HANSER

Leseprobe

zu

Virtual Reality-Spiele entwickeln mit Unity®

von Daniel Korgel

ISBN (Buch): 978-3-446-45147-6
ISBN (E-Book): 978-3-446-45372-2

Weitere Informationen und Bestellungen unter
http://www.hanser-fachbuch.de/978-3-446-43465-3
sowie im Buchhandel

Inhalt

Vorwort ..... **XV**

1 Einleitung **1**

1.1 Virtual Reality 2
1.2 Ziel des Buches 2
1.3 Unterstützte Virtual-Reality-Brillen 3
1.4 Unity 4
1.4.1 Die verschiedenen Versionen von Unity 5
1.5 Was brauche ich? 6
1.6 Weiterentwicklung der Technik 7
1.7 Online-Zusatzmaterial 7

2 Einführung in Virtual Reality **9**

2.1 Begriffserklärungen 9
2.2 Augmented Reality, Mixed Reality und Virtual Reality 11
2.3 Virtual Reality ist tot – lang lebe Virtual Reality 12
2.4 Unterstützte Virtual-Reality-Brillen 15
2.4.1 Oculus Rift 16
2.4.2 HTC Vive 17
2.4.3 Samsung GearVR 18
2.4.4 Google Cardboard (nur teilweise unterstützt) 19
2.4.5 Google Daydream 20
2.5 Andere VR-Brillen 21
2.5.1 Microsoft Mixed Reality 21
2.5.2 PlayStation VR 22
2.5.3 Razer OSVR 23
2.6 Spiele, Anwendungen, Demos und Erfahrungen 24
2.7 Seated, Standing & Room Scale 25
2.8 „Bekannte“ Probleme 26
2.8.1 Auflösung & Fliegengitter-Effekt 27

2.8.2 Hardwareanforderungen ... 27
2.8.3 Kurz- und Weitsichtigkeit ... 28
2.8.4 Simulator Sickness ... 28

3 Quickstart: 3D-Entwicklung mit Unity ... 31
3.1 Installation ... 31
3.1.1 Die Unity ID ... 33
3.1.2 Android SDK & Java Development Kit(GearVR, GoogleVR) ... 34
3.2 Der erste Start ... 38
3.3 Ein neues Projekt anlegen ... 39
3.4 Die Unity-Oberfläche ... 40
3.4.1 Drag & Drop ... 41
3.4.2 Project Browser ... 42
3.4.3 Console ... 43
3.4.4 Hierarchy ... 44
3.4.5 Scene View ... 46
3.4.6 Inspector ... 48
3.5 Scenes ... 54
3.5.1 Unity in der dritten Dimension ... 54
3.5.2 GameObjects ... 55
3.5.3 Prefabs ... 57
3.6 Die wichtigsten Components ... 58
3.6.1 Camera und Audio Listener ... 58
3.6.2 Lights ... 59
3.6.3 Mesh Filter und Mesh Renderer ... 60
3.6.4 Collider ... 61
3.6.5 Rigidbody ... 62
3.6.6 Audio Source ... 63
3.6.7 Scripts/Behaviours ... 64
3.7 Assets ... 64
3.7.1 Dateien importieren ... 67
3.7.2 Unity Packages ... 68
3.7.3 Asset Store ... 71
3.7.4 Import Settings ... 72
3.8 Projektvorbereitung (GearVR, GoogleVR) ... 72
3.8.1 Zielplattform zu Android wechseln ... 73
3.8.2 Externe Tools angeben (Android SDK, JDK) ... 74
3.9 Test Scene erstellen und starten ... 76
3.9.1 Rohbau für den Raum erstellen ... 78
3.9.2 Scene speichern ... 81
3.9.3 Licht anpassen ... 82
3.9.4 Collider hinzufügen ... 83
3.9.5 Möbelstücke einräumen ... 84

3.9.6 Bücher hinzufügen ... 87
3.9.7 Weitere Details ... 91
3.9.8 Lichter und Lichteinstellungen optimieren ... 92
3.9.9 Spieler positionieren ... 95
3.10 VR-Unterstützung aktivieren ... 96
3.10.1 Unitys VR-Unterstützung aktivieren ... 96
3.11 Player Settings anpassen (GearVR, GoogleVR) ... 97
3.12 Anti-Aliasing aktivieren ... 99
3.13 Die Scene in VR testen ... 100
3.13.1 Am PC in VR testen (Oculus Rift, SteamVR) ... 101
3.13.2 Auf dem Smartphone in VR testen (GearVR, GoogleVR) ... 102
3.14 So geht es weiter ... 111

4 Dos and Don'ts – Game-Design für Virtual Reality ... 113
4.1 Beispielprojekt ... 114
4.2 Immersion ... 114
4.2.1 Immersion brechen ... 118
4.3 Positional Tracking ... 118
4.3.1 Mobile VR-Brillen ... 120
4.4 Locomotion ... 121
4.4.1 Klassische Locomotion ... 121
4.4.2 Teleport Locomotion ... 124
4.4.3 Ohne künstliche Locomotion ... 125
4.4.4 Andere Eingabemethoden für das Laufen ... 128
4.5 Lebendige Spielwelt erschaffen ... 129
4.6 User Interface ... 133
4.6.1 Distanz und Größe ... 136
4.7 Aufmerksamkeit lenken ... 137
4.7.1 Objekte untersuchen ... 138
4.7.2 Ereignisse nicht verpassen ... 141
4.8 Uncanny Valley ... 142
4.8.1 Auswirkung für dich ... 145
4.9 Sound Design und Spatial Sound ... 146
4.9.1 Spatial Sound ... 146
4.9.2 Spatial-Audio-Engines ... 149

5 C#-Scripte für Unity programmieren ... 151
5.1 C#, .Net und Mono ... 152
5.2 Grundlagen ... 152
5.2.1 C#-Syntax ... 152
5.2.2 Kommentare ... 153
5.2.3 Variablen ... 154
5.2.4 Methoden ... 161

5.2.5 Klassen ... 164
5.2.6 Dynamische Datentypen ... 165
5.2.7 Statische Methoden und statische Variablen ... 167
5.2.8 Vererbung ... 168
5.2.9 Namespaces ... 171
5.2.10 Lebenszyklus von Variablen ... 172
5.2.11 Sichtbarkeiten ... 173
5.2.12 Ausnahmen – „Exceptions“ ... 174
5.3 Erster Test ... 175
5.3.1 Microsoft Visual Studio 2017 ... 175
5.3.2 Script, Behaviour, Component und ihr Aufbau ... 179
5.3.3 Debug-Ausgaben in die Console ... 180
5.3.4 Eigenes Script testen ... 181
5.3.5 Notation von Methoden und Variablen in diesem Buch ... 183
5.4 Erweiterte Grundlagen ... 183
5.4.1 Bedingungen und Verzweigungen ... 183
5.4.2 Schleifen ... 186
5.4.3 Eigenschaften ... 191
5.4.4 Dynamische Listen ... 192
5.5 Unity-spezifische Grundlagen ... 196
5.5.1 Zufallswerte ... 196
5.5.2 Editor-Eigenschaften ... 197
5.5.3 MonoBehaviour Lebenszyklus-Methoden ... 200
5.5.4 MonoBehaviour-Event-Methoden ... 202
5.5.5 Auf GameObjects zugreifen ... 203
5.5.6 GameObjects via Code erstellen ... 206
5.5.7 GameObjects zerstören ... 207
5.5.8 Transforms: GameObjects bewegen ... 207
5.5.9 Auf andere Components zugreifen ... 209
5.5.10 Components zu GameObject hinzufügen ... 212
5.5.11 Components entfernen ... 213
5.5.12 Parallele Methoden mit Coroutines ... 213
5.5.13 Scripted-Event-Beispiel ... 214
5.5.14 Methoden verzögert aufrufen mit Invoke ... 215
5.6 Problemlösung ... 216
5.6.1 Wort im Code ist rot unterstrichen ... 216
5.6.2 Fehler in der Console im Editor ... 217
5.6.3 Fehler in der Console, während das Spiel läuft ... 217
5.6.4 „Can’t add script“-Meldung oder Script ist nicht im „Add Component“-Menü vorhanden ... 218
5.6.5 Sonstige Fehler ... 219

6 Erweiterte Unity-Einführung ... 221

6.1 Licht, Schatten und Lightmapping ... 221

6.1.1 Global Illumination ... 222
6.1.2 Farbräume ... 232
6.1.3 Environment Light ... 233
6.1.4 Lichter in der Scene ... 234
6.1.5 Emissive Materials ... 240
6.1.6 Light Probes ... 240
6.1.7 Reflexionen ... 243
6.1.8 Lighting-Fenster ... 246
6.1.9 Light Explorer ... 248
6.2 Shader ... 249
6.2.1 Der Standard Shader ... 250
6.2.2 Mobile Shader ... 255
6.3 Audio ... 258
6.3.1 AudioSources ... 259
6.3.2 AudioSources via Script steuern ... 262
6.3.3 Temporäre AuidoSource via Script ... 264
6.3.4 Reverb Zones ... 264
6.3.5 Audio Mixer ... 265
6.4 Tastatur und Gamepad-Eingaben lesen ... 269
6.4.1 Der Input-Manager ... 270
6.4.2 Virtuelle Tasten und Achsen ... 270
6.4.3 Konfiguration einer Achse ... 271
6.4.4 Gamepad- und Tastatureingaben erfassen ... 273
6.5 Physik ... 275
6.5.1 Collider ... 275
6.5.2 Rigidbody ... 278
6.5.3 Kräfte und Beschleunigung ... 280
6.5.4 Physics Joints – Verknüpfungen ... 281
6.5.5 Kollisionen und Trigger erkennen ... 285
6.5.6 Raycasts ... 291
6.5.7 Character Controller ... 292
6.6 Partikeleffekte ... 295
6.6.1 Partikelsysteme ... 296
6.6.2 Particle-Effect-Steuerung ... 297
6.6.3 Module des Particle Systems ... 298
6.6.4 Particle Effect Editor ... 305
6.6.5 Particle Systems via Script steuern ... 305
6.6.6 Standard-Partikeleffekte ... 306
6.7 Landschaften mit Terrains erstellen ... 306
6.8 Animation System ... 309
6.8.1 Animation Workflow ... 310
6.8.2 Humanoide Animationen und Avatare ... 312
6.8.3 Modelle & Animationen importieren ... 313
6.8.4 Animation Controller ... 317

6.9 Wegfindung ... 321
6.9.1 Wegfindungssystem einrichten ... 321
6.9.2 Grundlagen ... 323
6.9.3 NavMeshs berechnen ... 325
6.9.4 NavMeshs verbinden ... 326
6.9.5 Dynamische Hindernisse ... 326
6.9.6 NavMeshAgent erstellen ... 327
6.9.7 NavMeshAgent über Script steuern ... 327
6.9.8 Erweiterte Beispiele ... 328
6.10 Savegames – Daten speichern und laden ... 329
6.11 Performance optimieren ... 330
6.11.1 Batching ... 330
6.11.2 Performance im Auge behalten ... 331
6.11.3 VR-Optimierungs-Checkliste und Grenzen ... 335
6.12 User Interface – Quickstart ... 337
6.12.1 User Interface im World Space erstellen ... 337
6.12.2 User Interface in VR bedienen ... 342

7 Die Virtual-Reality-SDKs ... 345
7.1 Unity-interne VR-Unterstützung ... 346
7.1.1 Unitys interne VR-Unterstützung aktivieren ... 346
7.1.2 Interne VR-Unterstützung in der Scripting-API ... 348
7.1.3 InputTracking ... 348
7.1.4 VRDevice ... 349
7.1.5 VRSettings ... 350
7.2 Oculus Integration for Unity (Rift und Gear) ... 352
7.2.1 Download und Import ... 352
7.2.2 Prefabs ... 354
7.2.3 Gamepad-, Headset- und Touch-Eingaben lesen ... 361
7.2.4 Haptisches Feedback ... 363
7.2.5 Oculus Avatar SDK ... 365
7.2.6 Mehr Infos und Hilfe ... 367
7.3 SteamVR (u. a. HTC Vive) ... 368
7.3.1 Download und Import ... 369
7.3.2 Prefabs ... 371
7.3.3 Eingaben der Controller lesen ... 378
7.3.4 Haptisches Feedback ... 379
7.3.5 Interaktionsbeispiel aus The Lab (u. a. Longbow, Teleport) ... 380
7.3.6 Mehr Infos und Hilfe ... 380
7.4 GoogleVR SDK ... 381
7.4.1 Download und Import ... 381
7.4.2 Prefabs ... 382
7.4.3 Eingaben des Daydream Controllers lesen ... 386
7.4.4 Erweiterte Editorfunktionen fürs Testen ... 387

8 Beispielprojekte – Einleitung ... 391

9 Bow and Arrow Castle Defense – GoogleVR mit Controller ... 393

9.1 Der Start ... 394
9.2 Die Kernmechanik – Bogenschießen ... 396
9.2.1 Scene vorbereiten ... 396
9.2.2 Einfaches Scene-Design ... 396
9.2.3 Spieler konfigurieren und Daydream Controller einbinden ... 399
9.2.4 Schusskraft für den Bogen ... 401
9.2.5 Ein Pfeil für den Bogen ... 403
9.2.6 Pfeile nachladen ... 406
9.2.7 Dynamische Schusskraft ... 408
9.2.8 Flugkurve Vorschau – Add-on ... 410
9.2.9 Schaden und zerstörbare Ziele ... 413
9.3 Game- & Level-Design ... 414
9.4 Game Manager ... 416
9.5 Gegner – Orks! ... 421
9.5.1 NavMesh erstellen ... 421
9.5.2 Ork-Animator Controller ... 421
9.5.3 Die künstliche Intelligenz ... 423
9.5.4 Der Ork ... 424
9.5.5 Prefab erstellen und platzieren ... 426
9.6 Abschluss ... 426
9.6.1 Umsetzung dieses Beispiels für andere VR-Brillen ... 427

10 Space Battle – Oculus Rift & SteamVR ... 429

10.1 Der Start ... 430
10.1.1 Scene vorbereiten ... 431
10.2 Die Kernmechanik – ein Raumschiff fliegen ... 434
10.2.1 Spieler-Raumschiff erstellen ... 435
10.2.2 Raumschiff-Steuerung ... 438
10.2.3 Starfield/Weltraumstaub ... 445
10.2.4 Schadenssystem & Plasmakanone ... 448
10.3 Feinde und Verbündete ... 456
10.3.1 KI-Raumschiff ... 456
10.3.2 Die künstliche Intelligenz ... 458
10.4 Abschluss ... 467
10.4.1 Umsetzung dieses Beispiels für andere VR-Brillen ... 468

11 Dungeon Crawler – GearVR ... 469

11.1 Der Start ... 470
11.1.1 Osig und Unity2017.1.0f3-Patch einfügen ... 472
11.2 Einen Dungeon bauen ... 472

11.2.1 Die Bauteile ... 473
11.2.2 Level-Design ... 473
11.2.3 Audio: Environment-Loop ... 476
11.3 Occlusion Culling ... 476
11.4 Kernmechanik – Laufen, Kämpfen, Interagieren ... 478
11.4.1 Spieler erstellen ... 479
11.4.2 Bewegungen entlang des Gitters ... 481
11.4.3 Interaktive und lebendige Objekte ... 484
11.4.4 Die Spielerfunktionen ... 486
11.4.5 Spielereingaben lesen ... 491
11.4.6 Player fertigstellen ... 494
11.5 Gegner: Goblins ... 495
11.5.1 Die künstliche Intelligenz ... 496
11.5.2 Das GameObject und Prefab ... 498
11.5.3 Bevölkerung des Dungeons mit Goblins ... 499
11.6 Pickups: Kristalle ... 500
11.7 Dungeon-Ausgang ... 502
11.8 Interaktive Hindernisse ... 504
11.8.1 Falle: Stacheln ... 504
11.8.2 Rätsel: Schalter und Tore ... 506
11.9 Abschluss ... 509
11.9.1 Umsetzung dieses Beispiels für andere VR-Brillen ... 510

12 Sci-Fi-Stealth-Shooter – Oculus Rift mit Oculus Touch und SteamVR ... 511

12.1 Der Start ... 512
12.2 Trainings-Areal anlegen ... 514
12.3 Player-GameObject erstellen ... 516
12.3.1 Oculus SDK: Player-GameObject erstellen ... 516
12.3.2 SteamVR SDK: Player-GameObject erstellen ... 517
12.3.3 Character Controller ... 518
12.3.4 Locomotion – klassisch und teleportieren ... 521
12.3.5 Eingaben lesen und Methoden aufrufen ... 525
12.3.6 Nach Gegenständen greifen ... 530
12.4 Lebensenergie verwalten ... 540
12.4.1 Lebensenergie des Spielers und Game Over ... 541
12.5 Laserpistole ... 542
12.5.1 Das GameObject vorbereiten ... 542
12.5.2 Die Laserpistolen-Funktion ... 543
12.5.3 Der Laserstrahl ... 545
12.6 Roboter-Gegner ... 547
12.6.1 EnemyAi – wie funktioniert die KI? ... 547
12.6.2 Roboter-GameObject anlegen ... 548

12.6.3 Trigger Collider ignorieren ... 550
12.7 Level erstellen ... 551
12.7.1 Level-Ende ... 552
12.8 Abschluss ... 553

13 VR-Spiele teilen, präsentieren und Feedback erhalten ... 555
13.1 Spieleidee entwickeln ... 555
13.2 VR-Spiele Builds erstellen und teilen ... 556
13.3 Feedback erhalten und Spiel bewerben ... 558
13.3.1 Tipps für das Präsentieren von Virtual Reality ... 559
13.4 Spiel veröffentlichen ... 560
13.5 Kostenlose Assets ... 561

14 Schlusswort ... 563

Index ... 565

Vorwort

Was war der größte Traum deiner Kindheit? Mit bezahlbaren Virtual-Reality-Brillen, deren Fokus auf immersivem Entertainment liegt, ist für mich mein Kindheitstraum wahr geworden. Lange war es nur eine Fantasie aus Sci-Fi-Büchern und eine Technologie, die Anfang der 90er gescheitert war. Doch als ich 2012 das erste Mal von der *Oculus Rift* hörte, schien der Traum plötzlich nicht mehr so fern und ich verbrachte viel Zeit damit, mich immer intensiver mit dem Thema zu beschäftigen. Schnell reichte mir das nicht mehr: Ich wollte andere ebenfalls für das Thema begeistern und helfen, das Thema leichter zugänglich zu machen. Deswegen habe ich 2012 *Bloculus.de* gegründet, wo ich mehrmals die Woche über die aktuellen Neuigkeiten und Spekulationen der Szene schrieb. Heute, wo jede IT-News-Seite über das Thema berichtet, ist mein Blog nicht mehr vonnöten, um Neuigkeiten zu verbreiten. Deshalb begann ich, mehr und mehr aus der Sicht eines VR-Entwicklers zu schreiben und so meinen Lesern eine alternative Perspektive zu bieten.

Ich hatte es mir schon häufig vorgestellt, aber nie hätte ich gedacht, dass ich mal wirklich ein Buch schreibe. Umso erfreuter war ich über den Anruf von Sylvia Hasselbach vom Hanser Verlag, mit der Frage, ob ich Interesse hätte, ein einsteigerfreundliches Buch über Virtual-Reality-Entwicklung zu schreiben.

Es hat Spaß gemacht, all dieses Wissen niederzuschreiben, und der Gedanke, es mit vielen Menschen zu teilen, war sehr motivierend. Aber es war nicht immer leicht, manche Wochen waren sehr anstrengend. Umso dankbarer bin ich für jede Unterstützung, die ich erfahren habe. Danke an alle Freunde, Bekannte und meine Familie, die mich bei diesem Buch, auf die eine oder andere Weise, unterstützt haben – und wenn es nur war, dass sie dafür Verständnis hatten, dass ich mich wenig gemeldet habe, zu spät kam und zu früh wieder ging. Mein ganz besonderer Dank geht an Kathrin Zenses, die mich stets unterstützt hat und mir mit viel Verständnis begegnet ist. Das alles ist nicht selbstverständlich, danke an euch!

Virtual Reality dringt immer mehr in den Massenmarkt und immer häufiger hört man die Worte: *„Jetzt stell dir das mal in Virtual Reality vor …“* Mit diesem Buch möchte ich das Entwickeln für Virtual Reality jedem zugänglich machen, damit auf diesen Satz immer die Aussage folgt: *„Das probiere ich mal aus!“*

Dortmund, September 2017

Daniel Korgel

2 Einführung in Virtual Reality

Bevor es nun richtig losgeht, möchte ich dir zunächst ein wenig Hintergrundwissen geben, damit wir eine gemeinsame Basis haben, auf der wir aufbauen können. Das hat den Vorteil, dass ich in den kommenden Kapiteln nicht immer wieder weit ausholen muss. Besonders wichtig ist natürlich die Erklärung häufig verwendeter Begriffe. Außerdem möchte ich dir ein wenig Hintergrundwissen zur Geschichte von Virtual Reality und zu den einzelnen Headsets, mit denen wir uns in diesem Buch beschäftigen, geben. Zusätzlich werfen wir noch einen Blick über den Tellerrand auf VR-Headsets, mit denen wir uns in diesem Buch nicht gezielt beschäftigen.

2.1 Begriffserklärungen

Ich möchte damit anfangen, dir ein paar Begriffe zu erklären, die dir im Umgang mit Virtual Reality und auch in der Spieleentwicklung immer wieder unterkommen werden. Du kannst in diesem Kapitel nachschlagen, wenn ich ein Wort benutze, unter dem du dir nichts vorstellen kannst.

Wort	Bedeutung
Assets	Das sind alle Medien oder auch „Ressourcen“ in deinem Projekt: Texturen, Sounds, 3D-Modelle, Materialien, Skripte etc.
Field of View, FoV	Beschreibt das Blickfeld in der Brille. Hierbei ist zu beachten, dass es sich meistens nicht auf das horizontale Blickfeld bezieht, sondern auf das diagonale. Zwei Headsets mit dem gleichen Field of View können daher trotzdem ein unterschiedliches Blickfeld in der Brille besitzen. Das eine Headset kann zum Beispiel ein besonders breites und das andere ein besonders hohes Blickfeld haben.
Frames	Ein Frame ist ein einzelnes auf dem Monitor oder Headset dargestelltes Bild, das von der Grafikkarte berechnet wurde.
Frames per Second, FPS	„Bilder pro Sekunde“, beschreibt die Anzahl an Bildern pro Sekunde, welche von der Grafikkarte an das jeweilige Ausgabemedium gesendet wird.

Wort	Bedeutung
Head-Mounted-Display	Wörtlich übersetzt ein „am Kopf befestigtes Display“. Diese Kategorie umfasst Virtual-Reality-Brillen, schließt aber auch Augmented-Reality-Brillen wie Microsoft *Hololens* mit ein.
Head-Tracking	Bezeichnet die Positions- und Rotationserkennung des Kopfes im Raum.
Immersion	Beschreibt die Eintauchtiefe in die virtuelle Welt.
Locomotion	Beschreibt die Art und Weise der Fortbewegung, zum Beispiel normales „Laufen“ oder „Teleportieren“.
Low Persistence	Eine Technologie, die es ermöglicht, auch in der Bewegung scharfe Bilder in einem VR-Headset anzuzeigen. Hierbei wird das Bild nicht dauerhaft auf dem Display angezeigt, sondern blitzt z. B. 60- oder 90-mal pro Sekunde kurz auf, wenn der Frame gerade frisch vom Grafikprozessor berechnet wurde.
Motion-Controller	Das sind Eingabegeräte, die der Anwender für gewöhnlich in der Hand hält und deren Position und Rotation erkannt werden können.
Positional-Tracking	Bezieht sich im entsprechenden Kontext nur auf die Positionserkennung und nicht auf die Rotationserkennung.
Presence	Beschreibt den Zustand, in welchem die Immersion so hoch ist, dass man vergisst, dass die dargestellte Welt nicht real ist. Häufig verwendet man den Begriff „Micro Presence“, um Situationen zu beschreiben, wo dieser Zustand für einen sehr kurzen Moment erreicht wird.
Recentern	Das Zurücksetzen der virtuellen Position und Rotation in die Ausgangsstellung. Hierdurch wird bestimmt, welche reale Blickrichtung mit welcher virtuellen Blickrichtung übereinstimmt.
SDK	Steht für *Software Development Kit*. Das sind Entwicklungswerkzeuge, die meistens von dem Hersteller einer Hardware oder Software für externe Entwickler (dich) zur Verfügung gestellt werden; zum Beispiel das „Oculus SDK“ zum Entwickeln für Oculus-Brillen.
Thumbstick	Eine alternative Bezeichnung für den Analog-Stick, den man auf den meisten aktuellen Gamepads findet.
Tracking	Beschreibt ganz allgemein Positions- und Rotationserkennung von Gegenständen im 3D-Raum. Der Begriff kann sich also z. B. sowohl auf die Headsets als auch auf die Controller beziehen.
Virtual-Reality-Headset	Ein anderes Wort für Virtual-Reality-Brille
VR	Kurz für Virtual Reality

2.2 Augmented Reality, Mixed Reality und Virtual Reality

Wenn du gerade erst anfängst, dich mit dem Thema zu beschäftigen, stößt du im Zusammenhang mit Virtual Reality wahrscheinlich auch häufig auf den Begriff *„Augmented Reality“*. Augmented Reality ist eine vollkommen eigenständige Technologie mit individuellen Einsatzmöglichkeiten und hat mit Virtual Reality nicht direkt zu tun. Zusätzlich gibt es noch den Begriff *„Mixed Reality“*, welcher mehr eine Oberkategorie als eine bestimmte Technologie beschreibt.

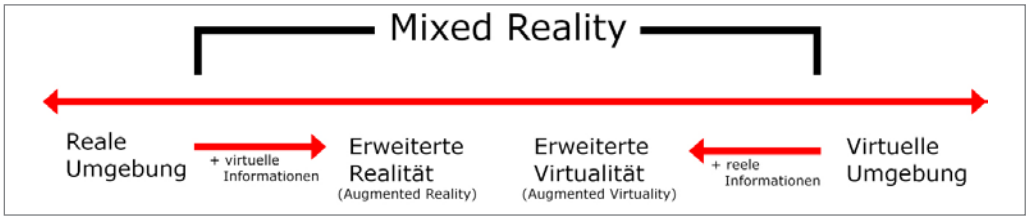

Bild 2.1 Mixed Reality

Bild 2.1 zeigt dir eine Grafik zur groben Orientierung. Bei den einzelnen Begriffen ist nicht definiert, ob man sich die jeweilige Realität über eine Brille (Head Mounted Display) oder zum Beispiel auf einem Smartphone-Bildschirm ansieht. Innerhalb der einzelnen Kategorien werden für die jeweiligen Darstellungsformen dann weitere Unterscheidungen vorgenommen.

Augmented Reality beschreibt eine Technologie, welche die echte Welt mit virtuellen Informationen erweitert. Das bedeutet zum Beispiel, dass du dir eine reale Speisekarte in einem Restaurant ansiehst, und wenn du mit deinem Finger auf eines der Menüs zeigst, erscheint eine dreidimensionale Darstellung des Menüs vor dir. Diese Darstellung kann entweder über AR-Brille, ein Smartphone, welches das Kamerabild augmentiert, oder auch über ein Hologramm erzeugt werden.

Virtual Reality beschreibt eine Technologie, bei der die Welt vollständig virtuell ist und mit der auf eine realistische Art und Weise interagiert werden kann. Du wärst also in einem virtuellen Restaurant und bestellst von einer virtuellen Speisekarte ein virtuelles Essen, ohne in der Realität etwas zu erhalten. Auch Virtual Reality muss nicht zwingend mit einer VR-Brille angesehen werden: Sogenannte *„CAVEs“* sind spezielle Räume, bei denen die Wände (aber nicht zwingend alle Wände) durch mehrere Beamer angestrahlt werden und den Anwender auf diese Weise in die virtuelle Welt versetzen.

Bei **Augmented Virtuality** wird die Darstellung einer virtuellen Welt mit Elementen aus der echten Welt überlagert. Du sitzt also zum Beispiel in einem virtuellen Restaurant, siehst und isst aber ein reales Essen, das auch wirklich vor dir steht. Von der Realität siehst du allerdings nur das Essen und deine Hände, nicht aber den Raum, in dem du dich tatsächlich befindest. Falls du von diesem Begriff noch nie etwas gehört hast, liegt das höchstwahrscheinlich daran, dass Augmented Virtuality häufig auch einfach als „Virtual Reality“ bezeichnet wird.

Bei den Begriffen geht es also immer darum, was das Ausgangsmedium ist: Bei Augmented Reality bist du von der realen Welt umgeben, die teilweise mit virtuellen Elementen angereichert wird. Bei Virtual Reality bist du von der virtuellen Welt umgeben und es werden höchstens einzelne Gegenstände aus der realen Welt dargestellt. Die Grenzen zwischen den einzelnen Arten sind jedoch fließend und rein technisch kann man noch viele weitere Unterscheidungen vornehmen. Man findet also durchaus Fälle, die nicht so einfach auf den Strahl von Bild 2.1 einsortiert werden können.

2.3 Virtual Reality ist tot – lang lebe Virtual Reality

In den letzten Jahren hat Virtual Reality begonnen, richtig durchzustarten, doch Virtual-Reality-Hardware existiert schon viel länger. Es gab sogar schon eine richtige Welle an Virtual-Reality-Headsets. Vielleicht kennst du Nintendos berühmt-berüchtigten Virtual Boy? Dieser war nur die Spitze des Eisberges an Virtual-Reality-Headsets in den späten 80ern und 90ern. In diesem Kapitel möchte ich dir gerne ein paar Informationen zur ersten großen Virtual-Reality-Welle mitgeben.

Der Begriff „Virtual Reality“ wurde 1982 durch Damien Brodericks Roman *The Judas Mandala* geprägt, das Konzept einer Virtual-Reality-Brille existierte, wurde jedoch schon 1934 erstmals beschrieben. Noch bevor es die ersten Computer gab, beschrieb Stanley G. Weinbaum in seiner Kurzgeschichte *Pygmalion's Spectacles* ein auf Brillen basierendes System mit holografischen Aufnahmen von fiktiven Ereignissen.

Mit dem Aufkommen von Arcade-Hallen, Heimcomputern und den ersten flachen LCD-Bildschirmen begann auch die Forschung an Virtual-Reality-Headsets für Heimanwender. 1980 forschte Atari zum Beispiel an einem Virtual-Reality-Arcade-Automaten, musste die Forschung 1983 allerdings wegen des Video-Game-Crashs[1] einstellen. 1985 entwickelte die NASA eine Virtual-Reality-Workstation namens *VIEW*. Die *Virtual Environment Workstation* umfasste sogar schon einen Datenhandschuh, mit dem virtuelle Objekte gegriffen werden konnten.

Mehr Details zur NASA Virtual-Reality-Workstation findest du hier:

https://www.nasa.gov/ames/spinoff/new_continent_of_ideas/

[1] Zwischen 1983 und 1985 brach der amerikanische Videospielmarkt zusammen und viele amerikanische Firmen gingen bankrott oder wendeten sich von der Videospielindustrie ab. Als Symbol des Crashs wird häufig der Niedergang des zuvor langjährigen Marktführers Atari verwendet. Erst das 1985 in den USA erscheinende Nintendo Entertainment System belebte den amerikanischen Videospielmarkt wieder. In den folgenden Jahren dominierten jedoch japanische Hersteller wie Nintendo, Sega und Sony den weltweiten Videospielmarkt.

In den 90ern kamen dann die ersten Virtual-Reality Headsets auf den Markt, welche nicht nur für den professionellen Einsatz bestimmt waren. Für nur 1800 DM hättest du dir zum Beispiel das *Forte VFX1* mit einer Auflösung von 263 x 230 Pixel pro Auge, 256 Farben, einem Field of View von 35.5° und einem Gewicht von 1100 g kaufen können. Für ein High-End-VR-Erlebnis hättest du jedoch in die Arcade-Halle gehen müssen: Hier hatten die Headsets von *Virtuality* eine Auflösung von 276 x 372 Pixel pro Auge mit einem Fliegengewicht von 645 g. Zum Vergleich: Die aktuelle Version der *Oculus Rift* bietet dir eine Auflösung 1080 x 1200 Pixel pro Auge, ein 110° großes Field of View und ein gut ausbalanciertes Gewicht von 470 g. Trotzdem wird bei aktuellen Headsets mit Tricks gearbeitet, da die Auflösung noch nicht perfekt ist.

Bild 2.2 Das Forte VFX1 Headset. (Quelle: Wikimedia Commons, Ajerimez)

Die Headsets waren nicht nur niedrig aufgelöst, sie litten auch unter starkem Motion-Blur. Das bedeutet, sobald du dich umgesehen hast, wurden aus dem angezeigten Bild nur noch matschige Farbflächen. Aber auch die Sensoren zur Bewegungserkennung waren bei Weitem noch nicht so genau, wie sie heute sind. Deine Bewegungen wurden daher häufig ungenau und nur verzögert erkannt.

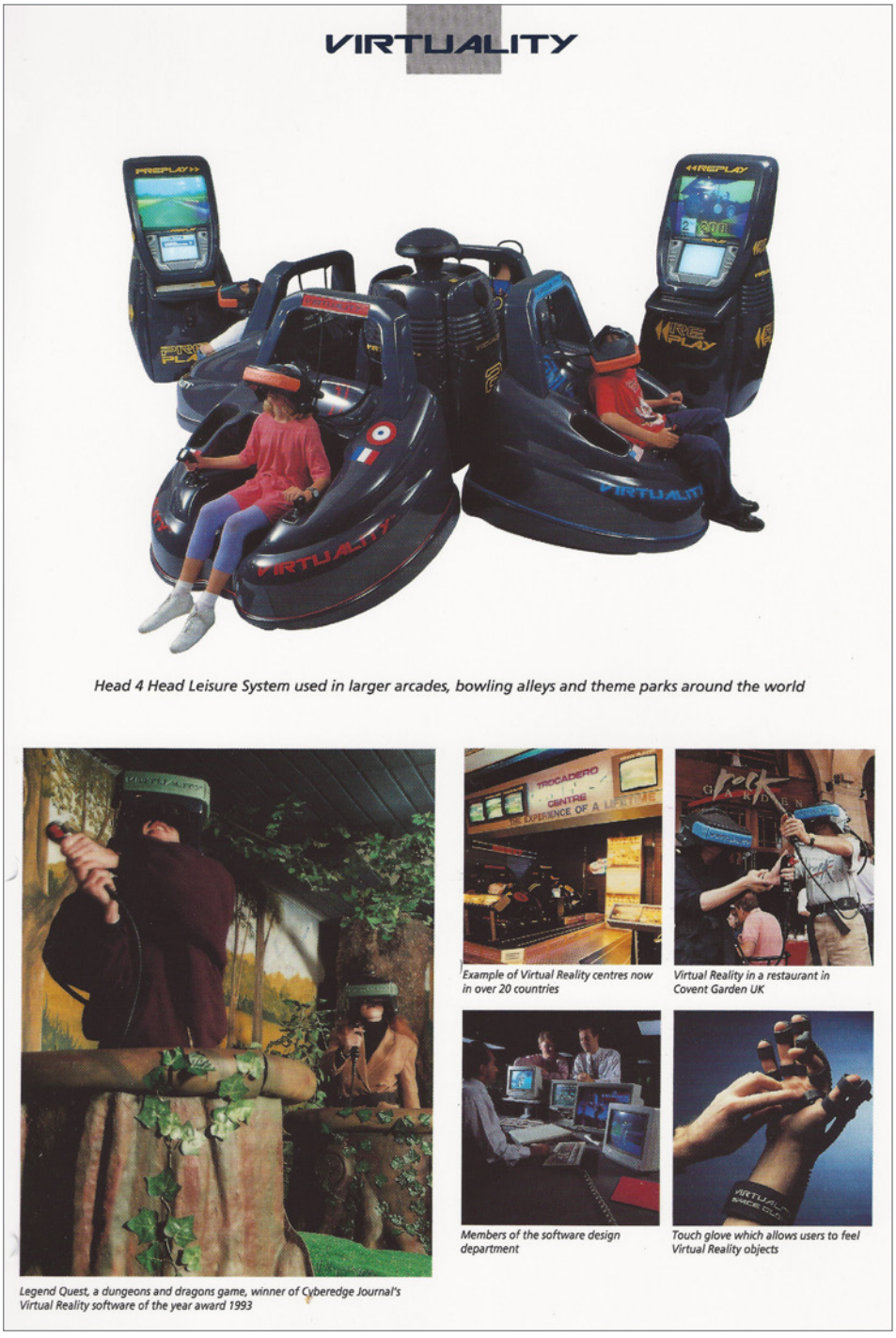

Bild 2.3 Eine Seite eines Prospekts zur Bewerbung von Virtual-Reality-Systemen der Firma Virtuality (ca. 1991). (Quelle: Dr. Waldern/Virtuality Group)

Doch sie waren nicht die einzigen Hersteller, die an VR-Brillen forschten: Die Videospielfirmen *Sega* und *Atari* hatten für ihre Konsolen *Sega Mega Drive* und *Atari Jaguar* jeweils ein Virtual-Reality-Headset angekündigt. Der große Computer-Hersteller *IBM* hatte ein *Elysium* genanntes Headset auf den Markt gebracht und auch einige weitere Firmen versuchten ein Stückchen von dem Kuchen abzubekommen.

Die Hardware war allerdings noch nicht so weit. Der hohe Preis und das hohe Gewicht, zusammen mit der schlechten Bild- und Tracking-Qualität waren nur ein paar der Sargnägel für Virtual Reality in den 90ern. Noch bevor Sega und Atari ihre Headsets herausbringen konnten, platzte die Blase und die Idee von Virtual Reality verschwand vom Massenmarkt. Es ist also kein Wunder, dass sich bis 2012 kein Consumer-Electronics-Hersteller mehr an Virtual Reality herangetraut hat, nachdem einige Firmen hier viel Geld verloren hatten. Es brauchte erst einen 19-jährigen Palmer Luckey, um den gebrandmarkten Firmen zu zeigen, dass nun die Zeit von Virtual Reality gekommen ist.

Mehr zum Scheitern von Virtual Reality in den 80ern und 90ern

Wenn dich das Thema interessiert, empfehle ich dir diesen Artikel auf TheVerge.com. Er enthält viele interessante Informationen und Bilder aus der Zeit:

The Rise and Fall and Rise of Virtual Reality:
http://www.theverge.com/a/virtual-reality

2.4 Unterstützte Virtual-Reality-Brillen

In diesem Buch werden wir uns mit verschiedenen Virtual Realitys beschäftigen und diese werde ich dir in diesem Kapitel zunächst einmal vorstellen. Deine eigene Brille kennst du wahrscheinlich schon, aber vielleicht kann ich dir ja noch ein paar Details aus der Entwickler-Perspektive verraten, die du noch nicht kennst. Als VR-Entwickler ist es generell wichtig, einen guten Überblick über den gesamten VR-Markt zu haben und die Stärken und Schwächen der einzelnen Headsets zu kennen.

2.4.1 Oculus Rift

Bild 2.4 Die erste Endbenutzer-Version der Oculus Rift.(Quelle: Oculus)

Die *Oculus Rift* war der Vorreiter der aktuellen Virtual-Reality-Generation. Bevor die aktuelle „Consumer Version" erschien, gab es bereits zwei für Entwickler vorgesehene „Developer Kits" und einige Prototypen. Um die einzelnen Headsets voneinander zu unterscheiden, wirst du im Internet häufig ein entsprechendes Kürzel hinter dem Headset-Namen finden. 2012 kam das erste Entwickler Kit DK1 raus und 2014 erschien eine modernisierte zweite Version des Entwicklerkits (DK2). Im Jahr 2016 erschien dann die heiß ersehnte Endbenutzerversion mit dem Kürzel CV.

Das Positional-Tracking funktioniert bei der Oculus Rift über eine externe Kamera, welche du auf deinen Schreibtisch stellst. Die Kamera muss stets Sichtkontakt mit dem Headset haben, ansonsten entstehen unangenehme Ruckler. Die *Rift* hat integrierte Kopfhörer und ein integriertes Mikrofon, weshalb du keine zusätzlichen Kopfhörer benötigst.

Bild 2.5 Die „Oculus Touch" genannten Controller für die Rift. (Quelle: Oculus)

Im Dezember 2016 veröffentlichte Oculus die *Oculus Touch* genannten Motion-Controller für ihre VR-Brille. Auch zu den *Oculus Touch*-Controllern darf der Sichtkontakt nicht abbrechen. Bei den Motion-Controllern ist die Wahrscheinlichkeit, dass du den Sichtkontakt mit deinem Körper unterbrichst, jedoch deutlich höher als bei einem Headset, das du auf dem

Kopf trägst. Aus diesem Grund werden *Oculus Touch*-Controller mit einem weiteren Sensor ausgeliefert. Optional kann auch ein dritter Sensor erworben werden, welchen du in einer gegenüberliegenden Ecke deines Raumes aufstellen kannst.

Die *Consumer Version* hat pro Auge eine Auflösung von 1080 x 1200 Pixel bei 90 Hz und wiegt 470 g. Das Field of View beträgt 110°. Oculus-kompatible Anwendungen bezieht man in erster Linie aus dem *Oculus Store*. Oculus-Rift Anwendungen können aber auch über andere Plattformen wie *Steam* oder einfach als „.exe“ verteilt werden.

2.4.2 HTC Vive

Bild 2.6 Die HTC Vive: Im Hintergrund sind die Lighthouses zu sehen, das Headset in der Mitte und die Motion-Controller im Vordergrund. (Quelle: HTC)

Die *HTC Vive* ist aus einer Kooperation von *HTC* und *Valve*, der Firma hinter *Steam* und Spielen wie *Half-Life*, entstanden. Der Bildschirm und die Linsen ähneln denen der Oculus Rift. Doch die *Vive* unterscheidet sich dennoch in ein paar entscheidenden Punkten von ihrer Konkurrenz. Anders als die Rift nutzt die Vive ein auf Laser basierendes Tracking-System. Mit zwei sogenannten *Lighthouses* lässt sich eine Fläche von 4 x 5 Metern lückenlos abdecken. Mit der Vive kann man also, wenn man so viel Platz zur Verfügung hat, in Virtual Reality herumlaufen. Die Motion-Controller gehören bei der Vive zum Lieferumfang dazu und müssen nicht extra gekauft werden. Deswegen werden die Controller auch von nahezu jedem Vive-Spiel verwendet. Die meisten Spiele für die Vive sind darauf ausgelegt, im Stehen gespielt zu werden, und benötigen meist eine Spielfläche von mindestens 2 x 2 Metern.

Die HTC Vive ist das erste *SteamVR*-Headset, das erschienen ist. Es sollen jedoch noch weitere SteamVR-Headsets von anderen Herstellern folgen.

Die HTC Vive hat pro Auge eine Auflösung von 1080 x 1200 Pixel bei 90 Hz und wiegt 555 Gramm. Das Field of View beträgt 110°. HTC Vive-kompatible Anwendungen können über die Distributionsplattform Steam und dem auf die Vive spezialisierten Viveport bezogen werden. Es gibt zwei offizielle Stores, da Steam in Ländern wie China nicht sehr weit verbreitet ist. Anwendungen werden aber auch direkt als „.exe“ geteilt.

2.4.3 Samsung GearVR

Bild 2.7 Die Samsung GearVR wird über den USB-Anschluss mit einem kompatiblen Smartphone verbunden. (Quelle: Oculus)

Die *GearVR* von Samsung wird im Gegensatz zu den zuvor vorgestellten Headsets nicht mit deinem Computer, sondern mit deinem Smartphone verbunden. Dies hat den Vorteil, dass du sie überall verwenden kannst und nicht an Kabel gebunden bist. Die GearVR unterstützt ausschließlich aktuelle Samsung-Smartphones (ab Galaxy S6), welche du in die Brille steckst. Die VR-Brille verwendet dann die Rechenpower und das Display deines Smartphones, um die virtuelle Welt darzustellen. Bei der Entwicklung der GearVR hat Samsung mit *Oculus* zusammengearbeitet, weshalb die VR-Brille eine mobile Version des *Oculus Stores* verwendet. Oculus Rift-Spiele sind jedoch nicht kompatibel.

Die Auflösung der GearVR hängt natürlich von dem Smartphone ab, das du verwendest. Die unterstützten Geräte haben derzeit[2] eine Auflösung von 2560 x 1440 Pixel, das sind also 1280 x 1440 Pixel pro Auge. Das Field of View beträgt 101°.[3]

Die GearVR ist ein geschlossenes System. Das bedeutet, offiziell kannst du Spiele und Anwendungen nur über den mobilen Oculus Store herunterladen. Anders als bei normalen Android-Apps ist es nicht möglich, die Installationsdateien auf einem anderen Weg herunterzuladen und zu installieren. Samsung und Oculus haben, ähnlich wie man es von Apple kennt, ein System entwickelt, das das Teilen von Apps außerhalb des Stores verhindert: Den Apps musst du, vor dem Erstellen der Installationsdateien, die Device-IDs (Gerätekennungen) der Smartphones mitteilen, auf denen die App laufen soll. Nur wenn die App über den Oculus Store heruntergeladen wird, muss sie nicht explizit für das jeweilige Gerät freigeschaltet werden.

Da nicht jede App zu den Richtlinien von Oculus und Samsung passt und deswegen unter Umständen nicht in den Store kommt, haben Tüftler eine alternative Plattform namens *SideloadVR* geschaffen. Möchtest du eine App von dieser Plattform downloaden, wird die Installationsdatei der App auf deren Server extra für dein Smartphone neu erstellt. Du kannst auf *SideloadVR.com* jedoch deine Apps nicht verkaufen, sondern sie nur kostenlos anbieten.

---

[2] Stand 2017

[3] Der Wert bezieht sich auf die schwarze 2017er-Version, der weiße Vorgänger hatte ein Field of View von 96°.

2.4.4 Google Cardboard (nur teilweise unterstützt)

Um Virtual Reality in den Massenmarkt zu helfen, entschied sich *Google* im Jahr 2014, eine *Open-Source-VR-Brille* zu veröffentlichen. Diese Brille ist so günstig in der Produktion, dass sie für weniger als 10 Euro verkauft werden kann, und sie besteht, bis auf die Linsen, ausschließlich aus Pappe. Daher stammt auch ihr passender Name: *Cardboard*.

Ein teurer Computer oder das allerneuste Smartphone wird nicht benötigt: Als Bildschirm und Recheneinheit kannst du jedes aktuelle Smartphone verwenden, das einen *Gyroskop* besitzt und mit mindestens Android 4.4 *KitKat* oder einer aktuellen iOS-Version läuft. Das Handy legst du einfach mit einer kompatiblen App in die Papphalterung ein und los geht's.

Bild 2.8 Die Google Cardboard besteht, bis auf die Linsen, ausschließlich aus Pappe. (Quelle: Google Inc.)

Natürlich ist das VR-Erlebnis, das man mit einer solchen Brille hat, nicht mit dem einer mehrere Hundert Euro teuren Lösung vergleichbar, aber der Plan ging auf: Durch den günstigen Preis und die Einfachheit der Cardboard gelang es Google, die sonst teure Technologie jedem zugänglich zu machen. Dank des Open-Source-Designs kann zum Beispiel jede Firma ihre eigenen Varianten herstellen und so für sich werben. Cardboards werden Zeitungen beigelegt oder auf Veranstaltungen an Besucher verschenkt. Dank öffentlich zugänglicher Entwicklerwerkzeuge ist es zudem jedem möglich, seine eigenen Anwendungen für die Brille zu entwickeln.

Cardboard ist eine gute VR-Brille für den ersten Kontakt mit VR. Sobald du dich aber intensiver damit auseinandersetzen möchtest, solltest du zu einer der anderen Brillen wechseln. Grundsätzlich kannst du diesem Buch mit einer Cardboard folgen, es gibt aufgrund der großen Einschränkung jedoch kein Beispielprojekt für diese Brille.

Cardboard-Apps können, wie normale Android-Apps, sowohl über den offiziellen *PlayStore* als auch über alternative Stores und direkt als Installationsdatei (*.apk) geteilt werden. Grundsätzlich ist es möglich, mit Cardboard-Apps einen großen Markt zu erreichen, jedoch sind die meisten Apps nur kurzweilige Erfahrungen und Spielereien. Für längere Spiele ist die Cardboard-Brille ohnehin nicht ausgelegt: Vielen Varianten fehlt zum Beispiel ein Kopfband und ein Großteil der unterstützten Smartphones sind relativ schwach, sodass komplexe Spiele kaum möglich sind.

2.4.5 Google Daydream

2016 kündigte Google den nächsten Schritt an: *Daydream*. Virtual Reality wurde ab Android 7.0 ein fester Bestandteil des Betriebssystems und kompatible Smartphones erhalten Zugang zu der Daydream-Plattform. Passend dazu stellte Google ein Daydream-Headset mit dem Namen *Daydream View* vor. Im Gegensatz zur *Cardboard*-Brille setzt Google bei Daydream auf Qualität: Große Linsen, Polsterungen und vorgegebene Mindestanforderungen für die Smartphones sollen ein hochqualitatives Virtual-Reality-Erlebnis liefern. Sie bleiben sich dennoch treu: Anders als zum Beispiel bei Samsung werden nicht nur die eigenen Smartphones unterstützt, sondern jedes Smartphone, das den aufgestellten Spezifikationen entspricht und dessen Hersteller dies möchte. Das von Google veröffentlichte Headset dient, wie die offiziellen Google-Smartphones, als Referenzdesign, das anderen Handy-Herstellern als Vorbild dienen soll. Das Headset kann über Googles Onlineshop gekauft werden.

Zur Brille selbst gehört auch ein einheitliches Eingabegerät: Der Daydream-Controller passt in eine Hand und bietet zwei Buttons, ein rundes Touchpad sowie eine Lautstärke-Steuerung. Das Besondere ist jedoch nicht sichtbar: Über interne Sensoren kann der Controller seine Rotation und Beschleunigung erkennen und erlaubt so eine intuitive Steuerung von Virtual-Reality-Spielen.

Bild 2.9 Das Referenzdesign von Google: links das „View“ genannte Headset, rechts der Daydream-Controller. (Quelle: Google Inc.)

Der offizielle Weg, Daydream-Apps zu installieren und zu teilen, ist über eine spezielle VR-Sektion in Googles PlayStore. Um die Apps bequem installieren zu können, ohne dass man die VR-Brille abnehmen muss, hat Google eine VR-Version des PlayStores in die Daydream-Plattform integriert.

Grundsätzlich können Daydream-Apps aber auch über alternative Stores und direkt als Installationsdatei (*.apk) geteilt und installiert werden. Nach der Installation tauchen diese Apps dann ebenfalls in der Daydream-VR-Oberfläche auf, sodass man sie wie Apps aus dem PlayStore bequem von innerhalb der Brille starten kann.

2.5 Andere VR-Brillen

Wie schon erwähnt, gibt es neben den unterstützten VR-Brillen auch noch andere Headsets, die ich dir natürlich nicht vorenthalten möchte. Von manchen hast du vielleicht schon in den Medien gehört. Einige dieser Brillen kannst du über das SteamVR-SDK ansprechen und viele Tipps aus diesem Buch werden dir auch bei der Entwicklung für diese Headsets helfen, jedoch wirst du keine detaillierten Anleitungen für die Besonderheiten dieser Brillen in diesem Buch finden.

2.5.1 Microsoft Mixed Reality

Neben der *Augmented-Reality*-Brille *Hololens* arbeitet Microsoft auch an einer Virtual- bzw. Mixed-Reality-Brille, welche in Zusammenarbeit mit verschiedenen Hardwarepartnern entsteht. Der Virtual-Reality-Modus soll ein fester Bestandteil des Windows 10-Betriebssystems werden, sodass kompatible Brillen ohne zusätzliche Software direkt mit Windows funktionieren. Microsoft selbst entwickelt jedoch nur die Plattform und hat sich für die Herstellung kompatibler Hardware Acer, Asus, Dell, HP und Lenovo ins Boot geholt. Preislich werden die Brillen unter den empfohlenen Preisen der Konkurrenz von Oculus und HTC liegen.

Bild 2.10 Das Acer Mixed Reality Bundle mit den Motion-Controllern (links) und dem Acer Headset rechts. (Quelle: Microsoft)

Microsofts VR-Brillen unterscheiden sich in einem Punkt sehr deutlich von der Konkurrenz: Hier werden keine externen Kameras oder Sensoren für das Positional-Tracking benötigt. An dem Headset befinden sich zwei nach vorn gerichtete Kameras, welche ähnlich wie bei der *Hololens* die Umgebung erkennen und so die Position im Raum bestimmen können.

Das System wird auch zwei Motion-Controller erhalten, die ebenfalls nicht über externe Sensoren, sondern über die Kameras an dem Headset getracked werden.

Die Brillen dieser ersten Generation haben eine Auflösung von 1440 x 1440 Pixel pro Auge und ein *horizontales Field of View* von 95°.[4]

Spiele für Microsofts Mixed-Reality-System werden primär in Microsofts eigenem Store verkauft. Spiele für diese Brillen werden aber ebenfalls im Steam-Store angeboten werden.

Leider konnte ich nicht rechtzeitig ein Entwickler-Kit erhalten, weshalb die Brillen in diesem Buch nicht explizit beschrieben werden. Sobald sie aber offiziell von Unity unterstützt werden, kannst du die meisten Inhalte dieses Buches auch mit den Microsoft-Brillen verwenden. Lediglich die auf das SDK bezogenen Beispiele musst du selbstständig anpassen. Die Microsoft Mixed Reality-Brille soll zukünftig auch durch das SteamVR-SDK unterstützt werden, sodass die Verwendung des Microsoft SDKs nicht zwingend notwendig ist und SteamVR-Spiele mehr oder weniger automatisch auch auf dieser Brille laufen werden.

2.5.2 PlayStation VR

Bild 2.11 Die PlayStation VR-Familie mit PlayStation Camera und den „Move“ genannten Motion-Controllern. (Quelle: Sony Computer Entertainment Inc.)

Die von Sony entwickelte *PlayStation VR* ist seit ihrem Erscheinen die meist verbreitetste Virtual-Reality-Brille. Für die PSVR benötigst du keinen leistungsstarken Computer, sondern lediglich eine PlayStation 4. Wobei eine PlayStation 4 Pro zu empfehlen wäre, da diese mit einer besseren Darstellung punkten kann. Die optische Qualität der Spiele ist allerdings trotzdem schlechter oder muss mehr Kompromisse eingehen als bei der Konkurrenz, die einen Gaming-PC benötigt. Dafür gibt es für PlayStation VR viele hochqualitative Spiele von

[4] Dieser Wert kann nicht mit den Field-of-View-Werten der anderen Headsets verglichen werden, da er nur das horizontale Field of View beschreibt. Ein vergleichbarer Wert liegt nicht vor.

bekannten Studios, wie zum Beispiel *Resident Evil 7*. Hochqualitative Spiele mit langer Spieldauer sind etwas, das bei manchen Konkurrenten noch Mangelware ist.[5]

PlayStation VR nutzt neben internen Sensoren eine neue Version der *PlayStation Camera* und die von *PlayStation Move* bekannten bunten Lichter, um die Position der Brille zu erkennen. Passend dazu verwendet Sony für Spiele mit Hand-Tracking auch die bekannten Move-Controller. Spiele ohne Hand-Tracking spielst du mit dem normalen PlayStation 4 Gamepad.

Die Sony-Brille hat pro Auge eine Auflösung von 960 × 1080 Pixel bei 120 Hz und wiegt 610 g. Das Field of View beträgt 100°. Vollpreis-Spiele können regulär im Geschäft gekauft werden, Indie-Spiele werden allerdings ausschließlich über Sonys *PlayStation Network* verkauft. Es ist möglich, mit Unity für PlayStation VR zu entwickeln, allerdings musst du hierfür ein offizieller Playstation-Entwickler sein, der von Sony freigegeben wurde. Davon abgesehen kannst du, und das auch nicht nur zum Testen, keine eigenen Anwendungen auf der PlayStation 4 starten. Wenn dein Spiel läuft, liegt es aber immer noch an Sony, ob du dein Spiel auf ihrer Plattform verkaufen darfst. Es empfiehlt sich also, eng mit Sony zusammenzuarbeiten, damit nach Fertigstellung des Spiels keine bösen Überraschungen auf dich zukommen.

Meiner Erfahrung nach ist der für Indie-Entwickler gängigste Weg, zunächst ein erfolgreiches Spiel für die *Oculus Rift* oder *SteamVR* zu entwickeln. Je nach Erfolg und Spiel ist es dann nicht unwahrscheinlich, als *PlayStation VR*-Entwickler von Sony freigegeben zu werden.

2.5.3 Razer OSVR

Bild 2.12 Das OSVR Hacker Development Kit 2 von Razer
(Quelle: Razer Inc.)

Die „Open Source Virtual Reality"-VR-Brille ist eine Virtual-Reality-Brille, die in Zusammenarbeit von einigen führenden IT-Firmen entstanden ist. Derzeit werden *OSVR*-Brillen ausschließlich von der Firma *Razer* hergestellt und verkauft. Die Brillen laufen unter dem

[5] Stand 2017

Namen *HDK*, was für *Hacker Development Kit* steht. Wie der Name es schon sagt, richten sich die Brillen derzeit primär an Softwareentwickler und nicht an Leute, die nur spielen wollen. Das Kit kann allerdings trotzdem von jedem über die Razer-Webseite gekauft werden.

Das in Bild 2.12 dargestellte *HDK2* ist bereits die dritte kaufbare Brille der Reihe nach dem *HDK 1.3* und *HDK 1.4*. Eine besondere Eigenschaft ist, dass die Brille aus einzelnen, austauschbaren Komponenten zusammengebaut ist. Dies erlaubt Razer, „Upgrade"-Kits zu verkaufen, die aus einem beliebigen HDK die neuste Version werden lassen. So kann man zum Beispiel durch Tausch des Displays aus dem HDK1.4 ein HDK2 machen. Die OSVR-Brillen sind als günstigere und upgrade-bare Alternative zur Oculus Rift und HTC Vive entworfen worden. Mit OSVR-Headsets kann *theoretisch* jedes Spiel in der *SteamVR*-Bibliothek gespielt werden. Jedoch fehlen der VR-Brille die Motion Controller, die für die vielen *SteamVR-Spiele* benötigt werden.

Das OSVR HDK2 hat eine Auflösung von 1080 x 1200 Pixel pro Auge und 90 Hz. Das Field of View beträgt, wie bei der Konkurrenz, 110°. Besonders wird die OSVR-Brille auch durch die individuell einstellbaren Linsen, welche bis zu +–4.5 Dioptrien ausgleichen können.

Mehr Infos zu Razers OSVR-Brillen:

Die offizielle Webseite: *http://www.osvr.org/hdk2.html*

Das HDK2 im offiziellen Razer-Online-Store:
http://www.razerzone.com/de-de/store/hdk2

2.6 Spiele, Anwendungen, Demos und Erfahrungen

Wenn du dich ein wenig mit VR-Spielen beschäftigst, wirst du schnell feststellen, dass Anwendungen von den Spielern in verschiedene Kategorien gepackt werden. Die gängigen wären *Spiel*, *Demo*, *Anwendung* und *Erfahrung*. Dir eine einfache Unterscheidungshilfe zwischen diesen vier Kategorien zu geben, ist nicht ganz einfach, da die Grenzen flüssig sind und die Begriffe „Demo" und „Erfahrung" häufig auch abwertend für „Spiele" verwendet werden. Nachfolgend findest du trotzdem eine Hilfe, was sich im Kontext „Virtual Reality" hinter den Begriffen verbirgt.

- **Demos**
  Demos sind, so wie außerhalb von Virtual Reality auch, „Demonstrationen" einer bestimmten Technik, einer Mechanik oder eines Spiels. Eine Demo zeigt in der Regel einen kurzen und eingeschränkten Ausschnitt eines *Spiels*, einer *Erfahrung* oder einer *Anwendung*. Das Ziel einer Demo kann sein, den Tester zum Kauf der Vollversion anzuregen oder auch eine bestimmte Spielmechanik in der Öffentlichkeit vorzustellen und zu testen. Demos werden häufig auch als „Alpha"- oder „Beta"-Versionen bezeichnet, da sie noch nicht vollständig

sind und fehlerhaft sein können. Eine Demo impliziert, dass es ein größeres oder umfangreicheres Produkt zu der gezeigten Software gibt oder geben wird. Eine Demo ist in der Regel keine vollwertige und in sich geschlossene Anwendung, Spiel oder Erfahrung.

- **Erfahrungen/Experiences**
  Als „Erfahrung“ oder „Experience“ bezeichnet man meist kurze Spiele, die dem Spieler eine, wie der Name es schon sagt, Erfahrung anbieten und erleben lassen sollen. Es geht also darum, Dinge zu erleben. Bei einer *T-Rex-Experience* könnte man zum Beispiel erleben, wie es sich anfühlt, wenn man plötzlich einem ausgewachsenen T-Rex begegnet. Bei einer *Strand-Erfahrung* könnte man sich in Virtual Reality an einen Strand legen und entspannen. In der Regel sind diese Erfahrungen größtenteils passiv und haben sehr wenige bis keine Spielmechaniken, das muss aber nicht so sein. Bei einer *Bankräuber-Erfahrung* könnte z. B. auch geschossen werden, man erlebt aber keine umfassende Geschichte, sondern nur einen kurzen Ausschnitt. Es geht darum, eine bestimmte Situation zu erleben und sich dabei beeindrucken zu lassen oder zu entspannen. Erfahrungen sind zwar ähnlich kurz wie *Demos*, sie sind aber in sich geschlossen und wirken nicht unvollständig, da sie in exakt diesem Umfang geplant wurden. Experiences sind aufgrund ihrer Kürze und ihrer Einfachheit meist gut geeignet, um VR-Neulingen einen Eindruck zu geben, ohne dass sie vollständig in ein Spiel abtauchen müssen.
- **Spiele/Games**
  Als „Spiele“ und „Games“ bezeichnet man alles, wo das *Spielen* im Vordergrund steht und der Spieler auch aktiv in das Geschehen einbezogen ist. Die Spieler erwarten in der Regel von einem Spiel, dass sie durch die Spielmechanik oder eine erzählte Geschichte mehrere Stunden unterhalten werden. Spiele, die zu wenig Spielzeit oder Umfang bieten, werden in den Foren häufig negativ als „Experience“ oder „Demo“ bezeichnet, auch wenn sie von dem Entwickler nicht als solche angedacht waren.
- **Anwendungen/Applications**
  Als „Anwendung“ oder „Application“ werden im Prinzip alle nicht spielerischen *Programme* bezeichnet, die man in Virtual Reality bedienen kann. Anwendungen erlauben dir, bestimmte Tätigkeiten in VR auszuüben: zum Beispiel Zeichnen, 3D-Modellieren oder Fotos und Videos zu betrachten etc.

2.7 Seated, Standing & Room Scale

Je nachdem welche der VR-Brillen du verwendest, kannst du unterschiedliche Arten von Eingabelösung für deine VR-Software verwenden, welche dir vollständig unterschiedliche Möglichkeiten für dein Spiel liefern. Ich beziehe mich in diesem Kapitel zwar auf „Spiele“, diese Kategorien existieren jedoch für jede Art von VR-Software.

Man unterscheidet zwischen diesen drei Kategorien:

- **Seated**
  Als „Seated“ bezeichnet man VR-Software, die du im Sitzen verwendest. Dabei ist es egal, ob du die Erfahrung mit einem normalen Gamepad oder Motion-Controllers steuerst. Bei *Seated-Games* bewegst du dich in der Regel mit einem Gamepad durch die virtuelle Welt.

Es gibt jedoch auch Spiele, bei denen du kein zusätzliches Gamepad benötigst und ausschließlich über Blicke mit der virtuellen Welt interagierst.

Diese Spiele können grundsätzlich mit jeder VR-Brille realisiert werden. Folgende Brillen sind jedoch speziell für diese Art vorgesehen: *Oculus Rift (ohne Oculus Touch), GearVR, Daydream, Cardboard*

- **Standing**
  Als „Standing" bezeichnet man VR-Software, die du im Stehen verwendest. Dabei kannst du zwar schon mal einen Schritt nach links oder nach rechts machen, insgesamt ist das Spiel aber darauf ausgelegt, dass du an einem Punkt stehen bleibst. Für gewöhnlich interagierst du hier über Motion-Controller mit der Umwelt. Du kannst also nach Dingen greifen, um sie aufzuheben. Bei Erfahrungen, die du im Stehen erlebst, kannst du dich häufig nicht über einen Thumbstick drehen, sondern musst dich in der Realität umdrehen. Einige Spiele erlauben es dir, dich mithilfe eines Gamepads fortzubewegen.

  Für diese Art von Erfahrung sind vor allem *Oculus Rift mit Oculus Touch, aber auch die HTC Vive* vorgesehen. Letztere ist aufgrund der fehlenden Thumbsticks jedoch besser für *Room-Scale* geeignet.

- **Room Scale**
  Als „Room Scale" bezeichnet man VR-Software, die voll und ganz darauf ausgelegt ist, dass du nicht nur auf der Stelle stehst, sondern zur Fortbewegung tatsächlich mit deiner VR-Brille in deinem Raum herumläufst. Damit du nicht gegen Wände läufst, definierst du vorher eine sogenannte „Play Area" (dt. „Spielfläche"). Die VR-Brillen blenden dann virtuelle Wände ein, wenn du dich den Grenzen dieses Bereiches näherst.

  Viele *Room-Scale*-Spiele verzichten vollständig auf andere Fortbewegungsarten und beschränken das gesamte Spiel auf die vorhandene Spielfläche. Andere Spiele ermöglichen es dem Spieler zusätzlich noch, sich wie bei *Standing*-Spielen zu teleportieren oder teilweise auch frei herumzulaufen. Der Übergang zwischen *Standing* und *Room Scale* ist fließend. Zu welcher Kategorie eine Software nun gehört, lässt sich am besten dadurch festmachen, wie wichtig das reale Herumlaufen für das Spiel ist.

  Für Room-Scale-Software ist *die HTC Vive am besten geeignet, dicht gefolgt von der Oculus Rift mit Oculus Touch*. Das kamerabasierte Tracking der Oculus Rift ist für größere Bereiche etwas schlechter geeignet als das laserbasierte Tracking-System der HTC Vive. In Bereichen von bis zu 3 x 3 Metern ist der Unterschied (*wenn man die Oculus Rift mit drei Sensoren betreibt*) jedoch häufig kaum bemerkbar.

2.8 „Bekannte" Probleme

In den Medien werden manche Probleme von Virtual Reality, wie zum Beispiel Übelkeit, sehr stark in den Vordergrund gerückt. Dabei ist das Problem eigentlich gar nicht so schlimm, wenn man es richtig angeht. Da Virtual Reality aber noch ein junges Thema ist, gibt es trotzdem noch ein paar Probleme, über die du Bescheid wissen solltest. Denn früher oder später wirst du wahrscheinlich mit Fragen zu diesen Problemen konfrontiert werden. Deshalb werden wir uns nun einige der bekanntesten Probleme ansehen.

2.8.1 Auflösung & Fliegengitter-Effekt

Bei aktuellen Virtual-Reality-Brillen ist die Auflösung eigentlich schon ziemlich gut. Bei dem ersten *Development Kit* von *Oculus* hatte das Display gerade mal eine Auflösung von 1280 x 720 Pixel. Das entspricht nur 640 x 720 Pixel pro Auge. Durch die höhere Auflösung bei der aktuellen Generation konnten die Hersteller zumindest schon den *Fliegengitter-Effekt* unterbinden. Bei diesem Effekt ist ein feines Gitter, das sich durch das gesamte Blickfeld zieht, sichtbar. Entstanden ist dieses Gitter durch die Abstände zwischen den einzelnen Pixeln des Displays. Es wirkt so, als würde man durch ein Fliegengitter aus dem Fenster schauen.

Bei aktuellen VR-Brillen kann man die einzelnen Pixel, dank der höheren Auflösung und ein paar Tricks der Hersteller, nicht mehr wirklich erkennen. Trotzdem ist die Auflösung noch nicht perfekt: Gerade besonders kleine Gegenstände mit vielen Details, die ggf. auch etwas weiter weg vom Spieler stehen, können häufig nicht so gut erkannt werden. Das macht sich zum Beispiel bei Flug- und Rennsimulationen bemerkbar, wo man Gegner oder kommende Kurven unter Umständen erst zu spät erkennt. Hier bist du als Entwickler dann gefragt: Du musst deinen Spielern entsprechende virtuelle Hilfsmittel zur Verfügung stellen, damit sie Kurven oder Gegner trotzdem früh genug bemerken.

2.8.2 Hardwareanforderungen

In Virtual Reality ist die Performance der Anwendung, also wie stabil sie läuft, ein sehr wichtiger Punkt. Auf einem normalen Monitor fällt es nicht sofort auf, wenn die Frames per Second mal unter 60 fallen. Mit einer Virtual-Reality-Brille sieht das ganz anders aus. Einige Brillen arbeiten zudem mit einer deutlich höheren Wiederholrate: Für Oculus Rift und HTC Vive sollten zum Beispiel konstant 90 FPS eingehalten werden. Die meisten VR-Brillen besitzen zwar technische Vorkehrungen, welche einen kurzen FPS-Abfall abfangen können (*Timewarp*), du solltest dich aber nicht darauf verlassen. Es empfiehlt sich, solche kritischen Situationen bestmöglich zu verhindern. Um die 90 FPS in den meisten Spielen garantieren zu können, haben sich *Oculus* und *HTC* dazu entschieden, höhere Hardwareanforderungen an die Computer der Spieler zu stellen. Mit einer niedrigeren Empfehlung würden eventuell nicht alle Spiele flüssig laufen, was zu Missmut bei den Spielern führen könnte.

Im Mobile-VR-Bereich (GearVR, Daydream) müssen nur 60 FPS konstant gehalten werden, dafür steht dir hier aber auch deutlich weniger Hardware-Power zur Verfügung. Damit alle Apps flüssig laufen, stellen Google und Samsung für ihre hochqualitativen VR-Brillen deswegen ebenfalls höhere Mindestanforderungen an die Hardware. Da es für Google Cardboard keine solche Mindestanforderung gibt, kann jeder die App laden, was teilweise zu negativen Reviews aufgrund schlechter Performance führt.

In Virtual Reality gilt also noch mehr als sonst beim Entwickeln: Optimieren, optimieren, optimieren!

2.8.3 Kurz- und Weitsichtigkeit

Du bist kurz- oder weitsichtig? Das ist in vielen Fällen kein Problem in Virtual Reality. Die meisten Brillen können so eingestellt werden, dass man problemlos eine Brille darunter tragen kann. Bei Brillen mit großem Rahmen kann es jedoch sein, dass deine Brille trotzdem nicht passt.

Wenn du weitsichtig bist, kannst du vielleicht sogar vollständig auf deine Brille verzichten. In einer Virtual-Reality-Brille fokussieren deine Augen nämlich für gewöhnlich auf „unendlich“, weshalb du, je nach Grad deiner Sehschwäche, keine Sehhilfe in Virtual Reality benötigst. Kurzsichtige Menschen profitieren jedoch nicht davon. Manche Brillen, wie Samsung GearVR, bieten extra eine Einstellung zum Anpassen an die eigene Sehschwäche. Je nach Stärke wirst du aber unter Umständen deine Brille oder Kontaktlinsen trotzdem tragen müssen. Hier gilt: ausprobieren!

Falls dir die Fähigkeit fehlt, dreidimensional zu sehen, bringt dir Virtual Reality übrigens, anders als ein 3D-Fernseher, trotzdem etwas: Auch ohne 3D-Effekt ist Virtual Reality, wegen der natürlichen Interaktion und dem Gefühl, dass dich die Welt umgibt, völlig anders als alles, was du bis jetzt kennst.

2.8.4 Simulator Sickness

Simulator Sickness ist eine Form der „Motion Sickness“ und wird deshalb häufig auch so bezeichnet. Simulator Sickness beschreibt Unwohlsein, das durch simulierte Umgebungen entstehen kann. Für Simulator Sickness sind viele Dinge verantwortlich, die teilweise nur vollkommen unterbewusst wahrgenommen werden. Einer der stärksten und offensichtlichen Punkte ist jedoch, dass deine Augen deinem Gehirn sagen, dass du dich in Bewegung befindest, dein Gleichgewichtsorgan allerdings meldet, dass du im Moment stehst oder sitzt. Es ist also genau andersherum als die Form von *Motion Sickness*, die manche Leute erleben, wenn sie im Auto lesen.

Während die Ursachen bei beiden Arten der *Motion Sickness* unterschiedlich sind, haben beide die gleiche Auswirkung auf dein Gehirn: Es empfängt widersprüchliche Signale. Diese Situation kann in der Natur zum Beispiel bei einer Vergiftung auftreten. Um sich vor der Vergiftung zu schützen, möchte der Körper das vermeintliche Gift herausspülen, weshalb die berühmt-berüchtigte Übelkeit entsteht. Die Übelkeit ist also lediglich eine Schutzfunktion des Körpers.

Die „Simulator Sickness“ hat mit der aktuellen Brillen-Generation aber schon stark abgenommen. Während bei den ersten *Developer Kits* noch fast jedem am Anfang zumindest „komisch“ wurde, ist dies bei den aktuellen VR-Brillen bei Weitem nicht mehr der Normalfall. Viele Spieler erleben von Anfang an gar keine *Simulator Sickness* und können sofort auch intensive Spiele spielen.

Aber auch bei der aktuellen VR-Generation kann es passieren, dass einem empfindlicheren Spieler hin und wieder übel wird. Das passiert allerdings meist bei Spielen, die nicht besonders auf Komfort ausgelegt sind.

Gehörst du zu diesen Personen, empfehle ich dir, es langsam anzugehen und die Komfortfunktionen der Spiele zu nutzen. Es bringt nichts, wenn du sofort alle Komfortfunktionen abschaltest, um das „beste“ Erlebnis zu haben, und du danach unbewusst VR mit Übelkeit assoziierst. Nahezu jede empfindliche Person kann sich durch regelmäßiges Spielen an Virtual-Reality-Brillen gewöhnen.

Komfortfunktionen

In der Virtual-Reality-Entwicklerszene hat sich schnell herausgestellt, dass Simulator Sickness vor allem durch künstliche Rotation und Bewegung verursacht wird.

Künstliche Bewegungen sind alle Bewegungen, die du nicht tatsächlich mit deinem Körper ausführst. Dazu gehört zum Beispiel Laufen und Umdrehen per Tastendruck oder Thumbstick. Leider sind diese künstlichen Bewegungen jedoch in den meisten aktuellen Spielen wichtige Grundfunktionen. Da es deshalb schwerfällt, völlig darauf zu verzichten, haben VR-Entwickler von Anfang an versucht, Methoden zu finden, künstliche Bewegungen, durch sogenannter *Komfortfunktionen*, trotzdem zu ermöglichen.

In Kapitel 4 werden wir uns unter anderem auch verschiedene Möglichkeiten zur Fortbewegung in Virtual Reality noch genauer ansehen. Dabei werden wir auch die Komfortfunktionen unter die Lupe nehmen.

Realismus & Schutz des Spielers

Ein wichtiges Argument beim Thema Übelkeit ist auch, dass wir uns in Videospielen häufig völlig übertrieben verhalten. Stell dir zum Beispiel eine Flugsimulation vor. In Videospielen fliegt man häufig eine Rolle und einen Looping nach dem anderen. Spielst du ein ähnliches Spiel mit einer Virtual-Reality-Brille, wird dir sehr wahrscheinlich übel. Abgesehen von den künstlichen Bewegungen kann es auch schlicht daran liegen, dass dir im echten Leben auch schlecht werden würde, wenn du diese Manöver fliegst. Es ist also eventuell sinnvoll, dass du, als Entwickler, unerfahrene Spieler an die Hand nimmst, um sie vor sich selber zu schützen. In einem *optionalen Komfortmodus* könntest du zum Beispiel die Rollgeschwindigkeit des Flugzeugs verlangsamen oder vollständige Überschläge generell verhindern.

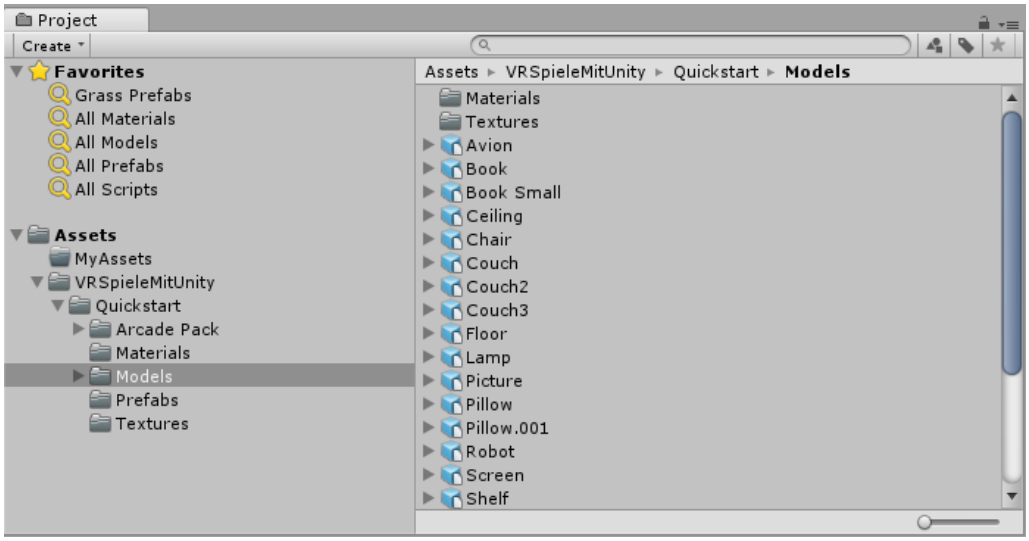

Bild 3.48 Diese Ordner solltest du nach dem Importieren in deinem Project Browser finden.

In dem Ordner *Models* findest du diverse 3D-Modelle, die wir in dem Beispiel verwenden werden. In dem Models-Ordner findest du außerdem einen *Materials*- und einen *Textures*-Ordner, welche jeweils die *Textures* und *Materials* enthalten, die aus den FBX-Dateien der Modelle stammen. Der Ordner Arcade Pack enthält ein Set von verschiedenen Arcade-Automaten, die ich für dich herausgesucht habe und die wir in dem Jugendzimmer aufstellen werden.

3.9.1 Rohbau für den Raum erstellen

Als Erstes erstellen wir einen Raum für unser Jugendzimmer. Woraus besteht ein Raum? Genau: aus einem Boden, vier Wänden und einer Decke. Los geht's.

1. Zu Beginn erstellen wir eine neue *Scene*.

   (In der Toolbar: **File/New Scene**)

2. Eine neue Scene enthält immer eine *Camera* und ein *Directional Light*, welche du in der Hierarchy sehen solltest.

3. Öffne jetzt im *Project Browser* den Ordner *VRSpieleMitUnity/Quickstart/Models*. Dort findest du ein Modell mit dem Namen *Floor* (dt. „Boden"). Ziehe das Modell aus dem *Project Browser* in die *Scene View*.

4. In der *Scene View* siehst du jetzt eine braune Fläche und in der *Hierarchy* ein neues GameObject mit dem Namen *Floor*.

5. Um eine gute Übersicht zu behalten, ist es sinnvoll, GameObjects um den Ursprung, also um Position *(0, 0, 0)* herum, anzuordnen. Verschiebe das „Floor"-GameObject so, dass es exakt an der Position (0, 0, 0) liegt. Das machst du am einfachsten, indem du das *Game-*

Object in der *Hierarchy* auswählst und dann im *Inspector* unter *Transform* die Position auf (0, 0, 0) setzt.

6. Markiere das GameObject im Inspector als *Static*.

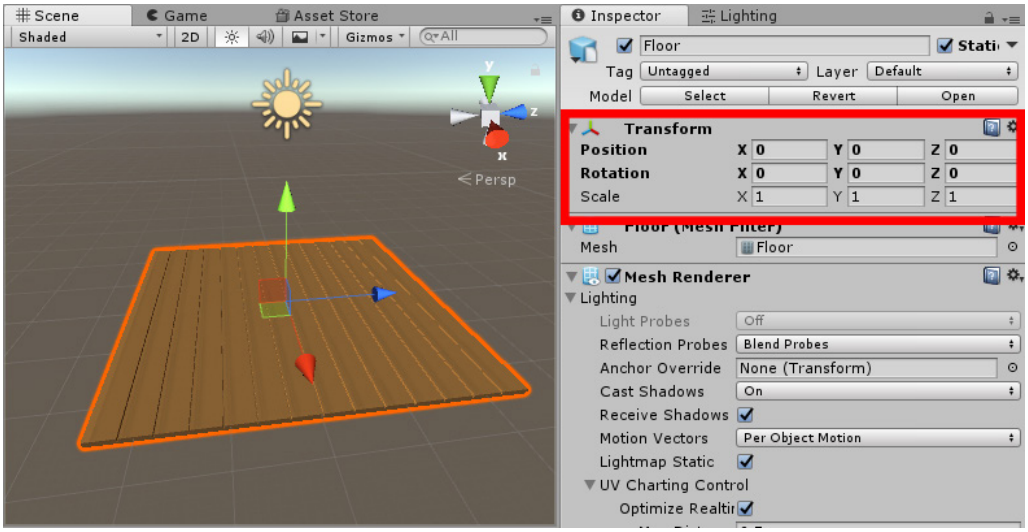

Bild 3.49 In dem rot markierten Bereich kannst du die lokale Position, Rotation und Skalierung eines GameObjects bestimmen.

Als Nächstes fügen wir eine Wand hinzu. Das geschieht genauso wie beim Boden über Drag & Drop und anschließende Positionierung über den *Inspector*:

1. In dem Ordner *VRSpieleMitUnity/Quickstart/Models* findest du ein Modell mit dem Namen *WindowWall* (dt. „Fensterwand“). Ziehe das Modell aus dem *Project Browser* in die *Scene View*.
2. Diese Wand musst du jetzt an einen der Enden des Bodens positionieren. Setze die *Position* dafür auf *(-2.5, 0, 0)*.
3. Dir fällt wahrscheinlich auf, dass das Wand-Modell von einer Seite durchsichtig ist. Das liegt daran, dass dieses Modell für Innenraum-Scenes optimiert ist. Da der Spieler ohnehin immer nur die Innenseite der Wand sieht, wird die unsichtbare Seite weggelassen, um Rechenleistung zu sparen. Das GameObject muss allerdings noch so gedreht werden, dass auch die richtige Seite nach innen zeigt.

   Setze deshalb im *Inspector* die *Rotation* auf *(0, 180, 0)*, um das GameObject um 180° zu drehen.
4. Markiere das GameObject als *Static*.

Bild 3.50 zeigt, wie das GameObject nun aussehen sollte.

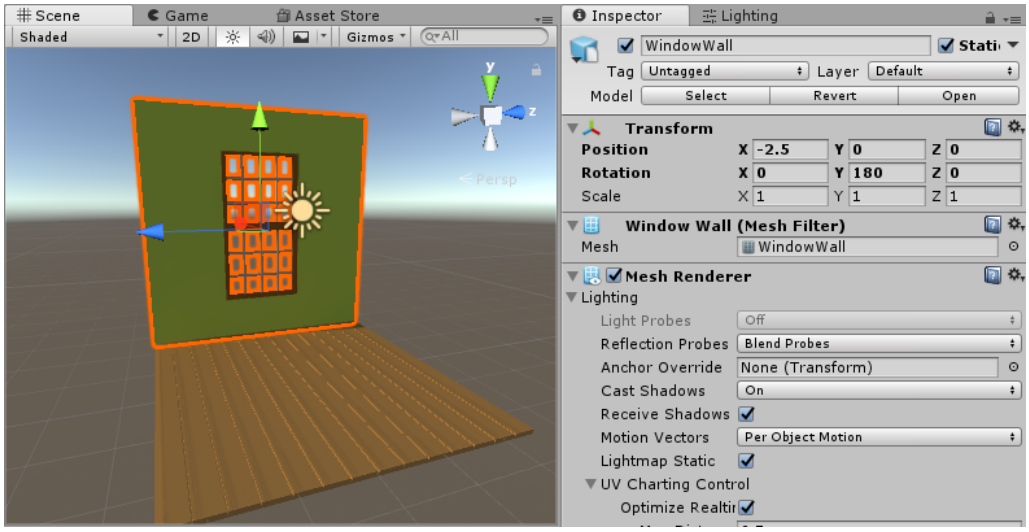

Bild 3.50 So sollte deine Scene mit dem neuen GameObject aussehen.

Eine Wand geschafft, fehlen nur noch drei. Kleine Erinnerung: Wenn du die rechte Maustaste gedrückt hältst, kannst du mit **W, S, A, D** und **Mausbewegungen** durch die Scene fliegen.

1. Für die restlichen drei Wände verwenden wir das „Wall"-Modell, welches du ebenfalls unter *VRSpieleMitUnity/Quickstart/Models* findest. Ziehe das Modell aus dem *Project Browser* in die *Scene View*.
2. Positioniere das neue „Wall"-GameObject an der Position *(0, 0, -2.5)* und mit einer Rotation von *(0, 0, 0)* und markiere die Wand als *Static*.
3. Wiederhole Schritt 1 und 2 für eine weitere Wand, verwende aber als Position *(2.5, 0, 0)* und als Rotation *(0, 270, 0)*.
4. Wiederhole Schritt 1 und 2 ein weiteres Mal für die letzte Wand, verwende diesmal als Position *(0, 0, 2.5)* und als Rotation *(0, 180, 0)*.

Spätestens jetzt bemerkst du auch einen großen Vorteil der Wände, die von einer Seite durchsichtig sind: Du kannst noch bequem von außen hineinschauen. Unter anderem aus diesem Grund ist auch die Decke, die wir als Nächstes einfügen werden, von nur einer Seite modelliert.

1. Wie die anderen Modelle auch, findest du die Decke ebenfalls in dem *VRSpieleMitUnity/Quickstart/Models*-Ordner. Ziehe das *Ceiling*-Modell ebenfalls in die Scene.
2. Positioniere sie exakt über dem Boden an der Position *(0, 5, 0)*.
3. Markiere das *GameObject* als *Static*.

Du solltest jetzt einen vollständigen Raum mit vier Wänden, einem Fenster, einem Boden und einer Decke in deiner Scene haben. Wie in Bild 3.51 ist eine der vier Wände immer nicht sichtbar, weil du von hinten durch diese hindurchschaust. Wenn du aber in der Scene um den Raum herumfliegst, kannst du überprüfen, dass alle Wände vorhanden und korrekt positioniert sind.

Kontrolliere nochmals, ob auch wirklich alle GameObjects, die du erzeugt hast, im *Inspector* oben rechts als *Static* markiert sind.

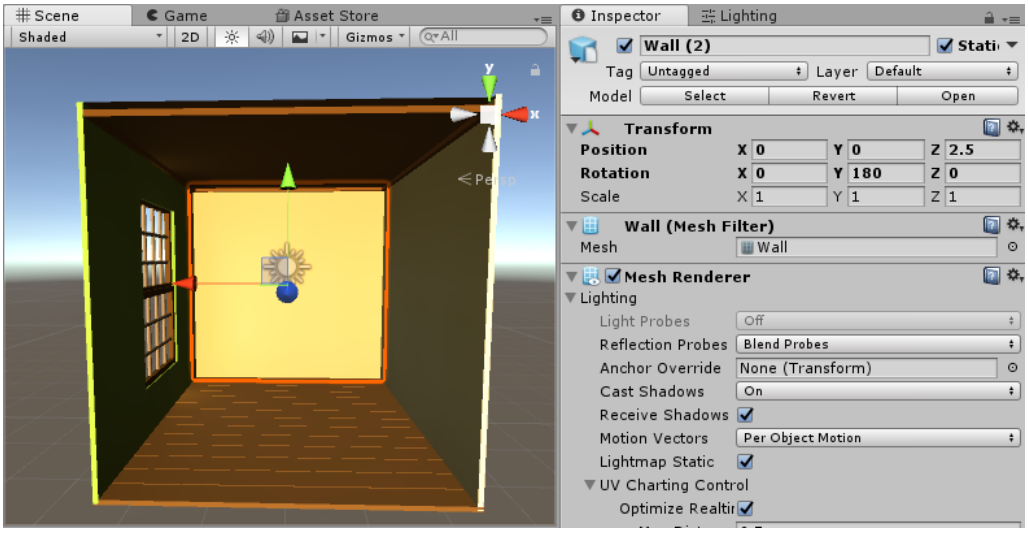

Bild 3.51 So sollte der Raum in deiner Scene nun aussehen.

3.9.2 Scene speichern

Bevor du weitermachst, solltest du erst einmal deinen Fortschritt speichern. Lege dazu in deinem *MyAssets*-Ordner einen neuen Unterordner mit dem Namen *Scenes* an.[2] Du solltest danach also den Pfad *MyAssets/Scenes* in deinem Projekt haben.

Speichere anschließend die Scene über die Toolbar: **File/Save Scene**. Wähle in dem Speichern-Dialog den soeben erstellten Ordner aus und gebe der Scene einen aussagekräftigen Namen (z. B. *Quickstart_Jugendzimmer*).

Regelmäßig speichern

Speichere deinen Fortschritt regelmäßig. Sollte doch mal etwas abstürzen und deine Arbeit geht verloren, ist das sehr ärgerlich. Speichere deine Scene deshalb regelmäßig ab. Über das Tastenkürzel **STRG + S** kannst du jederzeit speichern, ohne dass dein Arbeitsfluss gestört wird.

[2] Zur Erinnerung: Klicke dazu mit der rechten Maustaste auf den MyAssets-Ordner und wähle Create/Folder.

3.9.3 Licht anpassen

Jetzt, wo der Rohbau für unser Jugendzimmer steht, sollten wir uns um die grobe Lichtsituation in dem Raum kümmern. Wenn du von außen hineinschaust, sieht zwar alles okay aus, sobald du in den Raum hineinfliegst, stellst du aber fest, dass es dunkel ist. Das liegt daran, dass derzeit kein Sonnenlicht durch das Fenster in den Raum scheint und nur ein wenig indirektes Licht in den Raum fällt.

Um das zu ändern, musst du die Sonne, also das *Directional Light* in der Scene, so drehen, dass das Licht durch das Fenster fällt:

1. Wähle dazu zunächst das *Directional Light* in der Hierarchy aus.
2. Aktiviere nun das Rotations-Werkzeug in der Toolbar oben.
3. An dem Licht werden jetzt, wie in Bild 3.52, runde Helfer angezeigt. Greife und ziehe diese mit der Maus, um das GameObject zu drehen. Anhand der gelben Linien, die von dem Licht-Icon ausgehen, erkennst du die Strahlrichtung.
4. Wenn du mit der Strahlrichtung zufrieden bist, solltest du in dem *Light*-Component die Eigenschaft *Mode* auf *Baked* setzen. Damit sorgst du dafür, dass für alle statischen Objekte eine *Lightmap* berechnet wird.
5. Da es mir in dem Raum immer noch ein wenig zu dunkel war, habe ich außerdem die Intensität des Lichtes (*Intensity*) von 1 auf *1.1* und den Multiplikator für indirektes Licht (*Indirect Multiplier*) von 1 auf *1.5* erhöht.

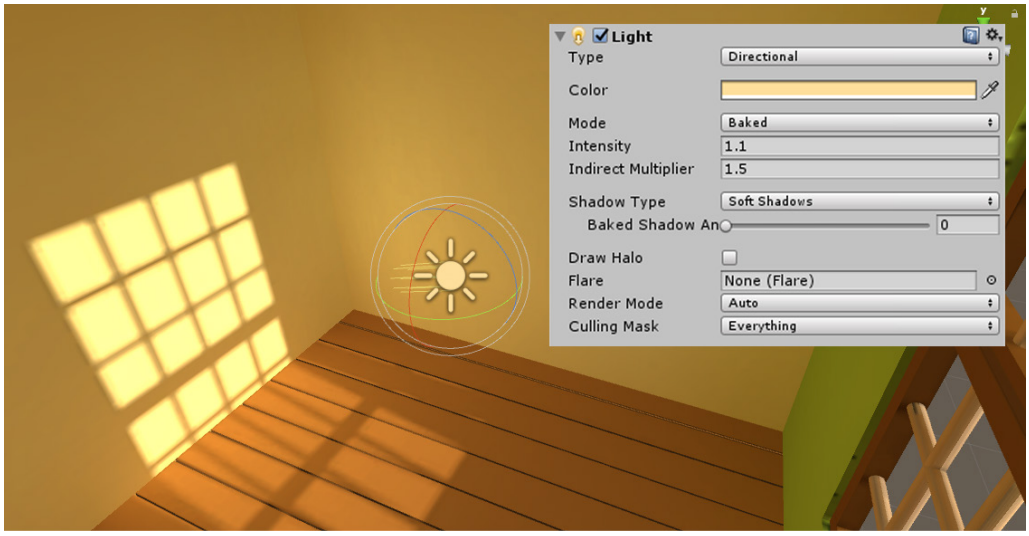

Bild 3.52 So könnte dein konfiguriertes Licht aussehen.

Sobald du mit dem Umstellen auf *Baked* dafür gesorgt hast, dass Unity Lightmaps berechnen soll, startet Unity auch mit der Berechnung. Unten rechts im Editor kannst du dann, die in Bild 3.53 dargestellte, Fortschrittsanzeige sehen. Sobald sie verschwindet, sind die Berechnungen abgeschlossen und die Lightmaps in der *Scene View* aktualisieren sich. Je mehr Objekte sich in der Scene befinden, desto länger dauert die Berechnung. Nimmst du

eine Änderung an der Scene vor, während die Berechnung läuft, wird die aktuelle Berechnung abgebrochen und mit der Änderung neu gestartet. Du musst also nicht warten, bis die Berechnung fertig ist, damit du weiterarbeiten kannst.

Bild 3.53 Die Fortschrittsanzeige gibt dir Aufschluss darüber, was Unity derzeit berechnet.

3.9.4 Collider hinzufügen

Weder der Boden noch die Wände besitzen derzeit einen Collider, das bedeutet, physikalische Gegenstände (*Rigidbodys*) würden einfach durch sie hindurchfallen. Gut, derzeit haben wir auch noch keine beweglichen Gegenstände im Raum, aber Böden und Wände mit Collidern auszustatten, hat noch einen anderen Vorteil: Wenn du ein Modell aus dem Project Browser in die *Scene View* ziehst, platzieret sich das neue *GameObject* automatisch auf oder an Objekten, die einen *Collider* zu besitzen. Wie in Bild 3.54 dargestellt, wird das Füllen von Räumen mit Collidern an dem GameObject des Bodens und an GameObjects der Wände deutlich einfacher.

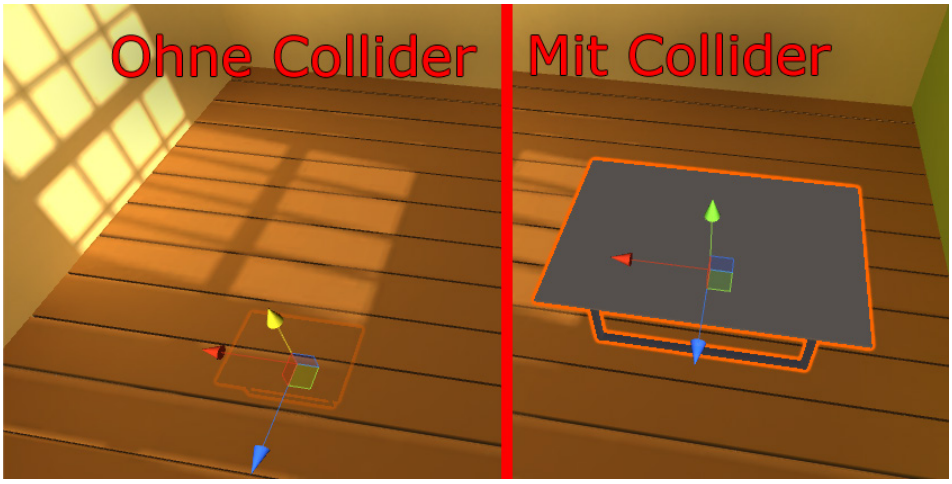

Bild 3.54 Links: Ohne Umgebungs-Collider landen neue GameObjects irgendwo.
Rechts: Mit Umgebungs-Collider platzieren sich hineingezogene Objekte automatisch pixelgenau auf den Collidern.

Um uns die Arbeit zu erleichtern, fügen wir also erst einmal Collider zu dem Boden und den Wänden hinzu. Da sowohl der Boden als auch die Wände rechteckig sind, verwenden wir dafür einen *Box Collider*:

1. Wähle das *Floor*-GameObject in der Hierarchy aus.
2. Klicke im *Inspector* auf **Add Component/Physics/Box Collider**.

   Der erzeugte *Box Collider* passt sich automatisch der Größe des durch den *MeshRenderer* dargestellten Modells, also unserem Boden, an.

3. Widerhole die Schritte 1 und 2 für die GameObjects der Wände: *WindowWall*, *Wall*, *Wall (1)*, *Wall (2)*.

Wenn du das erledigt hast, kannst du die GameObjects in der Hierarchy auswählen und siehst dann, wie in Bild 3.55, in der *Scene View* eine grüne Box, die den Collider darstellt.

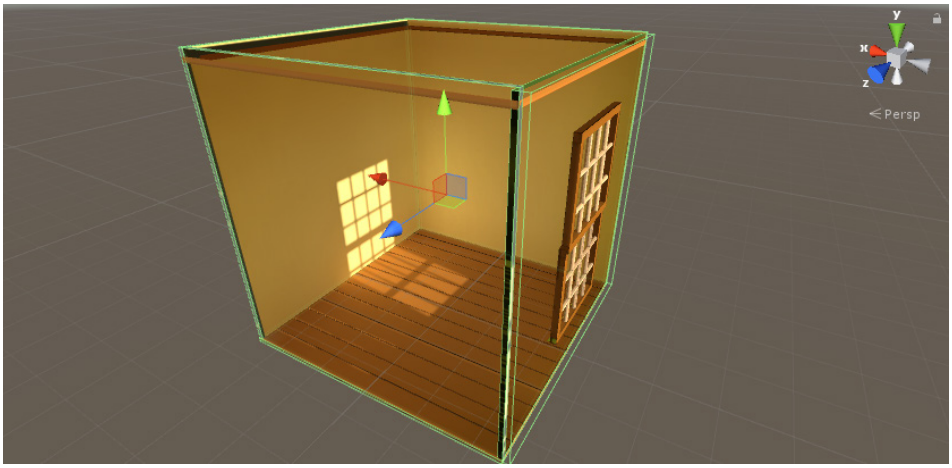

Bild 3.55 An den grünen Linien erkennst du die Collider.
(Für den Screenshot wurde die orange Outline in der *Scene View* über das Gizmo-Menü deaktiviert.)

3.9.5 Möbelstücke einräumen

Jetzt, wo der Raum steht, kannst du ihn mit Möbeln füllen. In diesem Abschnitt geht es zunächst um große Dinge wie Tische, Regale oder Couches. Um die Details kümmern wir uns im nächsten Abschnitt.

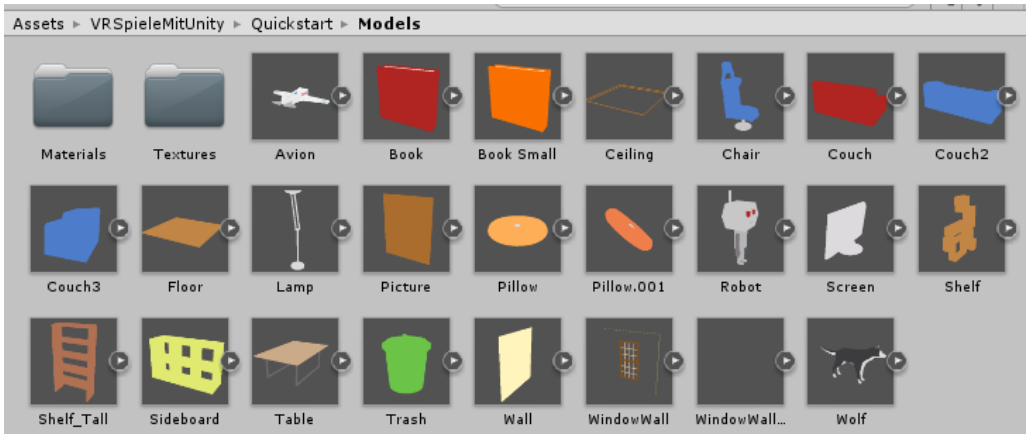

Bild 3.56 Diese Modelle stehen dir in dem Quickstart-Package zur Verfügung.

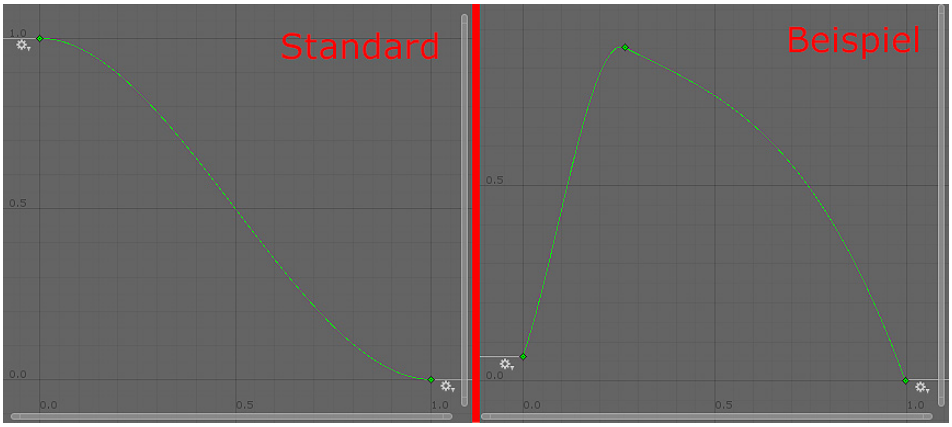

Bild 10.6 Links die Standardkurve, rechts ein alternatives Beispiel, bei dem das Raumschiff am Anfang langsamer beschleunigt. Die horizontale Achse beschreibt die Fluggeschwindigkeit (1.0 = Max Speed). Die vertikale Achse beschreibt die Beschleunigung (1.0 = Max Acceleration).

4. Sobald du das *SpaceShipPhysics*-Component hinzugefügt und es nach deinen Wünschen angepasst hast, kannst du das zweite neue Script, *SpaceShipControls*, zu dem *PlayerSpaceship*-GameObject hinzufügen. Hier musst du keine weiteren Einstellungen vornehmen.

Das wäre geschafft! Du kannst jetzt das Spiel starten, dich mit deiner VR-Brille in das Cockpit setzen und losfliegen! Am besten natürlich mit einem Gamepad oder Joystick. Per Tastatur ist aber auch möglich.

Korrekt im Cockpit sitzen

Setze dich bequem hin und schaue geradeaus. Nutze dann die RECENTER-Taste, um dich perfekt innerhalb des Sitzes zu positionieren. Ist die Perspektive immer noch komisch, überprüfe die Position der *Main Camera* in deiner Scene.

10.2.3 Starfield/Weltraumstaub

Deine Flugmechanik funktioniert nun und du kannst in Virtual Reality durch den Weltraum fliegen. Ein richtiges Geschwindigkeitsgefühl kommt dabei aber noch nicht wirklich auf. Das liegt daran, dass dir die Referenzpunkte in der Umgebung fehlen. Dies ist derselbe Effekt wie in einem Flugzeug, wo man die hohe Geschwindigkeit nicht mehr bemerkt, sobald man über den Wolken ist. Eine Lösung für dieses Problem wäre, die Scene mit sehr vielen Objekten zu füllen, sodass man ständig an etwas vorbeifliegt. Höchstwahrscheinlich wäre dies für viele neue VR-Anwender aber zu viel des Guten, denn zu viele Referenzpunkte zusammen mit zu vielen Drehungen kann *Simulator Sickness* verursachen. Außerdem würde eine derart gefüllte Scene wahrscheinlich auch auf vielen Computern an die Hardwaregrenze stoßen.

Wir verwenden deswegen einen in Weltraumspielen gängigen Effekt, um dem Spieler die Geschwindigkeit zu veranschaulichen. Der in Bild 10.7 dargestellte Effekt wird häufig als *Starfield* bezeichnet. Wie in der linken Hälfte des Bildes zu sehen, sollen nur einzelne Punkte sichtbar sein, die sich langsam durch den Raum bewegen, wenn das Raumschiff langsam fliegt. Je schneller das Raumschiff fliegt, desto mehr sollen die Punkte zu Linien verzerrt werden.

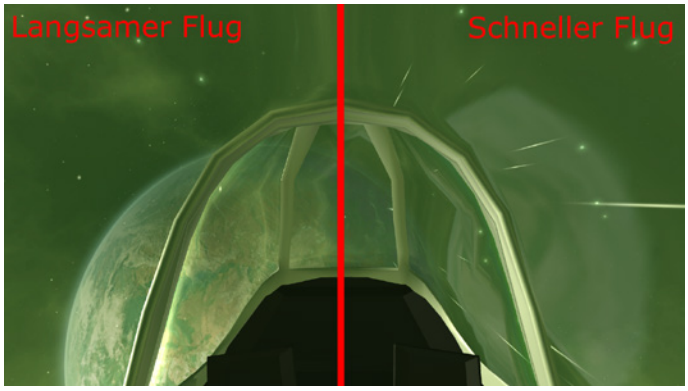

Bild 10.7 So soll der Effekt im langsamen und schnellen Flug aussehen, um ein gutes Geschwindigkeitsgefühl zu vermitteln.

Realisieren werden wir den Effekt über ein *Particle System*, weil dieses uns erlaubt, sehr viele einzelne Objekte automatisiert darzustellen. Aus Performance-Gründen kannst du kein riesiges *Particle System* erstellen, das sich über die gesamte Scene erstreckt und Millionen von Partikeln erzeugt. Stattdessen befestigen wir ein *Particle System* an dem Raumschiff, welches nur rund um das Raumschiff herum Partikel erzeugt. Um diesen Effekt zu realisieren, musst du die nachfolgenden Punkte befolgen. Bild 10.8 zeigt zum Vergleich, wie diese Variante am Ende aussehen sollte.

1. In dem *VRSpieleMitUnity/SpaceshipExample/Prefabs*-Ordner findest du ein *Prefab* mit dem Namen *StarfieldParticleSystem*. Ziehe dieses Prefab vom *Project Browser* in die *Hierarchy*, und zwar auf das *PlayerSpaceship*-GameObject, damit es als Kind angelegt wird.
2. Die Position sollte automatisch bereits auf (0,0,9) stehen und die Rotation auf (0, 0, 0). Wenn nicht, passe beide Werte an.

Wenn du dir das *Particle System* des soeben erzeugten GameObjects im *Inspector* ansiehst, kannst du seine Konfiguration sehen. Das *Particle System* ist so konfiguriert, dass sich die Partikel im *World-Space* bewegen. Das bedeutet, wenn sich das *Particle System* selbst bewegt, bewegen sich bereits erzeugte Partikel nicht automatisch mit, sondern bleiben an ihrer Stelle. Durch diese Option wird es auf den Spieler so wirken, als ob die gesamte Welt voll mit Partikeln ist.

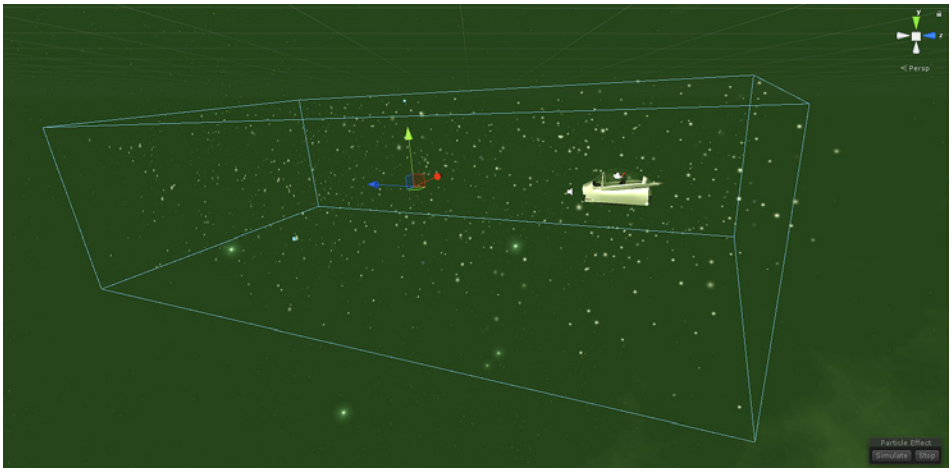

Bild 10.8 So sollte das Particle System in deiner Scene aussehen.
(Anzahl und Größe der Partikel wurden zur besseren Sichtbarkeit im Screenshot erhöht.)

Wenn du das Spiel jetzt einmal testest, siehst du bereits die einzelnen Partikel an dir vorbeirasen, während du fliegst. Für einen Starfield-Effekt müssen die einzelnen Partikel allerdings noch zu lang gezogenen Linien verzerrt werden, sobald du schneller an ihnen vorbeifliegst. Dieser Effekt lässt sich über ein Script sehr einfach realisieren:

Erzeuge in deinem *MyAssets/Scripts*-Ordner ein neues *C#-Script* mit dem Namen *StarfieldParticleEffect* und ersetze den automatisch generierten Inhalt mit dem Code aus Listing 10.8.

Listing 10.8 StarfieldParticleEffect.cs – zieht die Partikel, abhängig von der Geschwindigkeit des Raumschiffes, in die Länge

```
using UnityEngine;
// Erwarte ein SpaceShipPhysics Component

[RequireComponent(typeof(SpaceShipPhysics))]
public class StarfieldParticleEffect : MonoBehaviour {
    public ParticleSystemRenderer starfieldParticleSystem;
    private SpaceShipPhysics usedShip;
    void Start()
    {
        usedShip = GetComponent<SpaceShipPhysics>();
    }
    void Update () {
        // Lese aktuelle Geschwindigkeit
        float currentVelocity = usedShip.ForwardVelocity;
        // Aktiviere den Effekt ab einer Geschwindigkeit von 2
        if (currentVelocity > 2) {
            // Skaliere die Länge der Partikel in Abhängigkeit
            // von der aktuellen Geschwindigkeit
            starfieldParticleSystem.lengthScale = currentVelocity;
        } else { // Ansonsten skaliere auf 1 (Standardwert)
            starfieldParticleSystem.lengthScale = 1;
        }
    }
}
```

Das neu erzeugt Script muss du jetzt mittels Drag & Drop oder **Add Component** zu dem *PlayerSpaceship*-GameObject hinzufügen. Damit das neue Component weiß, welche Partikel es verzerren soll, musst du anschließend im *Inspector* der Eigenschaft *Starfield Particle System* das *StarfieldParticleSystem-GameObject* aus deiner Scene zuweisen.

10.2.4 Schadenssystem & Plasmakanone

Für epische Weltraumschlachten fehlt dem Raumschiff natürlich noch seine Bewaffnung. Deswegen entwickeln wir als Nächstes eine Plasmakanone für das Schiff. Die Kanone soll jedoch kein einfaches Board-Geschütz werden, das immer nur nach vorne schießt. Um die Steuerung zu vereinfachen, soll der Pilot die Plasmakanone mittels Headtracking steuern können. Sie soll also immer in die Blickrichtung des Spielers schießen.

Die Plasmakanone macht natürlich nur Sinn, wenn man damit auch Objekte zerstören kann. Deswegen werden wir zu der Kanone auch ein Schadenssystem entwickeln.

Insgesamt besteht das Waffen-System also aus diesen drei Bestandteilen:

- Ein Component, das die Lebensenergie von Objekten verwaltet und sie zerstört, wenn die Lebensenergie auf 0 fällt.
- Es werden Plasmakugeln benötigt, die verschossen werden können und getroffenen Objekten Schaden zufügen.
- Es wird eine Kanone benötigt, welche die Plasmakugeln in Blickrichtung des Spielers verschießt.

Um Arbeit zu sparen, ist es wichtig, bei der Entwicklung von Anfang an darauf zu achten, dass die einzelnen Bestandteile wiederverwendet werden können. Ansonsten müsstest du sie neu schreiben, wenn wir zu den gegnerischen Raumschiffen kommen, die ja ebenfalls in der Lage sein sollen zu schießen.

10.2.4.1 Zerstörbare Objekte

Zuerst kümmern wir uns um das Script, das für die Überwachung der Lebenspunkte zuständig ist. Dieses Script soll man zu einem beliebigen *GameObject* hinzufügen können, um es als zerstörbar zu markieren. Das Script definiert und überwacht dafür die Lebensenergie des *GameObjects* und zerstört es, wenn die Energie auf 0 fällt. Ist das zerstörte Objekt der Spieler gewesen, wird er an seine Ausgangsposition zurückgesetzt, von wo aus er direkt wieder in den Kampf starten kann.

Erzeuge in deinem *MyAssets/Scripts*-Ordner ein neues *C#-Script* mit dem Namen *Destroyable* und ersetze den automatisch generierten Inhalt mit dem Code aus Listing 10.9.

Listing 10.9 Destroyable.cs – gibt GameObjects die „Funktion" zerstört zu werden

```
using UnityEngine;
public class Destroyable : MonoBehaviour {
    public float startHealth = 100; // Lebenspunkte bei Start
    public bool isPlayer = false; // Ist dieses Objekt der Spieler?
    public GameObject prefabCreateOnDestroy; // wird bei tot erzeugt, z.B. Explosion
    private float currentHealth; // Aktuelle Lebenspunkte
    private Vector3 spawnPosition; // Start Position in der Scene
```

Index

Symbole

3D-Modell 66
3D Start Rotation (Particle System) 302
3D Start Size (Particle System) 302

A

Accel (Daydream) 387
AddComponent (Methode) 212
Add Component (Schaltfläche) 53
AddForce 280
AddTorque 280
Advanced Options 255
Agent Type 323
Albedo 252
Albedo Boost 248
Ambient Lighting 233
Anchor 283
Android 73, 346
Android SDK 34, 74
Angular Drag (Rigidbody) 279
Animation 66
Animation Clip 309
Animation Controller 309
Animator Component 309
Animator Controller 422
Animator-Fenster 319
Anti-Aliasing 99
AppButton (Daydream) 387
Applications 25
Apply Root Motion 328
Area Light 239
Arm-Modell 383
Array-Initialisierung 160
Arrays 160, 188, 192
Aspekt Ratio 339
Assets 9, 42, 64
Asset Store 41, 71
Attribute 199
AudioClip 258, 260, 364
Audio Listener 58, 373
AudioListener 258
AudioMixer 258
AudioMixerGroup 260
Audio SDK 149
Audio Source 63, 476
AudioSource 258
Auto Generate 231
Automatische Vervollständigung 177
Auto Random Seed (Particle System) 303
Avatar 365
Avatar (Animation) 309
Awake 200

B

Background Loop 437
Baked GI 224, 247
Baked Global Illumination 224
Baked Indirect 230
Batching 330
Bedingungen 183
Behavior 64, 179
Border 374
Bounces 224
Box Collider 276, 464
Box Projection 246
break 190
Break Force 282
Build 109, 556
Build and Run 110

Bundle Identifier 97
Button 340
ButtonMask 379
Bypass Effects 260
Bypass Listener Effects 260
Bypass Reverb Zones 260
By Reference 166
By Value 166

C

C# 152
Camera 58, 95, 347, 355, 373, 377
Camera Rig 372
Can't add script 218
Canvas 479
Capsule Collider 276, 464
Center 293
CenterEyeAnchor 355
Character Controller 292, 358
Character Joint 284
Clamp (Kurveneditor) 301
ClickButton (Daydream) 386
Collider 61, 83
Collision 202
Collision Detection (Rigidbody) 279
Color 51, 235
Color (Particle System) 301
Color-Picker 51
Component 49, 53, 58, 179
Compress Lightmaps 247
Configurable Joint 284
Connected Anchor 283
Connected Body 282
Console 41, 43, 174, 181
Constant (Particle System) 299
Constrains (Rigidbody) 280
Controller Emulator-App 388
Controller (head) 373
Controller (left) 373
Controller (right) 373
Convex 277, 398
Copy Component 54
Coroutine 213, 483, 489
Culling Mask 236
Curve (Particle System) 300
Cutout 252

D

Datentypen 155
  – konvertieren 157
Daynamic Batching 331
Debug-Log 188
Default State 318
Deklaration 156
Demos 24
Destroy 207, 486
Detail Mask 253
Dictionary-Klasse 193
Directional Light 237
Directional Mode 247
DisableAutoVRCameraTracking 349
disablePositionalTracking 348
Display Resolution Dialog 556
Distance Shadowmask 230
Divide By Zero Exception 218
Dividieren mit Integern 159
Doppler Level 261
Do-While-Schleife 187
Download Archiv 31
Drag & Drop 41
Drag (Rigidbody) 279
Draw Halo 235
Duration (Particle System) 302
Dynamische Datentypen 165

E

Editor 197
Eigenschaft 191
Einfrieren 186
else 186
else if 186
Emission 253
Emission-Modul 303
Emissive 240, 248
Enable Adaptive Resolution 357
Enable Collision 282
Enable Mixed Reality 357
Enable Preprocessing 283
enum 487
Environment Light 246
Environment Lighting 233
Error 181
eulerAngles 208
EVRButtonId 379

Exceptions 174, 217
Experience 25
Exportieren 68
External Camera 371
Extrapolate 279
Eye Level 357

F

Fade 252
Farbräume 232
Field of View 9
Final Gather 247
FindObjectOfType 212
FindObjectsOfType 212
FixedUpdate 201
Flare 235, 248
Floor Level 357
Fog 248
ForceMode 280
For-Schleife 187
forward 208
ForwardDirection 358
Forward Rendering Options 255
FPS 332

G

GameObject 55, 206
Games 25
Game View 41
Gamma Color Space 232
GearVR 72, 97, 105, 352
Geschichte 12
GetComponent 209
GetComponentInChildren 212
GetComponentInParent 212
GetComponents 211
GetComponentsInChildren 212
GetComponentsInParent 212
GetTrackingSpaceType 349
Global Illumination 222, 246
GoogleVR 72, 97
Gradient (Particle System) 301
Gravity 433
Gravity Modifier (Particle System) 302
GvrArmModel 383
GvrArmModelOffsets 382, 384
GvrController 383, 386
GvrControllerMain 382, 389
GvrControllerPointer 385
GvrControllerPrefab 385
GvrEditorEmulator 387
GvrFPSCanvas 385

H

Halo 248
Handbuch 52
Haptics Clips 364
Height Map 253
Heigth 293
Hierarchy 41, 44
Hinge Joint 284

I

IEnummerator 214
If-Abfragen 185
Immersion 114, 119, 135, 137
Importieren 67, 69, 72
Index Out Of Range Exception 218
Index (SteamVR) 375
Indirect Intensity 248
Indirect Multiplier 235
Indirect Resolution 247
Info 181
Initialisierung 156
InputTracking 348
Inspector 41, 48
Instantiate 485
Intensity 235
Interaction System 380
Interne VR-Unterstützung 346
Interpolate 279
Interpolate (Rigidbody) 279
Invalid Cast Exception 218
Inventar 194
isDeviceActive 350
Is Kinematic (Rigidbody) 279
IsTouching (Daydream) 386
Is Trigger 464
isUserPresent 357

J

Java Development Kit 34, 74

K

KI 438
Klammer 153
Klasse 164
Kollision 61
Kommentar 153
Kontextmenü 42
Konvexe Hülle 277
Kurveneditor 444

L

LateUpdate 201
LeftEyeAnchor 355
LeftHandAnchor 355
Licht 92
Lichtberechnung 246
Lichter 59
Light 59, 82
Light Explorer 248
Lighting 224
Lighting Mode 229
Lightmap 82
Lightmap Parameters 248
Lightmapper 247
Lightmapping 231
Lightmap Resolution 247
Lightmap Size 247
Light Probe 240, 248
Light Probe Group 241
Linear Color Space 232
LineRenderer 411
Listen 192
List-Klasse 192
LoadDeviceByName 351
loadedDeviceName 350
LocalAvatar 366
localEulerAngles 208
localPosition 208
localRotation 208
localScale 208
Locomotion 10, 121
Longbow 380
Loop (AudioSource) 260
Looping (Particle System) 302
Loop (Kurveneditor) 300
Luftwiderstand 63

M

Main Maps 252
Mass (Rigidbody) 279
Material 61, 65, 90
Max Distance (AudioSource) 261
Max Distance (Reverb Zone) 264
Max Particles (Particle System) 303
Mecanim 309
Mesh 66
Mesh Collider 276, 398
Mesh Filter 60
Mesh Renderer 60
Metallic 252
Methode 161
Min Distance (AudioSource) 261
Min Distance (Reverb Zone) 264
Min Move Distance 293
Mixed Lighting 225
Mixers (AudioMixer) 269
Mobile Shader 255
Mode 235
Modulo 190
MonoBehavior 200, 202
mp3 258
Multi Pass 347
Mute 260

N

Namespaces 171, 348
NavMesh 323
NavMeshAgent 323, 327, 425
NavMeshLinks 325 f.
NavMeshObstacle 326
NavMeshSurface 325
new-Schlüsselwort 160
Normal Map 252
Null Reference Exception 217

O

Occlusion Culling 476
Occlusion Map 253
Oculus SDK 347, 492
ogg 258
OnApplicationQuit 202
OnClick 340

OnCollisionEnter 286, 406
OnCollisionExit 286
OnCollisionStay 286
OnDestroy 202
Opaque 251
OpenVR 347, 368
Orientation (Daydream) 387
Osig 105
Output (AudioSource) 260
OverlapSphere 482
override 170
OVR Camera Rig 356
OVRCameraRig 355, 357, 366
OVRCubemapCaptureProbe 360
OVR Debug Info 360
OVRHapticsClip 364
OVRHaptics.LeftChannel 365
OVRInput 361, 492
OVRInput.Axis1D 361
OVRInput.Axis2D 361
OVRInput.Button 361
OVRInput.NearTouch 363
OVRInput.Touch 363
OVR Manager 356
OVR Player Controller 359
OVRPlayerController 357
OVRPlugins 353
OVR Scene Sample Controller 360

P

Parameter 162
parent 208
Parenting 45
Particles (Particle System) 298
ParticleSystem 446
Paste Component 54
PBR 250
PC 346
Physically Based Rendering 250
Physically Based Shader 250
Physic Glitches 282
PhysicsManager 433
Ping Pong (Kurveneditor) 300
Pitch (AudioSource) 260
PlatformUIConfirmQuit 357
PlatformUIGlobalMenu 357
Play Area 368, 374
Playback Speed (Particle System) 298
Playback Time (Particle System) 298
PlayClipAtPoint 406
PlayerPrefs 329
Player Settings 97
Play On Awake 260
Play On Awake (Particle System) 303
Point Light 236
Polymorphie 170
position 207
Preempt 365
Prefab 57, 87, 206
Presence 10
Prewarm (Particle System) 302
Priority (AudioSource) 260
private 173
Progressive Lightmapper 247
Project Browser 41 f.
Properties 191
public 173

Q

Qualitätsstufe 99
Quality Settings 100

R

Radius 293
Random Between Two Colors (Particle System) 301
Random Between Two Constants (Particle System) 300
Random Between Two Curves (Particle System) 300
Random Between Two Gradients (Particle System) 301
Randomize Rotation (Particle System) 302
Range 236, 238
Raycasts 482
Ready Only 191
Realtime 224
Realtime GI 225, 247
Realtime Global Illumination 225
Recenter 10, 348, 357
Rechen- und String-Operatoren 158
Reflection Probe 245, 248
refreshRate 349
Remove Component 54
Renderer-Modul 304
Rendering Mode 251

Render Mode 236
Render Scale 350 f., 357
renderViewportScale 350
Reset Tracker On Load 357
return 162
Reverb Preset 265
Reverb Zone Mix 260
RightEyeAnchor 355
RightHandAnchor 355
Rigidbody 62, 401, 403, 433
Rolloff Mode 262
Room-Scale 26, 349, 357
rotation 208
Rückgabetypen 162

S

Save 329
Scaling Mode (Particle System) 303
Scene 54
Scenes in Build 109
Scene View 41, 44, 46
Schatten 60
Schleifen 186
Schwerkraft 63, 433
Screen Space 133
Script 64, 179
SDK 345
Seated 25
Secondary Maps 254
Semikolon 153
SetControllerVibration 364
SetFloat 329
SetInt 329
SetPass-Call 347
SetString 329
SetTrackingSpaceType 349
Shader 65
Shadow 61
Shadowmask 230
Shadow Type 235
Shape-Modul 303
showDeviceView 351
Sichtbarkeiten 173
Simulation Space (Particle System) 302
Simulation Speed (Particle System) 303
Single Pass 347
Skin Width 293
Skybox 246, 431
Skybox Material 246
Slope Limit 293
Snapping 48, 123
Snapshots (AudioMixer) 268
Sound 146
Sound Design 146
Spatial Blend 260
Spatial Sound 146
Speichern 329
Sphere Collider 276
Spot Angle 238
Spot Cookie 248
Spot Light 238
Spread (AudioSource) 261
Spring Joint 283
Sprite 65, 341
Standard Shader 240, 243, 249f.
Standing 26
Start 200
Start Color (Particle System) 302
StartCoroutine 214
Start Delay (Particle System) 302
Start Lifetime (Particle System) 302
Start Rotation (Particle System) 302
Start Size (Particle System) 302
Start Speed (Particle System) 302
Static Batching 331
Statische Methoden 167
Statische Variablen 167
Stats-Anzeige 331
SteamVR 371
SteamVR_Camera 377
SteamVR_Controller 378
SteamVR_Controller.Input(id).GetAxis 379
SteamVR_Controller.Input(id).GetPress 378
SteamVR_Controller.Input(id).GetTouch 378
SteamVR_ControllerManager 373, 378
SteamVR_PlayArea 374
SteamVR_Renderer 371
SteamVR_RenderModel 373, 376
SteamVR_TrackedObject 373, 375
Step Offset 293
Stereo Pan 260
Stereo Rendering Mode 347
String-Operatoren 160
Subtractive 231
Sun Source 246
supportedDevices 351

T

Tag 205, 465
Terrain 306, 396
Testen 101
Texture 64
Time.deltaTime 209
Toolbar 41
TouchPos (Daydream) 386
TrackerAnchor 355
Tracking Origin Type 357
Tracking Space 368, 372, 374
TrackingSpaceType 349
Trail Renderer 403 f.
Transform 207
Transform-Tools 47
Transparent 251
Trigger 202
Trigger-Collider 419
TriggerHapticPulse 379
Type 235

U

Unable to merge android manifests 105
Uncanny Valley 142
UnityEngine.VR 348
UnityEvent 484
Unity ID 33
Unity Packages 68
Unity Profiler 331
Unity-Toolbar 41
Unity, VR-Unterstützung 96
up 208
Update 201
Use Fixed Update for Tracking 356
Use Gravity (Rigidbody) 279
Use per Eye Cameras 356
Use Recommended MSAA Level 357
User Interface 65, 133, 479
using-Anweisungen 171

V

Variablen 154
Vektor 55
Vererbung 168, 413
Vergleichsoperatoren 184
Verweis 178, 204
Verzweigungen 183
Views (AudioMixer) 269
virtual 169
Virtual Reality SDKs 347
Virtual Reality Supported 346
Visual Studio 175
void 162
Volume (AudioSource) 260
VRDevice 349
VRSettings 350

W

WaitForSeconds 214
Warning 181
wav 258
While-Schleife 187
World Space 135

Y

yield return null 214

Z

Zerstören 207
Zielplattform 73