will need to account for an offset when you use IK to get to the motion controller position.
4. Open the Skeleton editor by clicking on the Skeleton tab inside Persona.
5. Right-click the hand_l bone in the hierarchy and select Add Socket. Name this Socket hand_lSocket because you will reference it later.
6. This newly created Socket will initially be at the same position as its parent bone. To change this, select it and in the Details panel change the Relative Location to (X = 13.0, Y = –6.0, Z = –3.5). This places the Socket in the palm of the Skeleton's hand.

7. With this new Socket still selected, set its Relative Rotation to 180 degrees in the X-axis.
8. To test the position of this Socket if you are using Unreal Engine 4.13 or above or have access to the controller mesh of your desired motion controller, right-click the Socket and add a Preview Asset of your controller (see Figure 7.14).


IKAnimBP•
File Edit Asset View Debug Window Help
Parent class: Anim Instance
Search For Help
Save Find in CB Preview Ref Pose Anim Notifies
Skeleton UE4_Mannequi
Mesh SK_Mannequin
Animation (None)
Graph IKAnimBP
Skeleton Tree
Search Skeleton Tree...
Blend Profile: None
All Bones Active Sockets Show Advanced
Name
Perspective Lit Show LOD Auto
Previewing Reference Pose
hand_lSocket
Details
Search

Skeleton Tree:

```
root
 pelvis
  spine_01
   spine_02
    spine_03
     clavicle_l
      upperarm_l
       lowerarm_l
        hand_l
         hand_lSocket
          OculusControllerMesh
         index_01_l
          index_02_l
           index_03_l
         middle_01_l
          middle_02_l
           middle_03_l
         pinky_01_l
          pinky_02_l
           pinky_03_l
         ring_01_l
          ring_02_l
           ring_03_l
         thumb_01_l
          thumb_02_l
           thumb_03_l
        lowerarm_twist_01_l
       upperarm_twist_01_l
     clavicle_r
      upperarm_r
       lowerarm_r
        hand_r
```

Viewport:

```
LOD: 0
Current Screen Size: 1.88
Triangles: 41052
Vertices: 23297
UV Channels: 1
Approx Size: 144x37x187
```

Skeletal Mesh Socket

Socket Name	hand_lSocket		
Bone Name	hand_l		
Relative Location	X 13.0	Y -6.0	Z -3.5
Relative Rotation	X 180.0	Y 0.0	Z 0.0
Relative Scale	X 1.0	Y 1.0	Z 1.0
Force Always Animated	✓		

Figure 7.14 IKAnimBP: adding the controller offset Sockets to compensate for the hand bone placement

9. To make sure you have a Socket for both hands, repeat steps 5 through 8, but this time for the hand_r bone with a Relative Location of (X = –13.0, Y = 6.0, Z = 3.5) and a Relative Rotation of 180 degrees in the Z-axis.
10. To take advantage of these newly created Sockets, head back to the Animation Blueprint.
11. Create a variable setter for each of the left- and right-hand position and rotation variables.
12. Drag off of the CastToIKPawn node and get a reference to the MotionController_L, MotionController_R, and SkeletalMesh Components.
13. Drag off of the MotionController_L Component reference, call GetWorldRotation, and pass it directly into the LeftHandWorldRotation setter.

14. For the LeftHandWorldPosition you will need to do a little bit of vector math to subtract the difference between the Socket location and the actual bone location. Drag off of the SkeletalMesh reference and call GetSocketLocation twice.
15. For the first GetSocketLocation pass in the string hand_lSocket as the In Socket Name parameter, and for the second pass in hand_l (see Figure 7.15). This will get the world location of the hand bone and hand Socket.

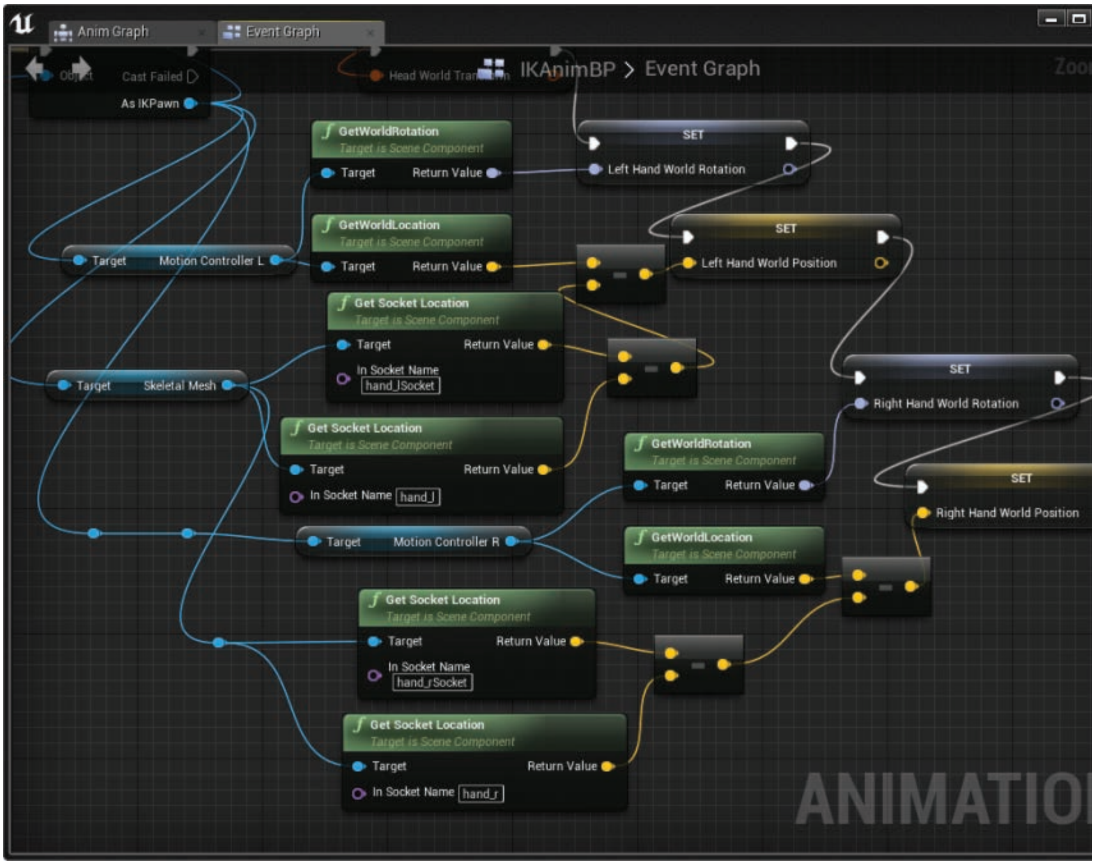


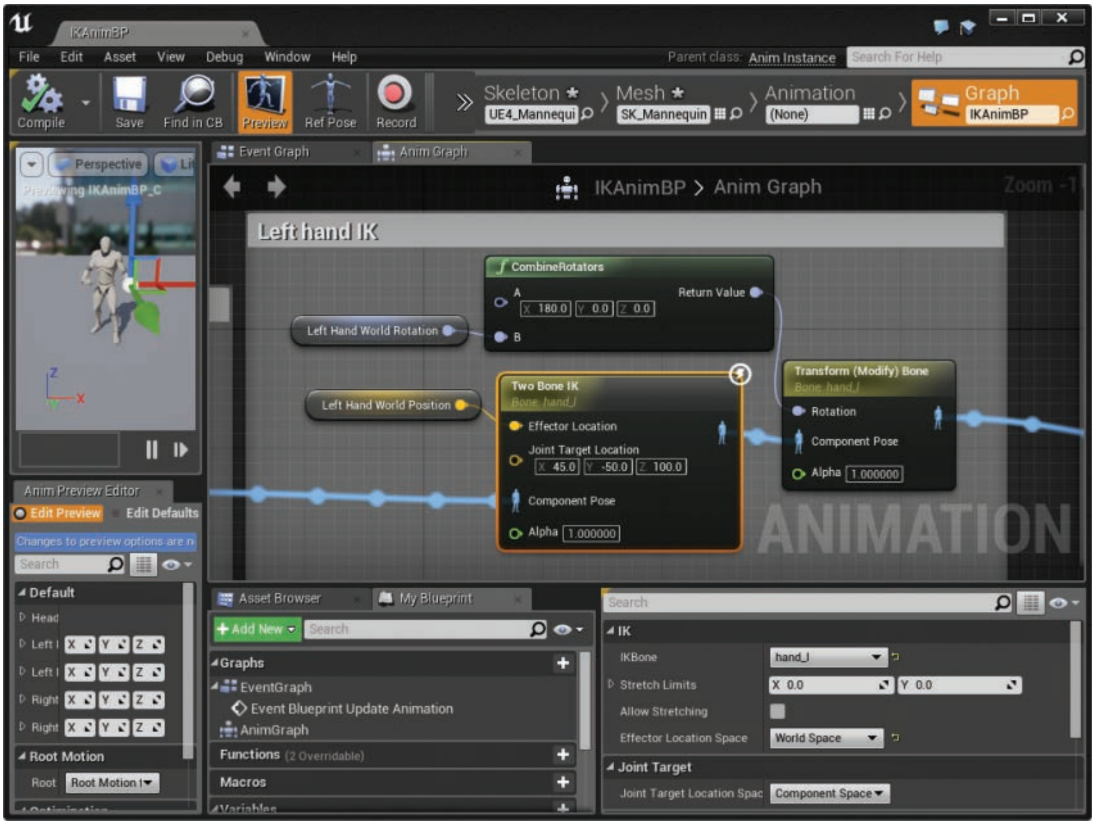
Figure 7.15 IKAnimBP: setting the hand variables and offsetting for controller positions in the IK

16. Drag off of the GetSocketLocation for the Socket and subtract the location of the hand bone. This will give you the relative distance from the hand to the Socket in world space.
17. Drag off of the MotionController_L reference, call GetWorldLocation, and subtract the relative distance calculated in step 16 from this world location.
18. Attach the output of this Subtract node to the LeftHandWorldPosition setter.

19. Repeat steps 13 through 18 but with the MotionController_R reference and the hand_rSocket and hand_r bone. If you need guidance, refer to Figure 7.15.
20. Attach these setters to the HeadWorldTransform setter already in the graph.

To implement the two-bone IK nodes, you need to head to the Anim Graph:

1. Once in the Anim Graph, create a new TwoBoneIK node.
2. Create a getter for your LeftHandWorldPosition variable and attach it to the Effector Location of the two-bone IK.
3. Pass in (X = 45, Y = –50, Z = 100) to the Joint Target Location. This is a point in Component space toward which the IK rotates. This value was obtained through trial and error, so feel free to change it to suit your needs.
4. Select TwoBoneIK and change the IKBone to hand_l and the Effector Location Space to World Space (see Figure 7.16).



CombineRotators: A X 180.0 Y 0.0 Z 0.0; B

Two Bone IK: Joint Target Location X 45.0 Y -50.0 Z 100.0; Alpha 1.000000

Transform (Modify) Bone: Alpha 1.000000

IK
IKBone hand_l
Stretch Limits X 0.0 Y 0.0
Allow Stretching
Effector Location Space World Space

Joint Target
Joint Target Location Spac Component Space

Figure 7.16 IKAnimBP: left-hand two-bone IK

5. Create a new Transform (Modify) Bone node, setting the Bone to Modify property to hand_l, the Rotation Mode to Replace Existing, and the Rotation Space to World Space. This will allow you to rotate the hand because the IK will not handle this.
6. Create a new variable getter for the LeftHandWorldRotation and connect it to the B input of a CombineRotators node.
7. Pass in 180 degrees to the X value of the A input of this CombineRotators node (see Figure 7.16). This value was obtained by looking at the Skeleton's initial bone rotation as you did in the "Setting Up Head IK" section.
8. Connect the output of CombineRotators to the Rotation input of Transform (Modify) Bone.
9. Repeat steps 2 through 8, replacing the LeftHandWorldRotation and Position with the right-hand variants, (X = –45, Y = –50, Z = 100) as the Joint Target Location, and 180 degrees in the Z for CombineRotators (see Figure 7.17).

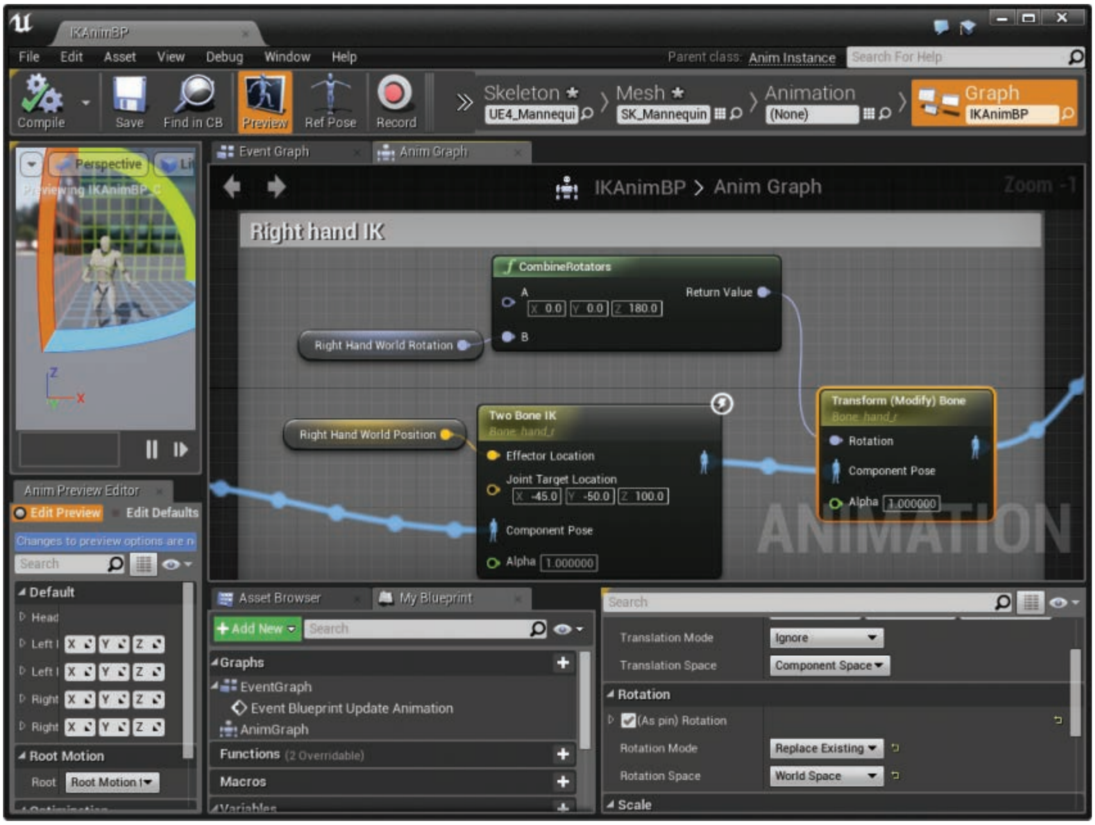


Figure 7.17 IKAnimBP: right-hand two-bone IK

10. Connect these new IK nodes between the Transform (Modify) Bone of the Head IK and the Component to the Local transform and you should be able to play in VR and see a full IK upper body (see Figure 7.18).

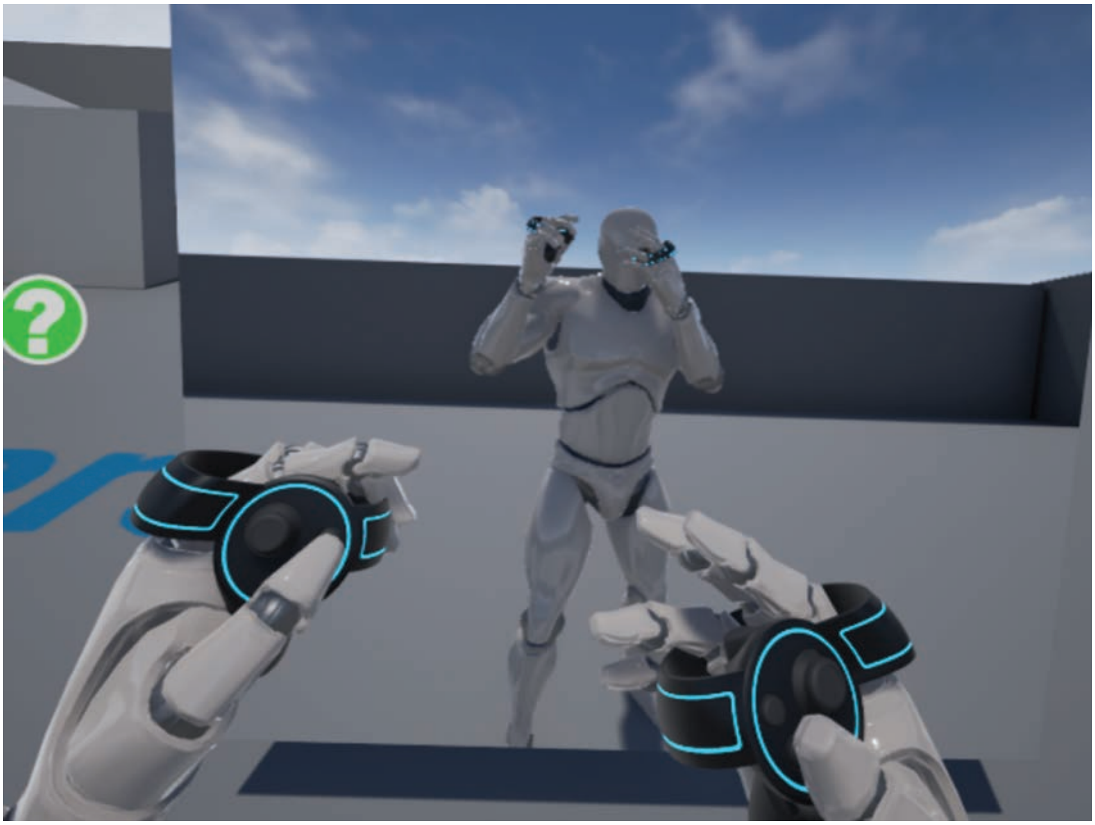


Figure 7.18 Upper-body IK example

Summary

In this chapter, you looked at a simple way to get started working with IK for VR in Unreal Engine. You looked at the two different types of IK offered natively, and you created a simple head and hands IK system that allows players' upper-body movements to be replicated to their in-game characters.

Exercises

Now that you have a taste of what IK can do, experiment with the values provided in this chapter and tune them to better suit your content.

If you are feeling adventurous, research some full-body IK setups. The main challenge here is to find a heuristic to detect when a player is bending his or her legs rather than simply leaning over.

If you are feeling less adventurous but still would like a full-body IK solver, there are a few middleware companies that provide full-body IK setups for UE4 (such as IKinema or the community Full Body IK Plugin).

This page intentionally left blank

Index

Numbers

2D texture splash screen, 46
2D UIs (user interfaces)
  custom menu interaction, 113–122
  issues with, 104
  readability, 104
2D UIs (user interfaces), creating a basic VR menu
  menu Actor, 108–111
  menu Pawn, 111–113
  UMG widget, 105–107
3D scenes, rendering, 9

A

Acceleration, detecting, 190
Add Loading Splash Screen function, 46
Affordances, 148
Aim assist, 81, 83
Ambient Occlusion, disabling, 224
Animations
  arm movement, 138–144
  blending, 9
  creating inverse kinematics, 9
  editing. *See* Persona.
  hand movement. *See* Hand IK.
  head movement. *See* Head IK.
  interpolating joint movement. *See* IK (inverse kinematics).
  player movement. *See* Locomotion.
Arm movement, inverse kinematics, 138–144
Artificial locomotion, 193
Auto Exposure, disabling, 224

B

Barrel distortion, 208–209
Base rotation and offset, getting, 46
Best practices, 10
Bloom, disabling, 224
Blueprints
  animation for hand IK, 138–144
  vs. C++, xiv
  description, xiv
  enabling physics on Components, 166
  Gear VR global menu setup, 21
  GetForwardVector node, 55
  inheritance, 56
  native support for, 9
Boxes, defining, 52. *See also* Traces.
Bullet Train, 155
Button mesh, creating, 167–168
Buttons
  activating, 173–174
  pressed, detecting, 173–174
  pressing, 170–174
  raising, 172–173
  triggering events with, 175
  uses for, 175
Buttons, creating
  button mesh, 167–168
  overlaps, detecting, 167–168
  visual representation, 168–170

C

C++ vs. Blueprints, xiv
Camera managers, enabling Follow HMD, 46–47
Cameras
  cinematic, best practices, 10
  clipping plane, setting, 45
  controlling with player's head motions, 10
  shake, avoiding, 10
Capsules, defining, 52. *See also* Traces.
Cardboard, native support for, 5
Channel traces, 53
Circular spinner, creating, 26–32
Classes vs. interfaces, 56
Clear Loading Splash Screens function, 46
Clustered rendering, 219–220
Collision conservation, 166
Collision responses, 53, 82
Commands, console, 226–228
Community resources, 245
Configuring VR. *See* VR project settings.
Conserving collision, 166
Console commands, 226–228
CPU bound vs. GPU bound, 227–228
Culling unseen objects, 221
Customizing VR. *See* VR project settings.

D

Daydream motion controller, native support for, 6
Daydream VR, native support for, 5
Deferred rendering, native support for, 9
Deferred shading vs. forward rendering, 215–220

Deployment to a phone, readying for, 14
Devices. *See* Hardware.
Direction vectors
  getting, 55
  length of, 52
Disorientation, from teleportation, 86
Documentation, Epic
  Gear VR project setup, 14
  Rift setup, 32–41
  Vive setup, 32–41
Dot products, 80–83
Dragging, hovered-on objects, 157–158
Draw calls, checking, 228
Dropped frames, compensating for, 8
Dropping, objects, 158–160

E

Editors
  animation. *See* Persona.
  VR. *See* VR editor.
Enable Auto Loading Splash Screen function, 46
Enable HMD function, 45
Enable Player Camera Manager Follow HMD function, 47
Enable Player Controller Follow HMD function, 47
Engine feature defaults, changing, 226–228
Engine Scalability Settings, 226
Epic
  Bullet Train, 155
  resources, 244
Epic, documentation
  Gear VR project setup, 14
  Rift setup, 32–41
  Vive setup, 32–41

F

FABRIK (Forward and Backward Reaching Inverse Kinematics), 126–127
Field of view, best practices, 10
First Person Template
  running in place, 201–205
  snap turning, 194–201
Focus state, getting, 44
Follow HMD, enabling for HMDs, 46
Forward kinematics, vs. inverse kinematics, 126
Forward rendering
  vs. deferred shading, 215–220
  native support for, 9
FOV (field of view), 208–209
Fragment shaders, 215–219
Frame rates, best practices, 10
Front buffer rendering, 8, 213–214
Functions. *See also specific functions.*
  generic, 44–45
  Oculus, 45–47
  Steam VR, 47

G

G-buffers, 217–219
Gear VR
  circular spinner, creating, 26–32
  global menu setup, 20–26
  native support for, 4
  progress indicator, creating, 26–32
  project setup, 14–20
  readying for deployment, 14
Geometric buffers, 217–219
Get Base Rotation and Base Offset in Meters function, 46
Get Hand Position and Orientation function, 47
Get HMD Device Name function, 45
Get Orientation and Position function, 44
Get Player Camera Manager Follow HMD function, 46
Get Pose function, 46
Get Positional Tracking Camera Parameters function, 44
Get Raw Sensor Data function, 46
Get Screen Percentage function, 44
Get Tracked Device Position and Orientation function, 47
Get Tracking Origin function, 44
Get Tracking Sensor Parameters function, 44
Get User Profile function, 46
Get Valid Tracked Device IDs function, 47
Get VR Focus State function, 44
Get World to Meters Scale function, 44
GetForwardVector node, 55
Google
  Cardboard, native support for, 5
  resources, 245
  VR, native support for, 5
  VR SDK, native support for, 7
GPU bound vs. CPU bound, 227–228
Gravity, disabling, 166

H

Hand IK, setting up. *See also* Head IK; IK (inverse kinematics).
  adding motion controllers, 137–138
  animation Blueprint, 138–144
Hardware, natively supported. *See also* Software, natively supported.
  Daydream motion controller, 6
  Daydream VR, 5

Gear VR, 4
Google Cardboard, 5
Google VR, 5
HMDs, 4–5
motion controllers, 5–6
Oculus Touch, 5
OSVR (Open Source Virtual Reality), 5
PlayStation Move, 6
PlayStation VR, 5
Rift, 4
Vive, 4, 6
Head IK, setting up. *See also* Hand IK; IK (inverse kinematics).
animation Blueprint, 132–137
Mirror creation, 128–130
Pawn creation, 130–132
Head tracking. *See* Head IK, setting up.
Hidden area mesh, 221
Hidden mesh optimization, native support for, 8
Hide Loading Icon function, 46
Hide Loading Splash Screen function, 47
`HMD SP` command, 44
HMDs (Head Mounted Displays)
base rotation and offset, getting, 46
detecting small motion changes, 46
enabled, testing for, 45
enabling, 45
Follow HMD, enabling, 46
generic functions, 44–45
getting raw data from the IMU, 46
Is Head Mounted Display Enabled function, 44
name, getting, 45
native support for, 4–5
neck position for virtual head model, getting, 46
position and rotation, getting, 46
scaling positional tracking, 46
testing for, 44
Hovered-on objects. *See* Objects, hovered on.
HTC/Valve
motion controllers, 149
resources, 244–245

I

IK (inverse kinematics)
arm movement, 138–144
creating, tool for, 9
FABRIK (Forward and Backward Reaching Inverse Kinematics), 126–127
vs. forward kinematics, 126
hand movement, 137–144
head movement, 128–137
introduction, 126–127
Joint Targets, 126
multi-bone method. *See* FABRIK.
previewing a character, 128–130
testing, 128–130
two-bone method, 126, 138–144
Inheritance
Blueprints, 56
interfaces, 56
Instanced Stereo rendering
native support for, 8, 220–221
optimization, 220–221
Interaction function, setting up, 64–71
Interaction interface, setting up, 58–61
Interaction Pawn
adding interaction to, 161–163
setting up, 73–75
Interactor component, creating, 153–163
Interfaces, 55–56
Is Auto Loading Splash Screen function, 46
Is Head Mounted Display Enabled function, 44
Is In Low Persistence Mode function, 45
Is Loading Icon Enabled function, 46
Is Player Controller Follow HMD Enabled function, 46

J

Joint movement, interpolating. *See* IK (inverse kinematics).
Joint Targets, 126
Judder, 8

L

Latency mitigation
front buffer rendering, 213–214
overview, 209–213
scanline racing, 213–214
a simple rendering pipeline, 209–214
timewarping, 212–213
Lens distortion, 208–209
Level loads, best practices for asynchronous loading, 10
Levers
components of, creating, 176–177
resetting, 182–185
rotating, 177–181
triggering, 186–187
visual representation of, 176–177
Light intensity, calculating, 216–220
Lines, traces
calculating a start and end point, 53–55
downsides of, 80–83
example, 52
LineTrace function, setting up, 62–63

Locomotion
  acceleration, detecting, 190
  First Person Template, 194–201, 201–205
  optic flow, 190
  running in place, 201–205
  snap turning, 194–201
  vection, 190
  visual sensation of movement, 190
Locomotion, types of. *See also* Teleportation.
  artificial, 193
  natural, 191
  physical, 192–193
  vehicle, 192
Low persistence mode, testing for, 45

M

Menu Actor, creating, 108–111
Menu interaction, VR editor, 239–241
Menu Pawn, creating, 111–113
Menus
  asynchronous loading, best practices, 10
  custom interaction, 113–122
  Gear VR global menu setup, 20–26
  global setup, 20–26
Menus, creating
  menu Actor, 108–111
  menu Pawn, 111–113
  UMG widget, 105–107
Mirror creation, 128–130
Mobile Scalable settings, 224
Motion Blur, disabling, 224
Motion changes, detecting, 46
Motion controllers
  adding to hand IK, 137–138
  affordances, 148
  HTC/Valve, 149
  introduction, 148
  natively supported, 5–6
  Oculus, 149
  picking up objects, 155
  player intent, determining, 155
  PlayStation, 149
  shared input paradigms, 149
  sphere overlap, alternative to, 155
  triggers, 149
Motion controllers, interactive objects. *See also* Buttons; Levers.
  conserving collision, 166
  gravity, disabling, 166
  physics simulations, disabling, 166
  Static Mesh Actor, 163–166
Motion controllers, world interaction project
  dragging on hovered objects, 157–158
  dropping objects, 158–160
  hover interface messages, changing, 155–156
  hovered-on objects, remembering, 153–156
  hovered-on objects, uses for, 161
  interacting with objects, 151–163
  Interaction Pawn, adding interaction to, 161–163
  interactor component, creating, 153–163
  interface, creating, 151–153
  object overlaps, detecting, 153–156
  picking up objects, 158–160
  setting up, 149–151
  with two controllers, 163
Motion sickness. *See* Simulator sickness.
MotionController Component, changing hand setting, 151
Motion-to-photon time, 208
Move, native support for, 6
Movement. *See* Locomotion.

N

Natural locomotion, 191
Navigation, with the VR editor, 237
Neck position for virtual head model, getting, 46

O

Object traces, 53
Object Types parameter, 53
Objects
  affordances, 148
  blocking lines, detecting, 52
  dropping, 158–160
  interacting with, 151–163. *See also* Motion controllers.
  interaction with the VR editor, 238–239
  interactive, setting up trace interaction, 75–79. *See also* Motion controllers, interactive objects.
  intersecting with other objects, 52
  overlaps, detecting, 153–156
  picking up, 158–160
  of player's attention, traces, 52–55
  players interacting with. *See* Trace interaction.
  properties that suggest use. *See* Affordances.
  searching for specific types of, 53
  selecting with the VR editor, 238–239
  transforming with the VR editor, 238–239
  between two points, detecting, 53–55. *See also* Traces.
  unseen, culling, 221
Objects, hovered on
  dragging, 157–158

hover interface messages, changing, 155–156
remembering, 153–156
uses for, 161
Occlusion, 123, 224
Oculus. *See also* Rift.
Audio SDK, native support for, 7
functions, 45–47
Mobile SDK, native support for, 7
motion controllers, 149
PC SDK, native support for, 6
resources, 244
Online resources. *See also* Resources.
companion website for this book, xv
Epic documentation, 14
practices that discomfort most players, 10
Rift projects, 33
Vive projects, 33
OpenVR SDK, native support for, 7
Optic flow, 190
Optimization
hidden mesh, native support for, 8
latency mitigation, 209–214
project settings. *See* VR project settings.
rendering. *See* Rendering, optimization.
visible mesh, native support for, 8
Origins
custom HMD, setting offset for, 47
resetting orientation and position, 45
Set Base Rotation and Base Offset in Meters function, 47
Orthographical projection of user interfaces, 104
OSVR (Open Source Virtual Reality), native support for, 5
OSVR SDK, native support for, 7
Overdraw, eliminating, 217
Overlapping objects, detecting, 153–156
Overlapping translucency, 225

P

Parabolic tracing, 86–92
Pawn creation, head IK, 130–132
Performance improvements. *See* Latency mitigation; Rendering, optimization; VR project settings.
Persona, native support for, 9
Physical locomotion, 192–193
Physics simulations, disabling, 166
Picking up objects, 158–160
Pixel shaders. *See* Fragment shaders.
Player controller
Follow HMD, enabling, 46
Follow HMD status, getting, 46
Player discomfort
common discomforting practices, 10
disorientation, 86
nausea, 10
orthographical projection of UIs, 104
simulator sickness, 190, 208
teleportation, 86
Players
aim assist, 81, 83
focus of attention, estimating, 80–83. *See also* Trace interaction; Traces.
head motion, best practices, 10
intent, determining, 81, 155
detecting interaction with objects, 80–82. *See also* Trace interaction.
movement. *See* Locomotion.
previewing a character, 128–130
profiles, getting, 46
selecting objects, 81–83
PlayStation
motion controllers, 149
Move, native support for, 6
VR, native support for, 5
Points, detecting objects between. *See* Traces.
Points in space. *See* Vertices.
Position, getting, 46
Positional tracking, scaling, 46
Pressing, buttons, 170–174
Previewing a character, 128–130
`ProfileGPU` command, 227
Programmable pipeline, 215
Progress indicator, creating, 26–32
Project setup. *See specific projects.*
Proportional scaling of the VR view
World to Meters scale factor, getting, 44
World to Meters scale factor, setting, 45
Protocols. *See* Interfaces.

Q

Quick Select Menu, 239–241

R

Radial Menu, 239–241
Raising buttons, 172–173
Rasterization, 217–218
Ray casts. *See* Traces.
Registering this book, xv
Remembering, hovered-on objects, 153–156
Rendering, natively supported
3D scenes, 9
deferred, 9
forward, 9
front buffer, 8
Instanced Stereo, 8, 220–221

Rendering, optimization
  barrel distortion, 208–209
  clustered rendering, 219–220
  culling unseen objects, 221
  deferred shading, 215–220
  forward rendering vs. deferred shading, 215–220
  FOV (field of view), 208–209
  fragment shaders, 215–219
  G-buffers, 217–219
  geometric buffers, 217–219
  hidden area mesh, 221
  Instanced Stereo, 220–221
  light intensity, calculating, 216–220
  motion-to-photon time, 208
  native resolution, 209
  overdraw, eliminating, 217
  programmable pipeline, 215
  rasterization, 217–218
  screen percentage, 209
  shaders, 215–219
  tiled rendering, 219–220
  transparent objects, 218
  vertex shaders, 215–219
  VR lens distortion, 208–209
  VR project settings, 222–225
Rendering pipeline, example, 209–214
Reprojection, native support for, 8
Reset Orientation and Position function, 45
Resetting levers, 182–185
Resolution, getting, 10
Resources. *See also* Online resources.
  community, 245
  conferences, 246
  Epic, 244
  Google, 245
  HTC/Valve, 244–245
  Oculus, 244
  physical meetings, 245
Rift
  low persistence mode, 45
  native support for, 4
  tracking origins, 39–41, 44
  tracking screen percentage, 44
Rift projects
  documentation, 32–33
  setting up, 33–39
Rotating levers, 177–181
Rotation, getting, 46
`r.screenPercentage` command, 44
Running in place, 201–205

S

Samsung Gear VR. *See* Gear VR.
Scaling
  Engine Scalability Settings, 226
  Get World to Meters Scale function, 44
  Mobile Scalable settings, 224
  positional tracking, HMDs, 46
  proportional scaling of the VR view, 44–45
  Set Position Scale 3D function, 46
  Set World to Meters Scale function, 45
  the world with the VR editor, 237
Scanline racing, 8, 213–214
Screen percentage
  changing, 226
  definition, 209
  getting, 10
Screen space effects, best practices, 10
SDKs, native support for, 6–7
Searching. *See* Traces.
Seated origin vs. standing, 40–41
Selecting objects, by player attention, 81–83
Selecting objects with the VR editor, 238–239
Separate Translucency, disabling, 225
Sequencer, native support for, 9
Set Base Rotation and Base Offset in Meters function, 47
Set Clipping Plane function, 45
Set Position Scale 3D function, 46
Set Tracking Origin function, 45
Set World to Meters Scale function, 45
Setting up VR. *See* VR project settings.
Shaders, 215–219
Shapes, traces for, 52–53, 83
Show Loading Icon function, 46
Show Loading Splash Screen function, 47
Simulator sickness, 190, 208
Snap turning, 194–201
Software, natively supported. *See also* Hardware, natively supported.
  animation editor. *See* Persona.
  Blueprints, 9
  cinematics tools. *See* Sequencer.
  Google VR SDK, 7
  hidden mesh optimization, 8
  Oculus Audio SDK, 7
  Oculus Mobile SDK, 7
  Oculus PC SDK, 6
  OpenVR SDK, 7
  OSVR SDK, 7
  Persona, 9
  reprojection, 8
  scanline racing, 8
  SDKs, 6–7
  Sequencer, 9
  timewarp, 8
  UMG (Unreal Motion Graphics), 9
  Unreal Engine systems, 9
  user interface system, 9
  visible mesh optimization, 8
  visual scripting language. *See* Blueprints.

Sphere overlap, alternative to, 155
Spheres, defining, 52. *See also* Traces.
Splash screens between level loads
  2D texture, 46
  adding, 46
  alternative 2D display, 47
  auto loading, detecting/enabling, 46
  clearing, 46
  customizing, 46
  enabling, 46, 47
  hiding, 47
  loading icon, showing/hiding, 46
  loading icon, testing for, 46
  turning on/off, 46, 47
Standing origin vs. sitting, 40–41
`Stat Game` command, 228
`Stat SceneRendering` command, 228
`Stat Unit` command, 227
Static Mesh Actor, 163–166
Steam VR controllers
  device IDs, getting, 47
  device orientation and position relative to HMD origin, getting, 47
  hand position and orientation, getting, 47
Steam VR functions, 47

T

Target Hardware and Graphics Performance settings, 222–223
Teleportation
  description, 191–192
  downsides, 86
  parabolic tracing, 86–92
  pros and cons, 191–192
  setting up, 86–92
  vertical, 86–92
Teleportation, visualizing
  visualization Actor, 94–95
  visualization Material, 92–94
Teleportation volume, 95–100
Testing inverse kinematics, 128–130
3D scenes, rendering, 9
Tiled rendering, 219–220
Timewarp, native support for, 8
Timewarping, 212–213
Tips. *See* Best practices.
Touch, native support for, 5
Trace Channel parameter, 53
Trace interaction
  overview, 52–55
  parabolic tracing, 86–92
Trace interaction, setting up
  basic interactive objects, 75–79
  basic project, 57–58
  interaction function, 64–71
  interaction interface, 58–61
  Interaction Pawn, 73–75
  LineTrace function, 62–63
  Trace Interaction Component, 61–73
Trace Interaction Component
  custom menu interaction, 113–122
  setting up, 61–73
Traces
  Channel, 53
  definition, 52
  direction vectors, getting, 55
  Object, 53
  for object of player's attention, 52–55
  setting up, 53–55
  shapes, 52–53, 83. *See also specific shapes.*
  for specific types of objects, 53
  types of, 52–53
  world origin, getting, 55
Traces, lines
  calculating a start and end point, 53–55
  downsides of, 80–83
  example, 52
Tracking head movement. *See* Head IK, setting up.
Tracking origins
  Eye Level, setting, 45
  Floor Level, 40–41
  Floor Level, setting, 45
  Get Orientation and Position function, 44
  Get Positional Tracking Camera Parameters function, 44
  Get Tracking Origin function, 44
  Get Tracking Sensor Parameters function, 44
  Rift, 39–41, 44, 45
  seated vs. standing, 40–41
  Set Tracking Origin function, 45
  Vive, 39–41, 44, 45
Tracking origins, defaults
  Rift, 40, 44
  Vive, 41, 44
Translucency, overlapping, 225
Transparent objects, rendering, 218
Triggering
  events with buttons, 175
  levers, 186–187
Triggers, motion controllers, 149
Trigonometry of rotating levers, 179–181
2D texture splash screen, 46
2D UIs (user interfaces)
  custom menu interaction, 113–122
  issues with, 104
  readability, 104
2D UIs (user interfaces), creating a basic VR menu
  menu Actor, 108–111
  menu Pawn, 111–113
  UMG widget, 105–107
Two-bone method of inverse kinematics, 126, 138–144

U

UIs (user interfaces)
- accessing with VR editor, 240–241
- determining the need for, 104
- head locked. *See* 2D UIs.
- intersection with game geometry. *See* Occlusion.
- native support for, 9
- orthographical projection, 104
- two-dimensional. *See* 2D UIs.

UMG (Unreal Motion Graphics)
- compatibility, 105
- history of, 105
- native support for, 9
- widget for basic menu, creating, 105–107

Unreal Engine systems, native support for, 9
Users. *See* Players.

V

Valve
- motion controllers, 149
- resources, 244–245

Vection, 190
Vectors, multiplying together. *See* Dot products.
Vehicle locomotion, 192
Vertex shaders, 215–219
Vertices, definition, 215
Vestibular system, 190
Visible mesh optimization, native support for, 8
Visual sensation of movement, 190
Visualizing teleportation
- visualization Actor, 94–95
- visualization Material, 92–94

Vive
- low persistence mode, 45
- native support for, 4, 6
- tracking origins, 39–41
- tracking origins, defaults, 41, 44
- tracking screen percentage, 44

Vive projects
- documentation, 32–33
- setting up, 33–39

VR editor
- accessing UI elements, 240–241
- controller, 235–236
- enabling, 234
- menu interaction, 239–241
- navigation, 237
- object interaction, 238–239
- opening, 235
- Quick Select Menu, 239–241
- Radial Menu, 239–241
- scaling the world, 237
- selecting objects, 238–239
- transforming objects, 238–239

VR lens distortion, 208–209
VR project settings
- Ambient Occlusion, disabling, 224
- Auto Exposure, disabling, 224
- Bloom, disabling, 224
- console commands, 226–228
- CPU bound vs. GPU bound, 227–228
- draw calls, checking, 228
- engine feature defaults, changing, 226–228
- Engine Scalability Settings, 226
- Mobile Scalable settings, 224
- Motion Blur, disabling, 224
- overlapping translucency, 225
- `ProfileGPU` command, 227
- rendering optimization, 222–225
- screen percentage, changing, 226
- Separate Translucency, disabling, 225
- `Stat Game` command, 228
- `Stat SceneRendering` command, 228
- `Stat Unit` command, 227
- Target Hardware and Graphics Performance, 222–223

W

Websites. *See* Online resources.
World origin, getting, 55
World to Meters scale factor
- getting, 44
- setting, 45

WYSIWYG (what you see is what you get), 105