Unreal Engine 4 Shaders and Effects Cookbook

Over 70 recipes for mastering post-processing effects and advanced shading techniques

Packt

www.packt.com

Brais Brenlla Ramos and John P. Doran

Unreal Engine 4 Shaders and Effects Cookbook

Over 70 recipes for mastering post-processing effects and advanced shading techniques

Brais Brenlla Ramos
John P. Doran

Packt

BIRMINGHAM - MUMBAI

Unreal Engine 4 Shaders and Effects Cookbook

Commissioning Editor: Kunal Chaudhari
Acquisition Editor: Trusha Shriyan
Content Development Editor: Pranay Fereira
Technical Editor: Diksha Wakode
Copy Editor: Safis Editing
Project Coordinator: Kinjal Bari
Proofreader: Safis Editing
Indexer: Tejal Daruwale Soni
Graphics: Alishon Mendonsa
Production Coordinator: Deepika Naik

First published: May 2019

Production reference: 1240519

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-78953-854-0

`www.packtpub.com`

Contributors

About the authors

Brais Brenlla Ramos is a passionate Architect, 3D artist, Unreal Engine 4 developer and first-time author based between A Coruña and his place of work in London, UK. His passion for all things 3D-related dates back to when he was playing games as a child, experiences that fuelled his later studies in architecture and computer animation. His entrance into the professional 3D world happened at the same time as his studies were finishing, with initial projects undertaken in the field of architectural visualization for different studios. Since then, he's worked on many different 3D modeling and app development projects, first as a team member, and later as the Unreal Engine 4 lead developer at a company called AccuCities, based in London.

To my friends and family, who got me this far; and to my partner, Tamy, whose support and love carried me throughout.

John P. Doran is a passionate and seasoned technical game designer, software engineer, and author based in Peoria, Illinois.

For over a decade, John has gained extensive hands-on expertise in game development, working in a variety of roles, ranging from game designer to lead UI programmer. Additionally, John has worked in game development education teaching in Singapore, South Korea, and the United States. To date, he has authored over 10 books pertaining to game development.

John is currently an instructor in residence at Bradley University. Prior to his present ventures, he was an award-winning videographer.

I want to thank my wife, Hien, for all of her support over the course of working on this book.

About the reviewer

Deepak Jadhav is a game developer based in Pune, India. Deepak received his bachelor's degree in computer technology and master's degree in game programming and project management. Currently, he is working as a game developer in India's leading game development company. He has been involved in developing games on multiple platforms, such as PC, Mac, and mobile. With years of experience in game development, he has a strong background in C# and C++, and he has developed his skills in platforms including Unity, Unreal Engine, and augmented and virtual reality.

I would like to thank the authors and the Packt Publishing team for giving me the opportunity to review this book.

Table of Contents

Preface 1

Chapter 1: Physically Based Rendering 6
- **Introduction** 6
- **Setting up a studio scene** 8
  - Getting ready 8
  - How to do it... 9
  - How it works... 14
- **Working inside the material editor** 14
  - Getting ready 15
  - How to do it... 15
  - How it works... 22
- **Our first physically based material** 23
  - Getting ready 23
  - How to do it... 24
  - How it works... 31
- **Creating some simple glass with the translucent blend mode** 32
  - Getting ready 32
  - How to do it... 33
  - How it works... 37
- **Lighting our scene with image-based lighting** 38
  - Getting ready 39
  - How to do it... 39
  - How it works... 45
- **Checking the cost of our materials** 47
  - Getting ready 48
  - How to do it... 48
  - How it works... 54

Chapter 2: Post-Processing Effects 56
- **Introduction** 56
- **Using a post-process volume** 57
  - Getting ready 57
  - How to do it... 59
  - How it works... 63
  - See also 64
- **Changing the mood of a scene through color grading** 64
  - Getting ready 65
  - How to do it... 65
  - How it works... 72

There's more... 73
See also 75
Setting up a cinematic shot using depth of field 75
Getting ready 75
How to do it... 76
How it works... 81
There's more... 82
See also 82
Applying cinematic effects to our games 83
Getting ready 83
How to do it... 83
How it works... 89
There's more... 89
See also 90
Mimicking a real-life camera using Bloom and Lens Flares 90
Getting ready 90
How to do it... 91
How it works... 97
There's more... 98
See also 99
A horror movie pulsating effect with post process materials 99
Getting ready 99
How to do it... 100
How it works... 108
There's more... 109
See also 110
Adjusting anti aliasing and other rendering features 110
Getting ready 111
How to do it... 111
How it works... 117
There's more... 118
See also 119

Chapter 3: Opaque Materials and Texture Mapping 120
Introduction 121
Using masks within a material 122
Getting ready 122
How to do it... 123
How it works... 129
There's more... 130
See also 131
Instancing a material 131
Getting ready 131
How to do it... 132
How it works... 136

There's more... 138
See also 139
Texturing a small prop 139
Getting ready 140
How to do it... 140
How it works... 146
There's more... 146
See also 147
A plastic cloth using Fresnel and detail texturing 147
Getting ready 148
How to do it... 148
How it works... 152
There's more... 153
See also 154
Creating a semi procedural material 155
Getting ready 155
How to do it... 155
How it works... 161
There's more... 161
See also 162
Baking out a material 162
Getting ready 162
How to do it... 163
How it works... 168
There's more... 169
See also 169
Distance-based texture blending 170
Getting ready 170
How to do it... 171
How it works... 174
There's more... 175
See also 177
Chapter 4: Translucent Materials and More 178
Introduction 179
Creating a candle material with SSS 179
Getting ready 180
How to do it... 180
How it works... 187
There's more... 189
See also 189
Setting up a truly transparent glass 189
Getting ready 190
How to do it... 190
How it works... 196

There's more... 197
See also 197
A different type of translucency – holograms 198
Getting ready 198
How to do it... 199
How it works... 207
There's more... 208
See also 208
Achieving realistic reflections 208
Getting ready 209
How to do it... 210
How it works... 220
There's more... 221
See also 221
Mastering refraction by creating a pool water material 221
Getting ready 222
How to do it... 223
How it works... 232
There's more... 233
See also 233
Water caustics 233
Getting ready 234
How to do it... 235
How it works... 243
There's more... 244
See also 244
Animating a sea shader 245
Getting ready 245
How to do it... 246
How it works... 257
There's more... 258
See also 258
Chapter 5: Beyond Traditional Material Uses 259
Introduction 259
Using an emissive material to light the scene 260
Getting ready 261
How to do it... 261
How it works... 268
There's more... 269
See also 270
Playing a video from the internet on a screen 270
Getting ready 271
How to do it... 271
How it works... 276

There's more... 277
See also 277
Creating a CCTV camera feed 278
Getting ready 278
How to do it... 279
How it works... 284
There's more... 284
See also 286
Highlighting interactive elements within our game 286
Getting ready 287
How to do it... 287
How it works... 297
There's more... 298
See also 298
Creating a game compass 299
Getting ready 299
How to do it... 300
How it works... 307
There's more... 308
See also 311
Creating a mini map 312
Getting ready 312
How to do it... 313
How it works... 321
There's more... 322
See also 322
Chapter 6: Advanced Material Techniques 323
Introduction 324
Painting a mesh with vertex painting 324
Getting ready 325
How to do it... 325
How it works... 330
There's more... 331
See also 332
Using decals to add granularity to our scenes 333
Getting ready 333
How to do it... 334
How it works... 338
There's more... 339
See also 340
Creating a brick wall with Parallax Occlusion Mapping 340
Getting ready 341
How to do it... 341
How it works... 347

There's more... 348
See also 349
A brick wall using displacement 349
Getting ready 350
How to do it... 350
How it works... 354
There's more... 354
See also 355
Proximity-based masking with mesh distance fields 355
Getting ready 356
How to do it... 356
How it works... 360
There's more... 362
See also 362
Chapter 7: Using Material Instances 363
Introduction 363
Creating snow on top of objects using layered materials 364
Getting ready... 364
How to do it... 365
How it works... 379
Changing from a sunny scene to a snowy one through parameter collection 380
Getting ready... 380
How to do it... 380
How it works... 384
Changing between seasons quickly with curve atlases 385
Getting ready... 385
How to do it... 385
How it works... 396
Blending landscape materials 397
Getting ready... 397
How to do it... 398
How it works... 404
Customizing UVs 405
Getting ready... 405
How to do it... 405
How it works... 411
Chapter 8: Mobile Shaders and Material Optimization 412
Introduction 412
Creating materials for mobile platforms 413
Getting ready... 413
How to do it... 413
How it works... 415
There's more... 416

See also... 417
Using the forward shading renderer for VR 417
Getting ready... 417
How to do it... 418
How it works... 421
See also... 421
Optimizing through texture atlases 421
Getting ready... 422
How to do it... 422
How it works... 427
Baking a 3D model material into a texture 428
Getting ready... 429
How to do it... 430
How it works... 435
Combining multiple meshes with the HLOD tool 436
Getting ready... 437
How to do it... 438
How it works... 445
General material-optimization techniques 446
Getting ready... 446
How to do it... 447
How it works... 453
Chapter 9: Some Extra Useful Nodes 454
Introduction 455
Adding randomness to identical models 455
Getting ready 456
How to do it... 456
How it works... 461
There's more... 461
See also 462
Adding dirt to occluded areas 462
Getting ready 462
How to do it... 463
How it works... 466
There's more... 467
See also 467
Matching texture coordinates across multiple meshes 467
Getting ready 468
How to do it... 468
How it works... 473
There's more... 474
See also 474
Adjusting material complexity through quality switches 475
Getting ready 475

How to do it... 476
How it works... 481
There's more... 481
See also 482
Using interior cubemaps to texture the interior of a building 482
Getting ready 483
How to do it... 483
How it works... 486
There's more... 487
See also 488
Using fully procedural noise patterns 489
Getting ready 489
How to do it... 490
How it works... 494
There's more... 495
See also 496
Other Books You May Enjoy 497
Index 500

Preface

Unreal Engine 4 Shaders and Effects Cookbook aims to take you on a journey of creation and discovery within the Unreal Engine 4 game engine. As the title of the book implies, we'll travel hand in hand to every corner of the engine, performing actions that affect the visuals of our games and apps. We'll do so in an orderly way, starting from the very beginning by covering fundamental topics that will stay with us throughout the rest of the book. Each chapter that follows will expand upon that base, allowing for a gentle progression curve that will allow almost any user to follow along. In spite of that, each entry – or recipe – has been also conceived as an independent unit, letting you tackle it separately from the others in case you are already proficient with the other topics.

We'll start by covering the core concepts behind Unreal Engine's rendering pipeline, such as its physically based rendering approach and post-processing effects. With solid foundational knowledge about those two topics, we'll expand upon them and study different types of materials: opaque ones, translucent ones, and more, such as the different subsurface materials and other shading models. We'll also explore several advanced material creation techniques and tricks that the engine lets us use to create multiple different effects—from mixing materials and blueprints, to instancing and material optimization. There's a whole lot we are going to be covering!

Upon finishing this book, you will have a thorough knowledge about many different material concepts and techniques, both from a practical and a theoretical point of view. You'll be able to use these newly learned concepts in any games, apps, or personal projects that you tackle, with the absolute confidence that you are doing it right. With that said, let's get to it!

Who this book is for

Unreal Engine 4 Shaders and Effects Cookbook benefits from a structure that goes *in crescendo*, covering more difficult topics as we move along together. Thus, the book lends itself to being read by multiple different profiles of user from novice users, to more seasoned ones that haven't yet touched Unreal's material pipeline. Whatever the case, a good understanding of Unreal is definitely a plus, and something that will make your journey throughout this book a much smoother experience.

What this book covers

`Chapter 1`, *Physically Based Rendering*, starts off this book by going over the fundamental rendering concepts that Unreal relies on, as well as introducing us to the material editor.

`Chapter 2`, *Post-Processing Effects*, introduces the user to the powerful concept of post-processing in Unreal and explains the different effects that can be achieved through it.

`Chapter 3`, *Opaque Materials and Texture Mapping*, goes into detail about one of the most common type of materials in Unreal and the different uses that it has.

`Chapter 4`, *Translucent Materials and More*, covers one of the most exciting type of materials, the translucent ones, as well as many others, including subsurface and emissive materials.

`Chapter 5`, *Beyond Traditional Material Uses*, goes over different uses that materials can have beyond simply being applied to 3D models, including light functions, UI elements, and displaying videos.

`Chapter 6`, *Advanced Material Techniques*, talks about some of the most high-end effects that can be created within the material editor by using advanced techniques, such as parallax occlusion mapping and mesh distance fields.

`Chapter 7`, *Using Material Instances*, discusses how to use the concept of instancing to quickly make tweaks to a material instance, layer different shaders on top of each other, and affect multiple material settings at once.

`Chapter 8`, *Mobile Shaders and Material Optimization*, goes over various ways to optimize your materials to make them more performant on different hardware where efficiency is important, such as on mobile devices or when working in virtual reality.

`Chapter 9`, *Some Extra Useful Nodes*, focuses on some of the most useful nodes we can find within Unreal that don't really belong to a collective category of their own.

To get the most out of this book

Any reader will need to have installed a version of Unreal Engine on their computers; the latest version, if possible. Most of the recipes we'll look at should work on different engine versions, but we recommend 4.22 in order to have the latest features installed.

Prior knowledge about the engine is not a must, but having some working experience with Unreal will help the reader enjoy a smoother experience throughout the book. Whilst no coding skills are required, some fluency with the Blueprint visual scripting language would also be of great help.

Download the example code files

You can download the example code files for this book from your account at `www.packt.com`. If you purchased this book elsewhere, you can visit `www.packt.com/support` and register to have the files emailed directly to you.

You can download the code files by following these steps:

1. Log in or register at `www.packt.com`.
2. Select the **SUPPORT** tab.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box and follow the onscreen instructions.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR/7-Zip for Windows
- Zipeg/iZip/UnRarX for Mac
- 7-Zip/PeaZip for Linux

The code bundle for the book is also hosted on GitHub at `https://github.com/PacktPublishing/Unreal-Engine-4-Shaders-and-Effects-Cookbook`. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: `https://www.packtpub.com/sites/default/files/downloads/9781789538540_ColorImages.pdf`.

Conventions used

There are a number of text conventions used throughout this book.

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "Add a `Cheap Contrast` node after the Texture Sample, and connect its In (S) input pin with the output of the previous image"

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "Drag a cable out of the original **Texture Sample** and create a new **Multiply** node"

Warnings or important notes appear like this.

Tips and tricks appear like this.

Getting ready

This section tells you what to expect in the recipe and describes how to set up any software or any preliminary settings required for the recipe.

How to do it…

This section contains the steps required to follow the recipe.

How it works…

This section usually consists of a detailed explanation of what happened in the previous section.

There's more…

This section consists of additional information about the recipe in order to make you more knowledgeable about the recipe.

See also

This section provides helpful links to other useful information for the recipe.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at `customercare@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packt.com/submit-errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit `packt.com`.

1
Physically Based Rendering

Welcome to the first chapter of the book! In the next few pages, we are going to start looking at how to set up a scene in Unreal for visualization purposes—we want to make sure that we nail this first part down before we move any further. Beginner or advanced, no matter what type of user you are, we'll need to make sure to take a look at some of the most critical elements that can make or break a scene in Unreal. Things like taking advantage of the right type of lighting, knowing where to look for the most common material parameters, or learning to measure the impact in performance that the shaders have are vital in any project. With that in mind, we are going to be learning about the following topics:

- Setting up a studio scene
- Working inside the material editor
- Our first physically based material
- Creating some simple glass with the translucent blend mode
- Lighting our scene with image-based lighting
- Checking the cost of our materials

Introduction

Welcome to this in-depth journey through the material creation process in Unreal Engine 4! I think you are going to have a great time if you are excited about the possibilities that this game engine brings to the table in terms of state-of-the-art rendering techniques. And by state-of-the-art I mean a powerful and robust rendering pipeline, where both photorealistic and stylized game art are possible without changing to a different development suite.

The fact that such a flexible system is in place is courtesy of the continuous advances over the years in the field of real-time rendering. We've journeyed from the 2D era into the 3D era, from sprites and flat images to the rendering of polygons and whole worlds.

Each of these changes happened thanks to a combination of new and more powerful hardware as well as increasingly intelligent rendering pipelines and techniques. One of the latest improvements that we can talk about is what we are going to be covering throughout this book—the PBR workflow.

And what does PBR stand for? That would be **Physically Based Rendering**—a particular method that takes into account how light behaves when it comes into contact with 3D objects. In order to represent materials placed in a 3D environment, artists need to specify certain properties for each of the materials that they create—such as what the underlying color should be, how much light they reflect, or how defined those reflections are.

This is significant change from previous workflows, where light propagation and its simulation wasn't taken into account in a realistic way. This meant, for example, that materials couldn't be replicated under different lighting conditions—having, for instance, a night and a day scene using the same assets resulted in them looking substantially different. An artist would therefore need to create different sets of textures or adjust the materials to make them look right for each particular scenario they might be in.

This has changed with the recent introduction of the PBR workflow. Newer game engines, such as Unreal Engine 4, have made this rendering approach their quasi default one—and I say *quasi* as they also allow for older rendering methods to be thrown into the mix in order to give artists more freedom. Materials are coherent under different lighting settings, and knowing how to create content under this pipeline ensures usability under a lot of different circumstances.

However, PBR is not a universally defined convention as far as its implementation goes. This means that how things work under the hood varies across the different rendering engines. The exact implementation that Epic has chosen for their Unreal Engine platform is different from that of other third-party software creators. Furthermore, PBR workflows in real-time applications are slightly different to offline renderers, as efficiency and speed are a must in this industry and things have to be adapted consequently. What we need to take away from these facts is that a physically based approach to rendering has huge advantages (as well as some limitations) that we as artists need to be aware of if we are to use the engine to its full potential.

We conceived the present book with that goal in mind. We aim to present you with a series of recipes that tackle many different functionalities within Unreal, structured in a way where each unit can be read independently from the rest. In order to do so, we'll be taking a look in the following pages at how to get a hold of the engine and how to set up a basic scene, which we'll use to visualize our projects.

Setting up a studio scene

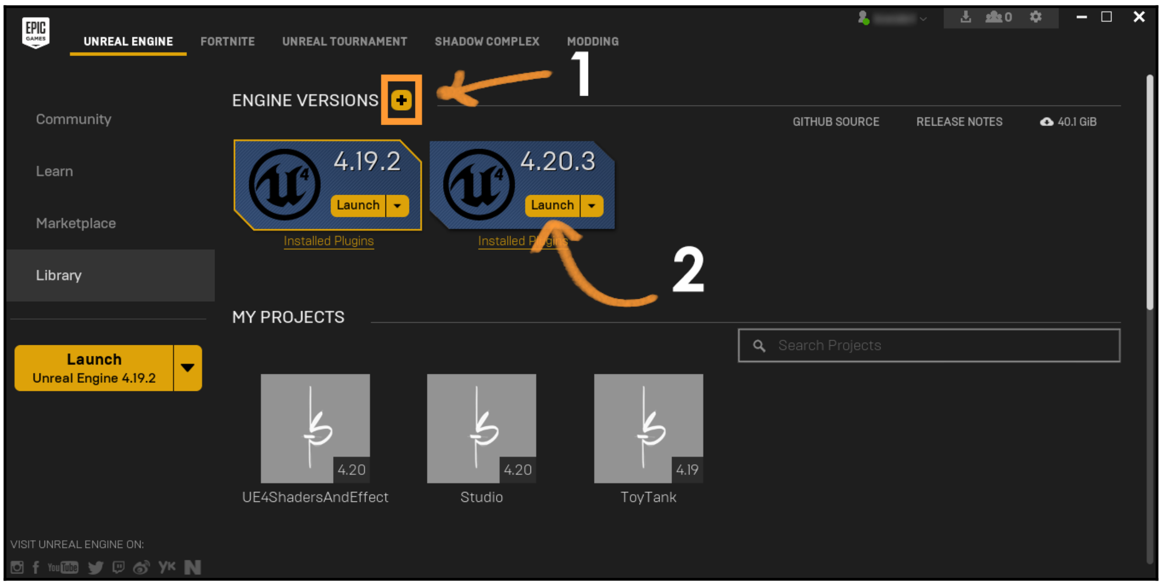
In this first recipe, we are going to create a basic scene that we'll be able to use as our background level throughout this course. This initial step is here just so we can go over the basics of the engine and get familiar with different useful websites from where we can download multiple assets.

Getting ready

Before we actually start creating our basic studio scene, we will need to download Unreal Engine 4. I've started writing this book with version 4.20.3, but don't hesitate to use the latest version at the time of reading.

Here's how you can download it:

1. Get the **Epic Games Launcher** from the engine's website, `https://www.unrealengine.com/en-US/blog`, and follow the installation procedure indicated there.
2. Once installed, download the latest version of the engine. We can do so by navigating to the **UNREAL ENGINE** section of the launcher, in the tab named **Library**. In there, we'll be able to see a **+** icon (**1**), which lets us download whichever version of Unreal we want. Once we've downloaded it, launch it (**2**) so we can get started:

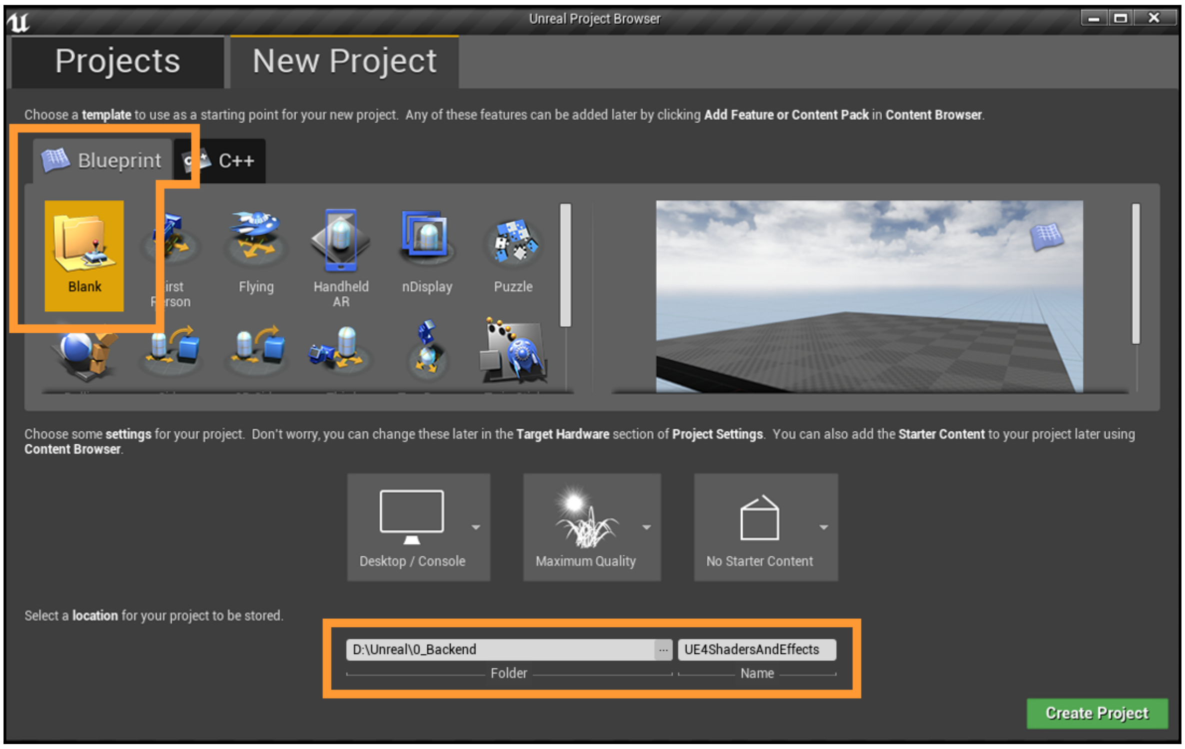


And that's all you need! We now have everything required to get started in Unreal Engine 4. How cool is that? A whole new game engine at our fingertips, completely free, and with a variety of tools within it that would take years to learn and master. It really is a thing of wonder! Next up, we are going to start learning about one of those tools—the materials. And in order to do so, let's start by creating our first project!

How to do it...

Let's start by launching the engine that we have just installed and creating a new project by taking the following steps:

1. Create a **New Project**—give it a name and select the folder where you want it to live. Just as a reference, as shown in the following screenshot, I've decided to start off with a blank blueprint-based project, but it doesn't really matter what we decide to initially include. Nothing special so far! You can choose to add the **Starter Content** if you want, as it comes with several useful resources that we can use later on:



Choose a **template** to use as a starting point for your new project. Any of these features can be added later by clicking **Add Feature or Content Pack** in **Content Browser**.

Choose some **settings** for your project. Don't worry, you can change these later in the **Target Hardware** section of **Project Settings**. You can also add the **Starter Content** to your project later using **Content Browser**.

Desktop / Console | Maximum Quality | No Starter Content

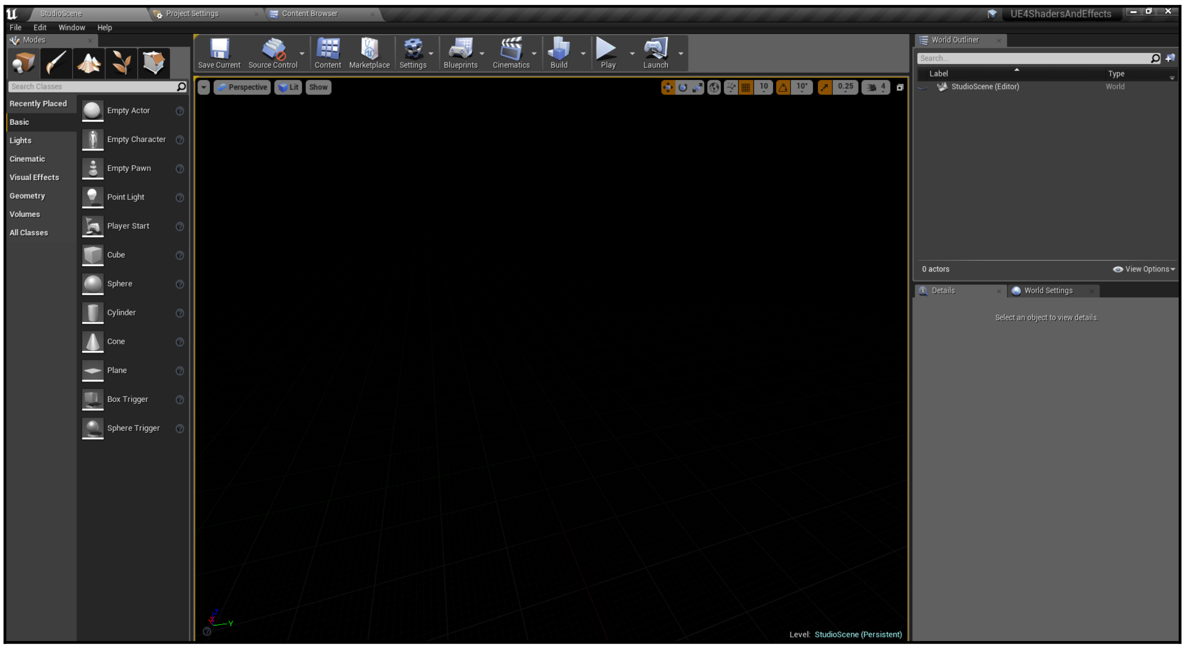
Select a **location** for your project to be stored.

Folder: D:\Unreal\0_Backend
Name: UE4ShadersAndEffects

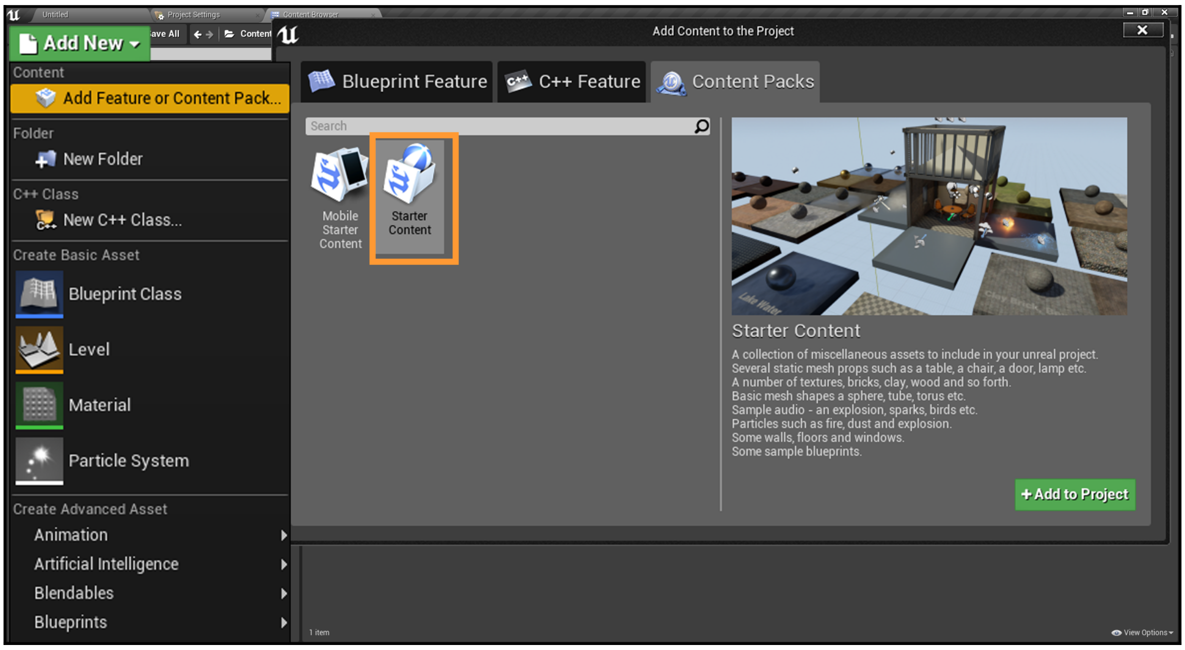
Additionally, you can get more free resources from other different places. You can check the **Learn** tab within the **Epic Games Launcher** to see what freely available examples you can get a hold of, or check the community section to see if there is any new cool content.

Epic has recently collaborated with multiple content creators to make a multitude of different assets available to anyone using Unreal, and you can check them out at the following website: `https://www.unrealengine.com/en-US/blog/new-free-content-coming-to-the-unreal-engine-marketplace?utm_source=launcher utm_medium=chromiumutm_term=forumutm_content=FreeContentutm_campaign=communitytab`.

2. The first thing that we need to do once the editor loads is to go to **File** | **Save Current As**, just to make sure that the changes we are about to implement get saved. Otherwise, we would just be working on the default untitled map, which wouldn't store any of the changes that we are about to make!
3. Once that's done, we are now ready to start spicing things up. Erase everything from the world outliner—we are not going to be using any of that for our studio scene. Your scene and the world outliner should look something like this:



4. If you haven't done so before, it is now time to include the **Starter Content**. Don't worry if you didn't do it at first! I didn't say it was mandatory only to be able to look at how to include it after starting a new project—just navigate to the content browser and look for the **Add New** option in the upper left corner. Select the first available option in there, named **Add feature or Content Pack**, as shown in the following screenshot:



Starter Content

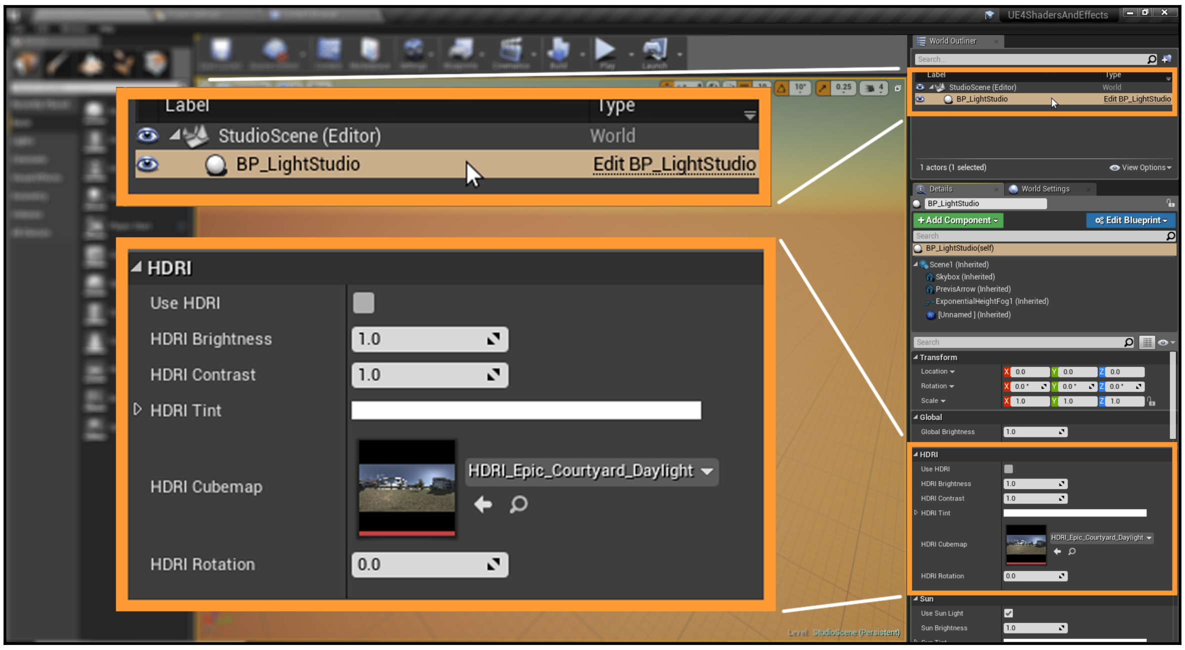
A collection of miscellaneous assets to include in your unreal project.
Several static mesh props such as a table, a chair, a door, lamp etc.
A number of textures, bricks, clay, wood and so forth.
Basic mesh shapes a sphere, tube, torus etc.
Sample audio - an explosion, sparks, birds etc.
Particles such as fire, dust and explosion.
Some walls, floors and windows.
Some sample blueprints.

5. With that included, we can see that the Starter Content includes a blueprint that can be quite useful for setting up the lighting in our scene. You can look for this inside of the **Content Browser** | **Starter Content** | **Blueprints** folder, and it's named **BP_ Light Studio**. Select it and drag it into the scene we have previously created.

The asset called **BP_Light Studio** is a blueprint that Epic Games has already created for us. It includes several lighting settings that will make our lives easier—instead of having to set up multiple lights and assign them different values, it automates all of that work for us so we just have to choose how we want our scene to look. Making a simple studio scene will be something very easy to do this way.

Retaining that level of control over which lights are placed and how we do that is, of course, very important, and something that we'll do later in the book, but for now this is a very powerful tool that we will use.

6. With the **BP_ Light Studio** placed in our scene, we can start tweaking its default values just so we can use it as a lighting studio setup. Select the blueprint from the world outliner and let's tweak several settings.
7. The first one we can look at is the **HDRi** tab inside the details panel for the **BP_ Light Studio**. **HDRi** is short for **High Dynamic Range imaging**, which is a type of texture that stores the lighting information from the place at which the photo was taken. Using that data as a type of light in 3D scenes is a very powerful technique, which makes our environments look more natural and real:



Label	Type
StudioScene (Editor)	World
BP_LightStudio	Edit BP_LightStudio

HDRI

Use HDRI	☐
HDRI Brightness	1.0
HDRI Contrast	1.0
HDRI Tint	
HDRI Cubemap	HDRI_Epic_Courtyard_Daylight
HDRI Rotation	0.0

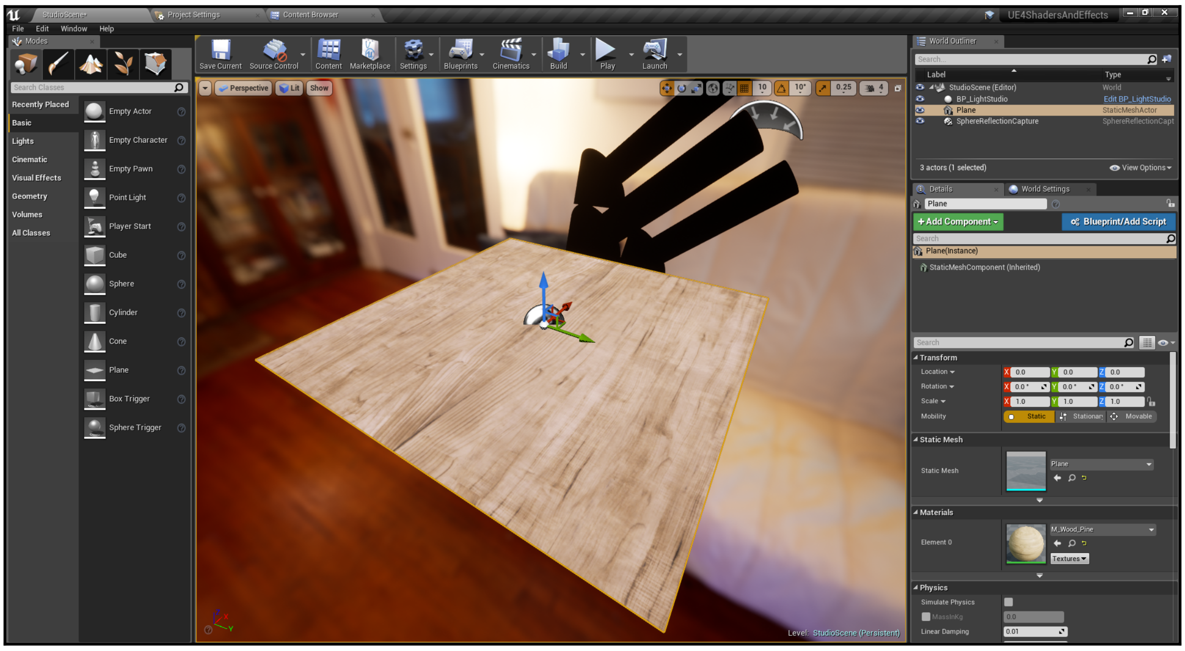
8. However, useful HDRi might be, this lighting method is turned off by default, so make sure to tick the **Use HDRi** checkbox. That will make the texture placed in the **HDRi Cubemap** slot light the scene. Feel free to use any other ones you might have or download one to use throughout the project!

HDRi images are very useful for 3D artists, even though they can be tricky to create as it is usually a lengthy process. There are many websites from which you can buy them, but I like the following one that gives you free access to some very useful ones: `http://www.hdrlabs.com/sibl/archive.html`.

We will be using the one called **Alexs Apartment**, which is quite useful for interior visualization.

9. You can now untick the **Use Light Sun** and the **Use Atmosphere** option found under the **Sun** and the **Atmosphere** section of the **BP_LightStudio** blueprint if you use an HDRi image. As we said earlier, this type of picture stores lighting information, which renders the use of other lights sometimes optional.
10. Once you've done that, let's create a basic plane on which we can use to lay out our objects. Dragging a plane into the scene from the **Modes** panel will do the job: **Modes** | **Basic category** | **Plane**.
11. Let's assign our newly placed plane an interesting default material so we have something to look at—with the plane selected, scroll down to the **Materials** section of the details panel and change its default value to **M_Wood_Pine**. Said material is part of the **Starter Content**, so make sure you have it installed!

We should now be looking at something like the following:



With that out of the way, we can say that we've finished creating our basic studio scene. Having done that will enable us to use this level for visualization purposes, kind of like having a white canvas on which to paint. We will use this to place other models and materials as we create them, in order to correctly visualize our assets.

How it works...

There are at least two different objectives that we can complete if we follow the previous set of steps—the creation of our intro scene being the first one and the second one being getting familiar with the engine. This final task is something that will continue to happen over time—but getting our hands dirty now will have hopefully accelerated that process.

Something that could also speed that up even more is a review process of what we've just done. Not only will we learn things potentially faster, but knowing why we do the things the way we do them will help us cement the knowledge we acquire—so expect to see a *How it works...* section after each recipe we tackle! As the first ever example of the aforementioned section, we'll briefly go over what we have just done before in order to understand how things work in Unreal.

The first step we've taken was to actually create the Unreal Engine project on which we'll be working throughout this book. We've then added the assets present in the Starter Content package that Epic Games supplies, as it contains useful 3D models and materials that we can check later on as we work on other recipes. The most important bit we've done was probably the lighting setup though, as this will be the basis of some of the next recipes. This is because having a light source is vital to visualizing the different assets that we create or add to the scene. Lighting is something that we'll explore more in some of the next recipes, but the method we've chosen in this one is a very cool technique that you can use in your own projects. We are using an asset that Unreal calls a **blueprint**, something that allows you to use the engine's visual scripting language to create different functionalities within the game engine without using C++ code. This is extremely useful, as you can program different behaviors across multiple types of actors to use to your advantage—turning a light on and off, opening a door, creating triggers to fire certain events, and so on. We'll explore them more as we go along, but at the moment we are just using an already available one to specify the lighting effects we want to have in our scene. This is in itself a good example of what a blueprint can do, as it allows us to set up multiple different components without having to specify each one of them individually—such as the HDRi image, the sun position, and others that you can see if you look at the **Details** panel.

Working inside the material editor

Let's get started with the material editor! This is the place where the magic will happen and also where we'll spend most of our time during this cookbook. Better get well acquainted with it then! As with everything inside Unreal, you'll be able to see that this space for creating materials is a very flexible one—full of customizable panels, rearrangeable windows, and expandable areas. You can place them however you want!

Because of its modular nature, some of the initial questions we need to tackle are the following ones: how do we start creating materials and where do we look for the most commonly used parameters? Having different panels means having to look for different functionalities in each of them, so we'll need to know how to find our way around the editor. We won't stop there though—the editor is packed with plenty of useful little tools that will make our jobs as material creators that much easier, and knowing where they live is one of the first mandatory steps.

So, without further ado, let's use the project we have already set up in the previous recipe as our starting point and let's start creating our first material!

Getting ready

There's not much we need to do at this point—all thanks to having previously created the basic blank project. That's the reason we created it in the first place, so we can start working on our materials straight away. Having set up the studio scene is all we need at this point.

In spite of this, don't feel obliged to use the level we created in the first recipe. Any other one will do, as long as there are some lights in it that help you visualize your world. That's the advantage of the PBR workflow, that whatever we create following its principles will work across different lighting scenarios. Let's jump right in!

How to do it...

It's now time to take a look at how the material editor works, at the same time as we create our first material. This editor includes many different tools and functionalities within it, so there are plenty of things to take a look at!

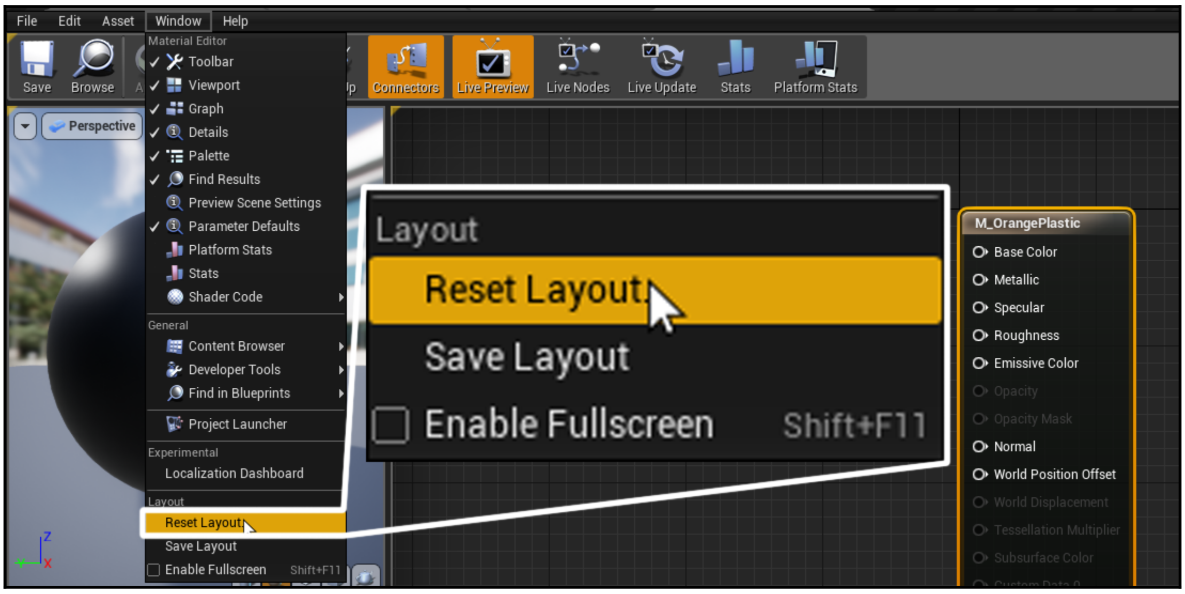
Remember that you can bring the material editor up by just creating a new material and double-clicking on it.

The first important thing we will be doing is to actually create a material. Of course, this is a very trivial action and there's not much to explain—just right-click anywhere on the content browser and select the **Create Basic Asset | Material** option. What is important is knowing how to name and organize our contents. Even though keeping the **Content Browser** organized is not the main goal of this chapter, I didn't want to pass up on the opportunity to briefly talk about that.

One good way of keeping things tidy is to organize the folder structure in categories (**Materials**, **Characters**, **Weapons**, **Environment...**) and naming the different assets using Unreal's recommended syntax. You can find more about that on several discussion forums or on Epic Games' wiki:

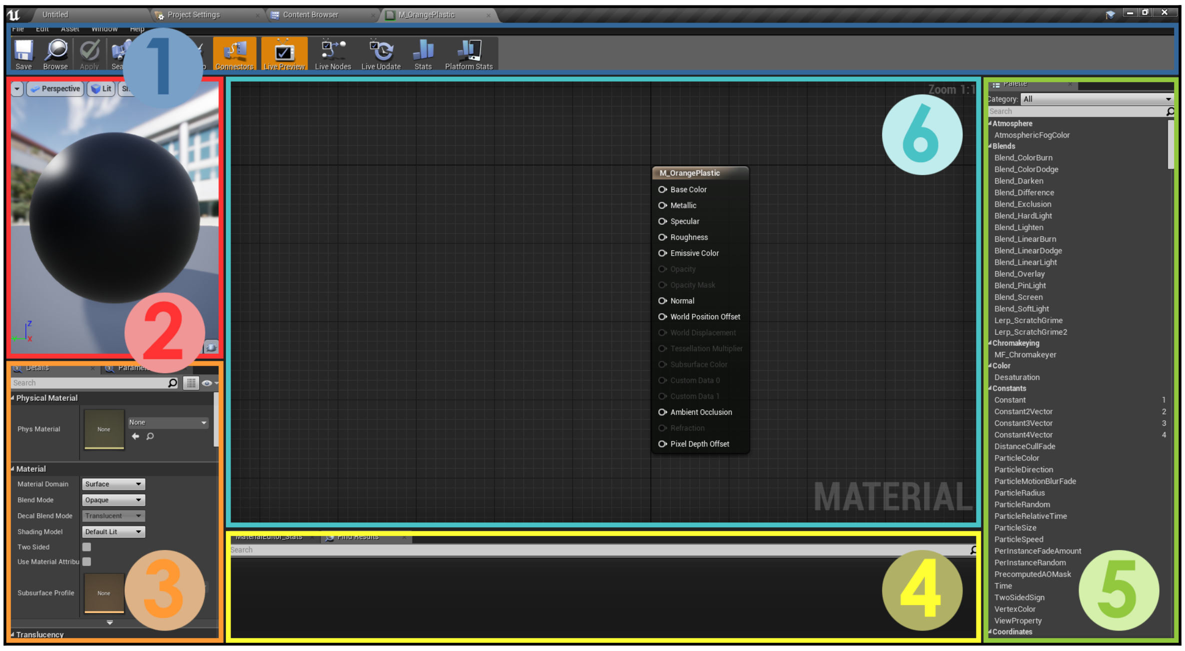
- Unreal Engine 4 style guide: `https://github.com/Allar/ue4-style-guide`
- Assets naming convention: `https://wiki.unrealengine.com/Assets_Naming_Convention`

The second important thing we want to be doing is to make sure that the layout we are looking at is the default one, just so that the images we will be including later on match what you'll be seeing in your monitor. To do that, go to **Window** | **Reset Layout**, as shown in the following screenshot:



Remember that resetting the layout to its default state can still make things not look perfectly equal between your screen and mine—that's because settings such as the screen resolution or its aspect ratio can hide panels or make them imperceptibly small. Feel free to move things around until you reach a layout that works for you!

Now that we've made sure that we are looking at the same screen, let's turn our attention to the material editor itself and the different parts that constitute it. By default, this is what we should be looking at:

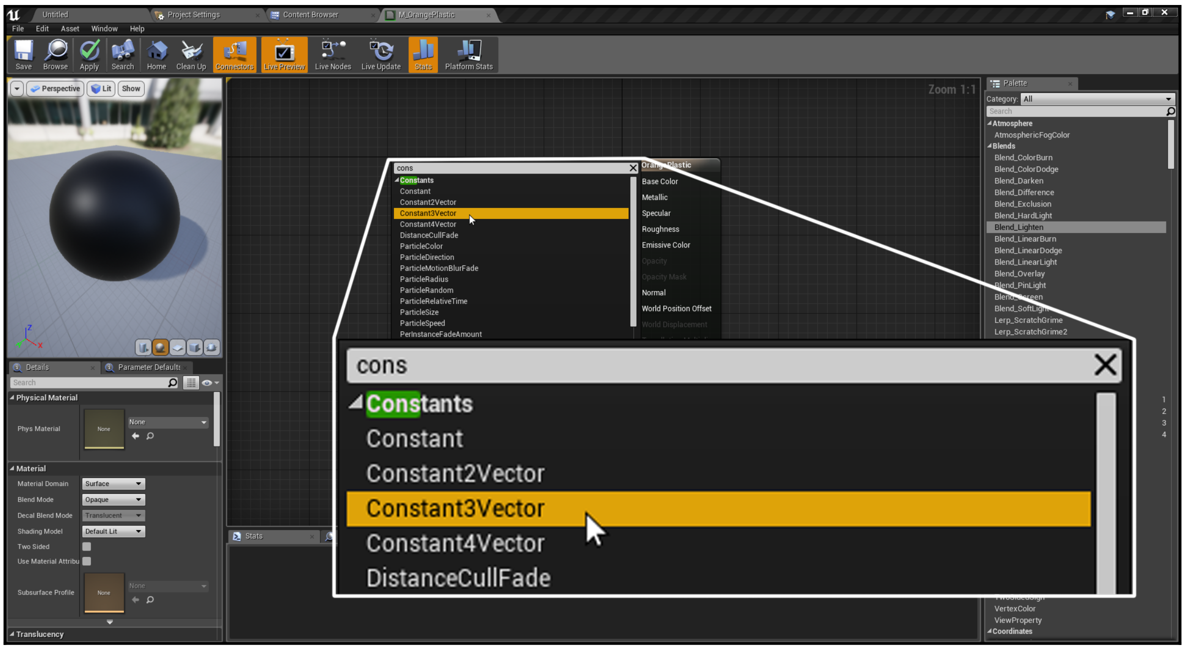


Physical Material
Phys Material: None
Material
Material Domain: Surface
Blend Mode: Opaque
Decal Blend Mode: Translucent
Shading Model: Default Lit
Two Sided
Use Material Attribu
Subsurface Profile: None
Translucency

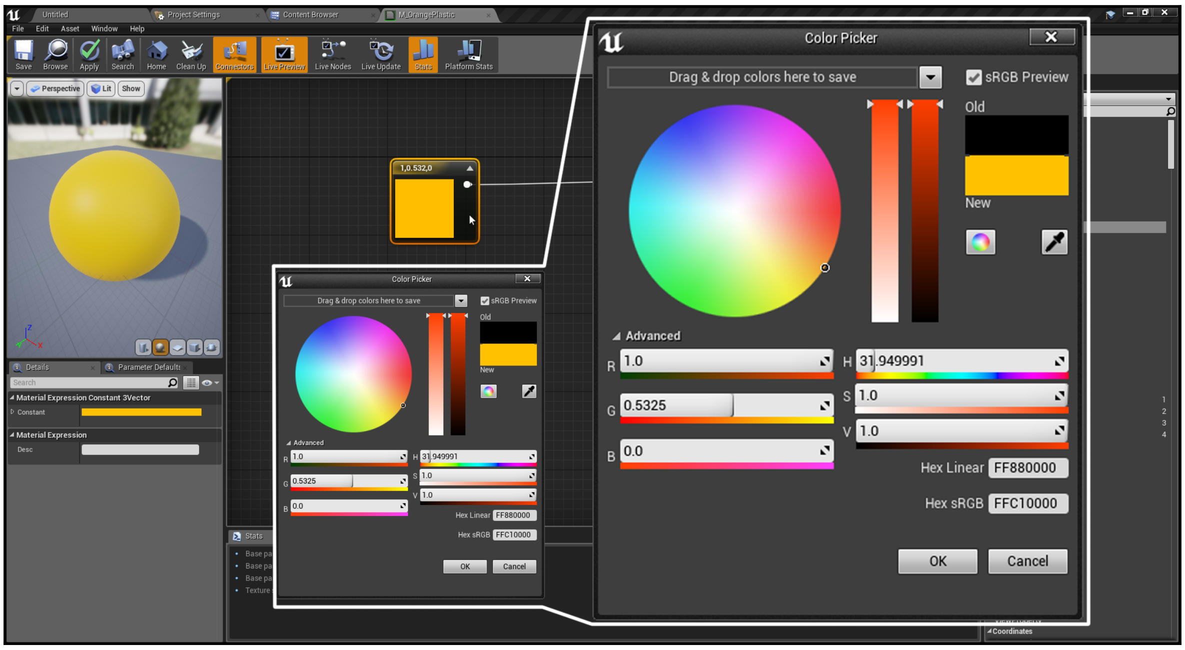
- The first part of the material editor is the **Toolbar**, a common section that you'll find in many other places within the engine. It lets you save your progress or apply any changes that you've made to your materials amongst other things.
- The second panel is the **Viewport**, where we'll be able to see what our material looks like. You can rotate the view, zoom in or out, and change the lighting setup of that window.
- The **Details** panel (**3**) is a very useful one, for here is where we can start to define the properties of the materials that we want to create. Its contents vary depending on what is selected in the main graph editor (the panel numbered **6**).
- The **Stats** and the **Find Results** panels (**4**) is where you can take a look at how costly your materials are or how many textures they are using.
- The **material node Palette** (**5**) is a library of different nodes and functions that we'll use to modify the materials we create.
- The **main graph editor** (**6**) is where the action happens, and where most of the functionality that you want to include in your materials needs to be visually scripted.

Now that we've taken a look at the different parts that make up the material editor in Unreal, we can start creating our own first simple material—a plastic. I find plastics to be a very straightforward type—even though we could make them as complicated as we want to. So, let's explore how we would go about at creating it:

1. Take a look at the main graph. By default, every time you create a new material, you should be looking at a central main node. You will see multiple pins, which are the elements where we want to connect the different elements we will be creating.
2. Right-click on the main graph, preferably to the left of the main material node, and start typing `constant`. As you start to write, notice how the auto-completion system starts to show several options: **Constant**, **Constant2Vector**, **Constant3Vector,** and so on. Select **Constant3Vector**, as shown in the following screenshot:



3. Having chosen that option, you will be able to see that a new node has now appeared. You can now connect it to the **Base Color** of the material node. If you are on the constant node, take a look at the **Details** panel and you'll be able to see that there are a couple settings that you can tweak. Since we want to move away from the default blackish appearance that the material now has, click on the black rectangle to the right of where it says **Constant** and use the color wheel to change its current value. I'm going to go with orange:



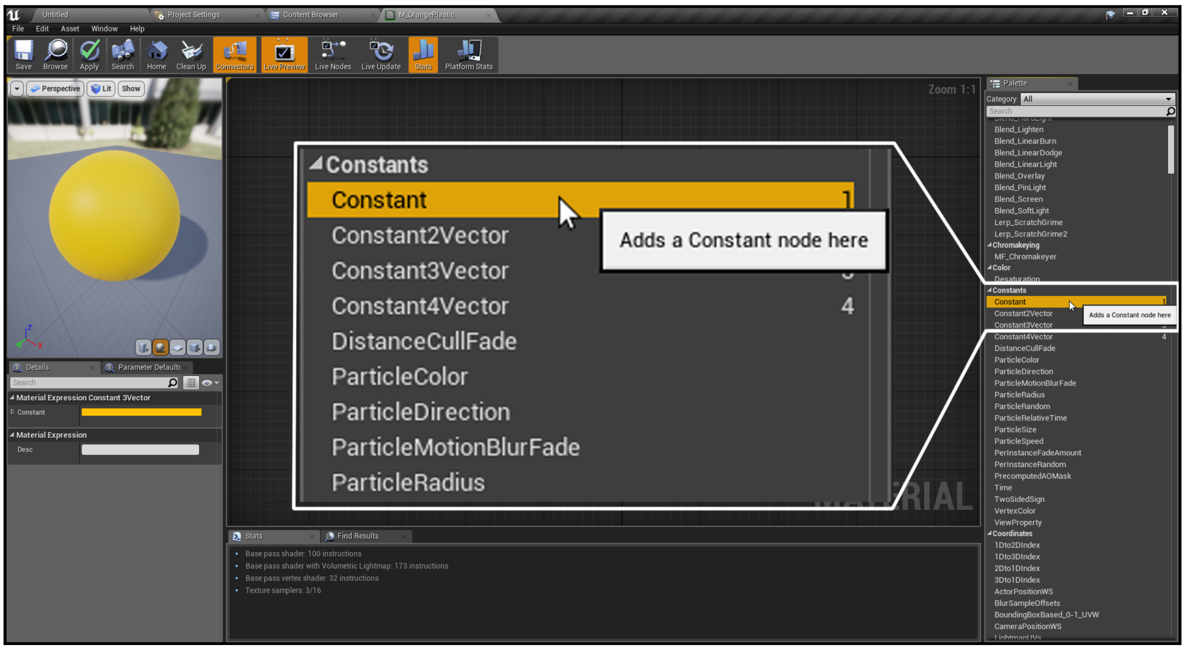
R 1.0
G 0.5325
B 0.0
H 31.949991
S 1.0
V 1.0
Hex Linear FF880000
Hex sRGB FFC10000

There's more to the base color property than meets the eye! Apart from the different options that are available to select a color, you might be interested to know that the actual value that gets connected to the material slot matters beyond the color choice. Certain materials have a measured intensity to them, and you can check that out on the following website: `https://docs.unrealengine.com/en-us/Engine/Rendering/Materials/PhysicallyBased`.

It's not something that you should concern yourself with at this stage, but can come in handy in the future!

At the moment, we can see that we have managed to modify the color of our material. We can now change how sharp the reflections are, as we want to go for a *plastic* look. In order to do so, we need to modify the **Roughness** parameter with another different constant. Instead of right-clicking and typing, let's choose it from the palette menu instead.

4. Navigate to the **Palette** section, and look for the **Constant** category. We want to select the first option in there, aptly named like this subsection itself. Alternatively, you can type its name in the search box at the top of the panel:

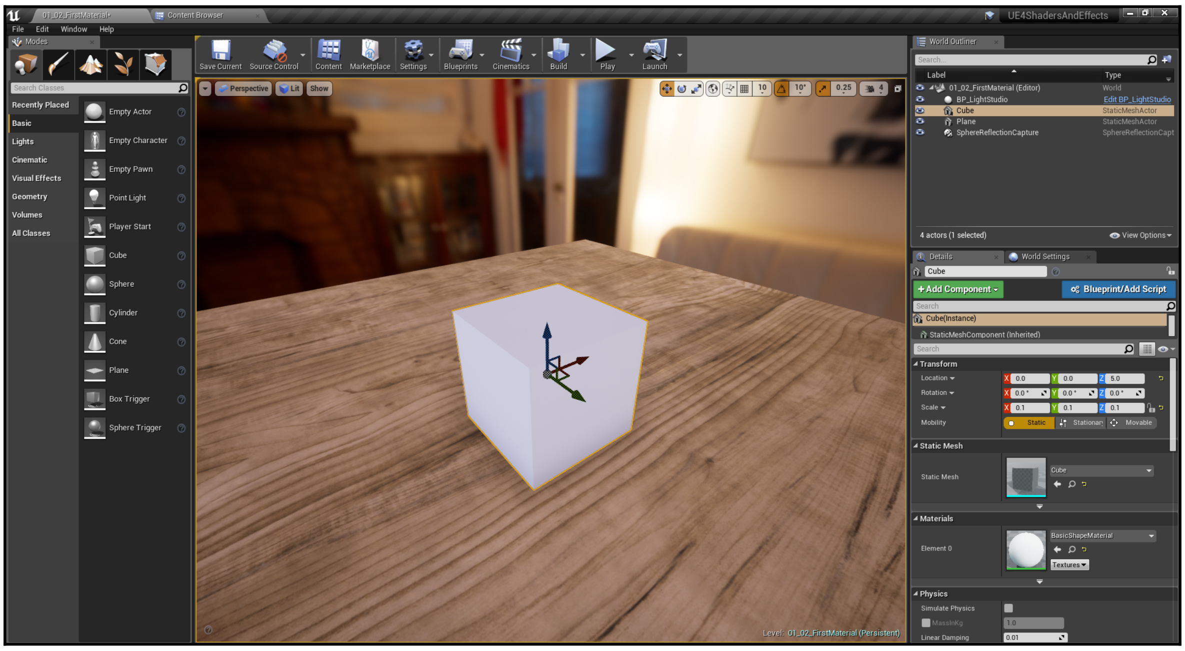


- Base pass shader: 100 instructions
- Base pass shader with Volumetric Lightmap: 173 instructions
- Base pass vertex shader: 32 instructions
- Texture samplers: 3/16

5. A new, smaller node should have now appeared. Unlike the previous one, we don't have the option to select a color—we need to type in a value. Let's go with something low, about **0.2**. Connect it to the **Roughness** pin.

If you look at the preview viewport, you will notice that the appearance of the material has now changed. It looks like the reflections from the environment are much sharper than before. This is happening thanks to the previously created constant pin, which, using a value closer to 0 (or black), makes the reflections stand out that much more. Whiter values decrease the sharpness of those reflections or, in other words, make the surface appear much more rough.

Having done so, we are now in a position where we can finally apply this material to a model inside of our scene. Let's go back to the main level and look at the **Modes** panel, particularly to the **Basic** section. Drag and drop a cube into the main level, and assign it the following values inside of the **Details** panel just so we are looking at the same:



Cube(Instance)
StaticMeshComponent (Inherited)

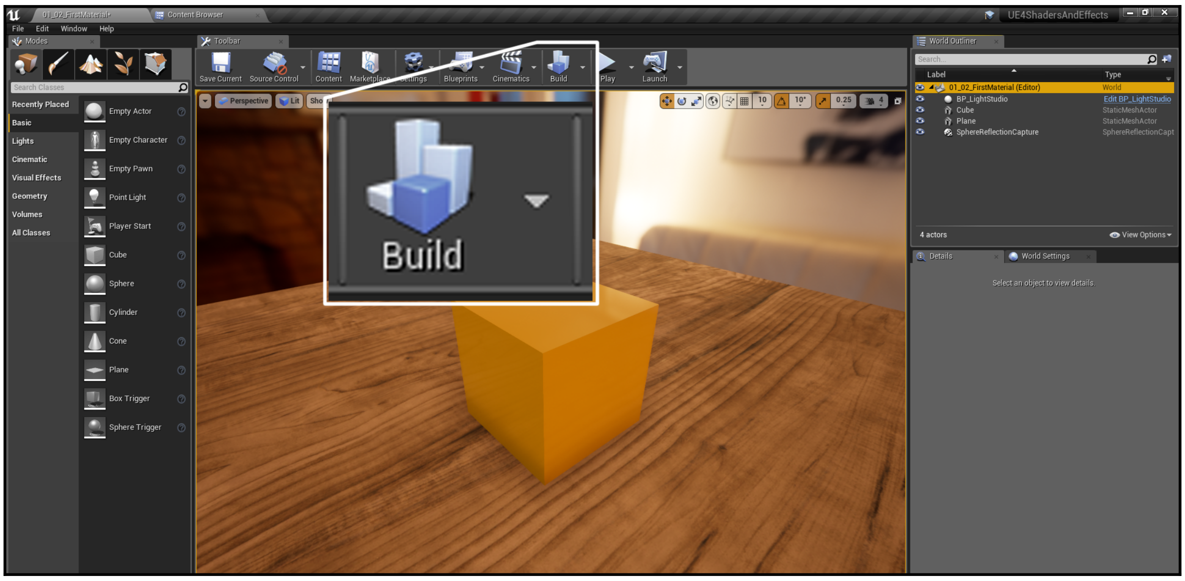
Transform
Location: X 0.0, Y 0.0, Z 5.0
Rotation: X 0.0°, Y 0.0°, Z 0.0°
Scale: X 0.1, Y 0.1, Z 0.1
Mobility: Static | Stationary | Movable

Static Mesh
Static Mesh: Cube

Materials
Element 0: BasicShapeMaterial

Physics
Simulate Physics
MassInKg: 1.0
Linear Damping: 0.01

Reducing the size of the cube will make it fit better into our scene. Now head over to the **Materials** section of the **Details** panel, and click on the drop-down menu. Look for the newly created material and assign it to our cube. Finally, click on the **Build** icon located on the toolbar as follows:



And there it is! We now have our material applied to a simple model, being displayed on the scene we had previously created. Even though this has served as a small introduction to a much bigger world, we've now gone over most of the panels and tools that we'll be using in the material editor. See you in the next recipe!

How it works...

We've used the present recipe to learn about the material editor and we've also created our first material. Knowing what each section does within the editor will help a lot in the immediate future, as what we've just done is but a prelude to our real target—creating a physically based material. Now we are in a much better position to tackle that goal, so let's look at it in the next recipe!

Before moving on though, let's check the nodes that we have used to create this simple material. From an artist's point of view, the names that the engine has given to something like a color value or a grayscale value can seem a bit confusing. It might be difficult to establish a connection between the name of the `Constant3Vector` node and our idea of a color. But there is a reason for all of this!

The idea behind that naming convention is that these nodes can be used beyond the color values we have just assigned them. At the end of the day, a simple constant can be used in many different scenarios—such as depicting a grayscale value, using it as a brightness multiplier, or as a parameter inside a material function. Don't worry if you haven't seen these other uses yet, we will—the point is, the names that these nodes were given tell us that there are more uses beyond the ones we've seen.

With that in mind, it might be better to think of those elements we've been using in more mathematical terms. For instance, think of a color as an **Red Green Blue** (**RGB**) value, which is what we are defining with that previous `Constant3Vector` node. If you want to use an RGB value alongside an alpha one, why not use the `Constant4Vector`, which allows for a fourth input? Even though we are at a very early stage, it is always good to familiarize ourselves with the different expressions the engine uses.

Our first physically based material

PBR is, at its core, a principle that several graphic engines try to follow. Instead of being a strict set of rules that every rendering program needs to abide by, it is more of an idea—one that dictates that what we see on our screens is the result of a study on how light behaves when it interacts with certain surfaces.

As a direct consequence, the so-called *PBR workflow* varies from one rendering solution to the next, depending on how the creators of the software have decided to program the system. For our case, what we are going to be looking at is the implementation that Epic Games has chosen for their Unreal Engine 4 real-time renderer.

However, we are going to do so in our already established recipe process, that is, by creating real examples of materials that follow the PBR workflow rather than just talking in a general way. Let's get to it!

Getting ready

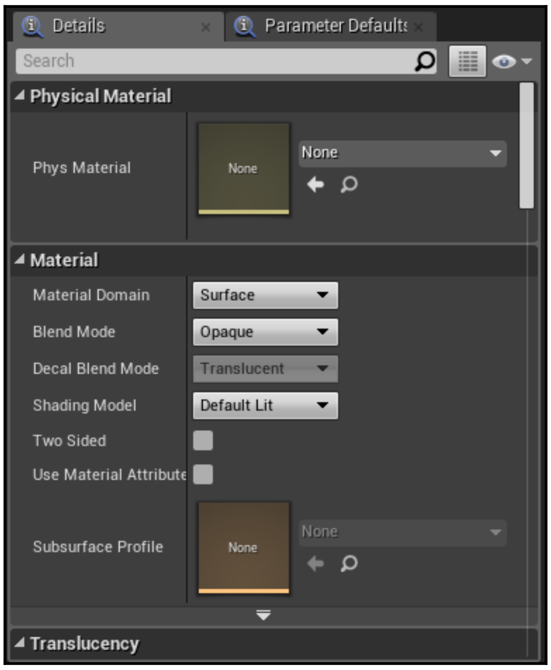
We don't need a lot in order to start working on this recipe—just the project we have previously created so we don't have to start from scratch. You can continue using the previous section's materials or create new ones, whatever works best for you! Something that would be helpful to have is the scene from the previous recipe open, for instance—that way we already have a 3D model in it that we can use to show our materials on.

We are going to be creating multiple materials in this section, so duplicating and modifying an already existing asset is going to be faster than creating several ones from scratch. To do this, just select any material that you want to duplicate on the content browser and press *Ctrl* + *W*.

How to do it...

Let's start our journey into the PBR pipeline by creating a new material and looking at the different attributes that define it:

1. Right-click anywhere inside of the **Content Browser** and select the material option in the **Create Basic Asset** section. Name it whatever you want—I'll go with `M_PBR_Metal` for this particular instance. Double-click on the newly created material to open up the material editor.
2. With the **Material** editor now open, we can start taking a look at the PBR workflow. The first material we are going to create is a metallic one, a particular type that uses most of the attributes associated to this pipeline. With that said, let's focus our attention on the following two different places—the **Details** panel and the main **Material** node itself:



Phys Material: None
Material Domain: Surface
Blend Mode: Opaque
Decal Blend Mode: Translucent
Shading Model: Default Lit
Two Sided
Use Material Attribute
Subsurface Profile: None

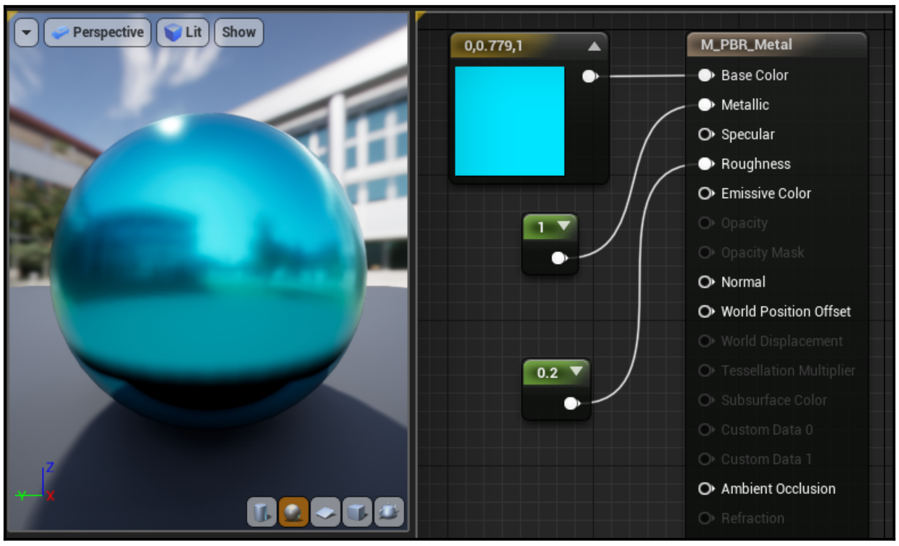
The settings you see here are the default ones for most materials in Unreal, and they follow the PBR pipeline very closely. The first option, the **Material Domain**, is currently set to **Surface**. That tells us that the material we are creating is meant to be used on a 3D model. **Blend Mode**, which has a value of **Opaque**, indicates that it is not a translucent material like glass. Finally, the shading model is set to **Default Lit**, which is the default one for most materials.

This configuration is the default one for most common materials, and the one that we'll need to use to define materials such as metal, plastic, wood, or concrete, to name a few.

3. With that bit of theory out of the way, let's create a **Constant3Vector** node anywhere in the graph and plug it into the **Base Color** input pin of our material. We've used the **Base Color** attribute in the previous recipe, and as we saw, this is the node where the overall color of a material should be plugged into.
4. The next item we will be creating is a **Constant**. You can do so by holding the *1* key on your keyboard and clicking anywhere within the material editor graph. Give it a value of `1` and plug it into the **Metallic** attribute of our material.

The **Metallic** attribute defines whether we are creating a metal or a non-metal material. We should use a value of **1** to define metallic surfaces and a value of **0** for non-metals—or we can leave this attribute unconnected, which would be the same as using a zero. Values between **0** and **1** should only be used in special circumstances, such as when dealing with metals that have been treated—corroded or painted metals and the like.

5. For our next step, let's replicate what we have just done—start by creating another constant and plugging it into the **Roughness** slot. This time, let's not give it a value of `1`, but something like `0.2` instead. The final material graph should look something like this:

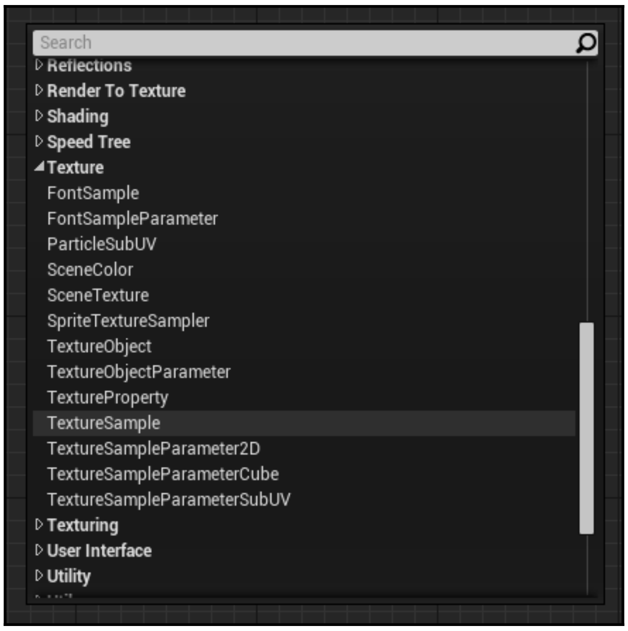


The attribute we are controlling through the previous constant defines how rough the surface of a given material should be. Higher values, such as **1**, simulate the micro details that make light scatter in all directions—which means we are looking at a matte surface where reflections are not clear. Values closer to zero result in those imperfections being removed, allowing a clear reflection of the incoming light rays and a much clearer reflected image.

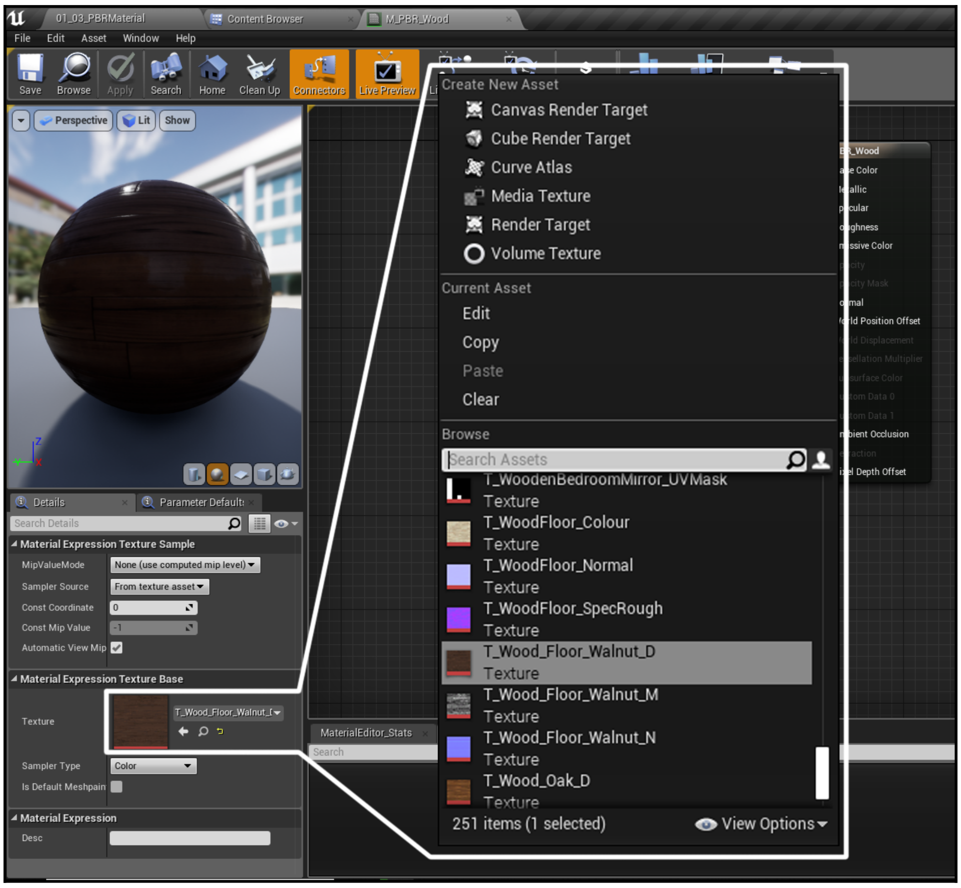
Through the previous steps, we have taken a look at some of the most important material attributes used to define a PBR material. We've done so by creating a metal, which can be a good example for some of the previous properties. However, it will be good to create another quick material that is not a metallic one—this is because some of the other properties of the PBR workflow, like the specular material attribute, are meant to be used in such cases.

6. Create another material, which we can name `M_PBR_Wood`, and open the material editor for that asset.
7. Let's plug something into the **Base Color material** attribute—but instead of using a plain value, let's go with an image this time. The **Starter Content** provides multiple textures that can be used for this very purpose, so let's make use of one of those resources.

Right-click anywhere inside of the main graph for our newly created material and search for **TextureSample**, like in the next screenshot:



8. With that new node on our graph, click on it to access the options in the **Details** panel. Click again on the drop-down menu found in the **Material Expression Texture Base | Texture** slot and type `wood`. Select the **T_ Wood_ Floor_ Walnut_ D** asset and connect the **Texture Sample** node into the **Base Color material** attribute as follows:

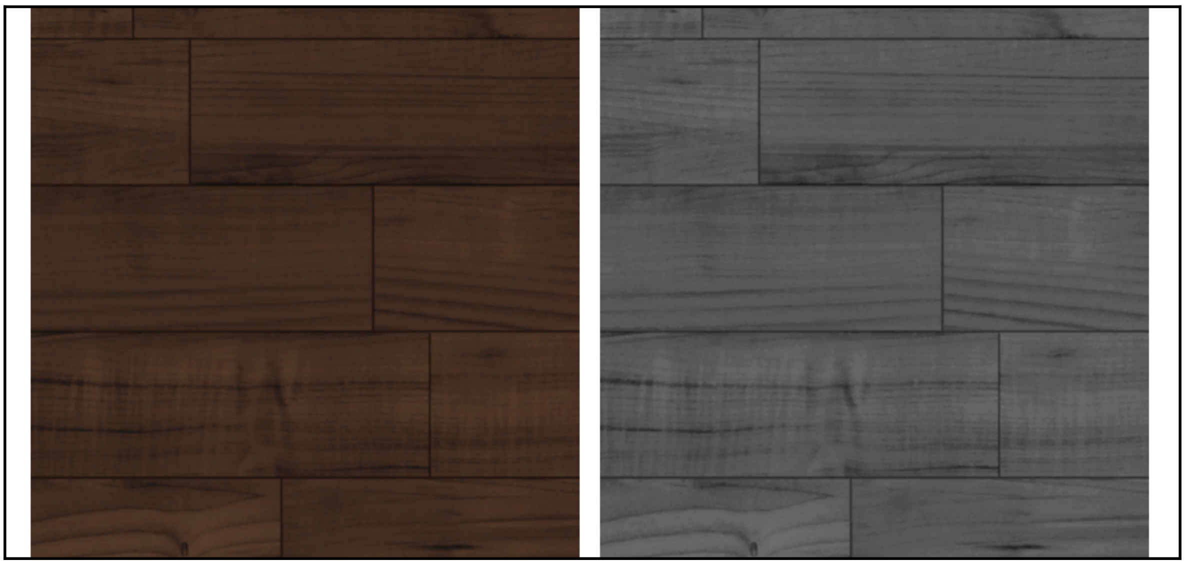


If you want to get hold of more textures online, feel free to browse the internet for more of them. A good place where I like to search for these types of resources is `www.textures.com`, which allows you to download several samples a day once you create a free account.

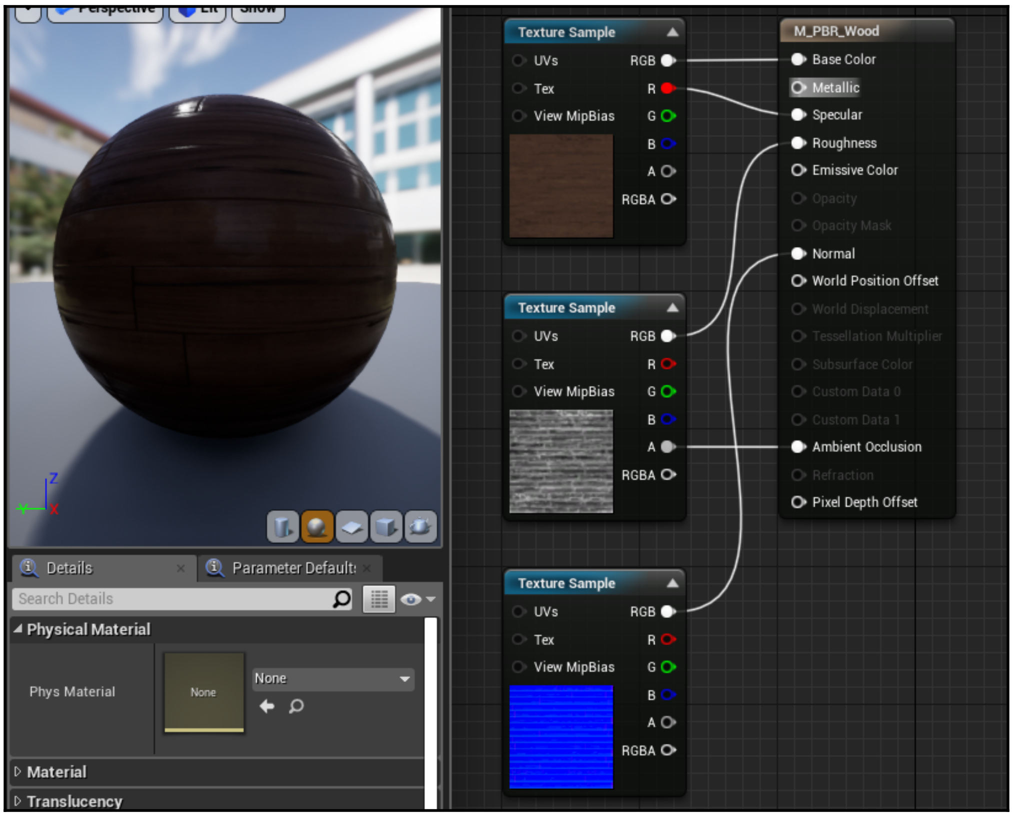
With that done, it's time to be looking at another material attribute—the **Specular** parameter. Unlike roughness, this node controls how much light is being reflected by the material and not how clear those reflections are. We therefore tend to modify the specular level when we have small-scale occlusion or small shadows happening across a surface, similar to what would be happening for the texture that we chose before.

9. The seams in between the wood boards are a good place to use a specular map, as those areas will reflect less light. In Unreal, such places are described with values close to **0** (black). Knowing that, drag a pin from the red channel of the previously created **Texture Sample** node into the **Specular** attribute of the main material node.

   You might be wondering why we are using the red channel of the wood texture to drive the specular parameter. The simple answer is that even though we could create a custom black and white image to achieve the same effect, any of the original textures' channels are black and white values that contain the information that we are after. Because seams are going to contain darker pixels than other areas, the end result we achieve is still very similar if we use the red channel of the original texture. You can see in the next image our source asset and the red channel by its side:

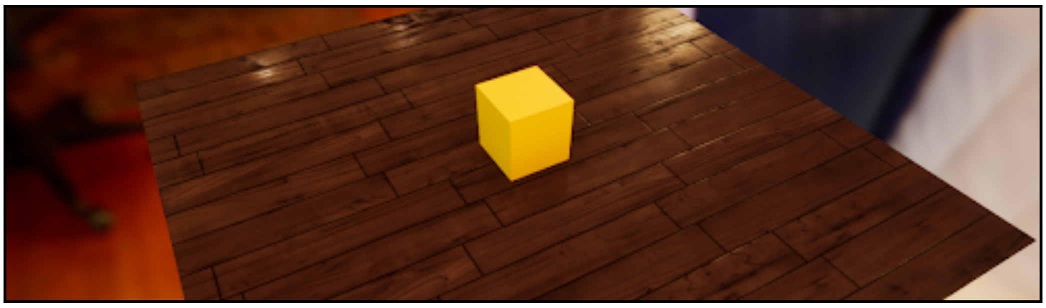


10. Copy the **Texture Sample** node twice, since we are going to use more textures for the roughness and the normal material attribute slots.
11. Just as we did previously, select the **T_ Wood_ Floor_ Walnut_ M** and the **T_ Wood_ Floor_ Walnut_ N** assets on each of the new nodes. Connect the first one to the **Roughness** slot and the second one to the **Normal** node. Save the material and click on the button that says **Apply**. Your material node graph should look something like this:



Phys Material: None

12. Navigate back to the main level, and select the floor plane. In the **Details** panel, scroll down to the **Materials** section and assign the **M_PBR_Wood** material we have just created. Take a look at what our scene looks like now:



Nice job, right? The new nodes we've used, both the specular and the normal ones, contribute to the added details we can see in the preceding screenshot. The specular node diminishes the light that is being reflected in the seams between the wood planks, and the normal map modifies the direction in which the light bounces from the surface. The combined effect is that our model, a flat plane, looks as if it has much more geometrical detail than it really has.

How it works...

Remember how we were talking about each renderer having its own implementation of a PBR workflow? Well, we have just taken a look at how Epic has chosen to set up theirs!

As we have already said, efficiency and speed are at the heart of any real-time application. These are two factors that have heavily influenced the path that the engineers at Epic have chosen when coding their physical approach at rendering. That being the case, the parameters that we have tweaked are the most important ones when it comes to how Unreal deals with the interaction between light and 3D models. The base color gives us the overall appearance of the material, whilst roughness indicates how sharp or blurry the reflections are. Metallic enables us to specify whether an object is made out of metal, and the specular node lets us influence how intense those reflections are. Finally, using normal maps allows for the modification of the direction in which the light gets reflected—a useful technique for adding details without actually using more polygons.

The previous parameters are quite common in real-time renderers, but not every program uses the same ones. For instance, offline suites such as VRay use other types of calculations to generate the final output—physically based in their nature, but using other techniques. This shows us that, at the end of the day, the PBR workflow that Epic uses is specific to the engine and we need to be aware of its possibilities and the limitations.

Throughout the current recipe, we have managed to take a look at some of the most important nodes that affect how the physically based rendering gets tackled in Unreal Engine 4. Base color, roughness, specularity, ambient occlusion, normal maps, and the metallic attribute all constitute the basics of the PBR workflow.

Having seen all of them, we are now ready to start looking into how to build more complex materials and effects. And even though we still need to understand some of the other areas that affect our pipeline, we can do so with the certainty that the basics are covered.

Creating some simple glass with the translucent blend mode

In the previous section, we had the opportunity to create a basic material that followed the physically based approach that Unreal Engine uses to render elements into our screens. By using nodes and expressions that affected the roughness or the metallic attributes of a material, we saw how we could potentially create endless combinations—going from plastics to concrete, metal, or wood.

Those previous examples can be considered simple ones—for they use the same shading model to calculate how each element needs to be rendered. Most of the materials that we experience in our daily lives fall into that category, and they can be described using the attributes we have previously tweaked. In spite of that, there are always examples that can't be exactly covered with one unique shading model. The way that light behaves when it touches glass, for example, needs to be redefined in those cases. The same applies to other elements, such as human skin or foliage, where light distribution varies from that of a wooden material.

With that in mind, we are going to create several small examples of materials that deviate from the standard shading model—starting with some simple glass. This will work as an introductory level, just so we can create more complex examples at a later stage. Buckle up and let's dive right in!

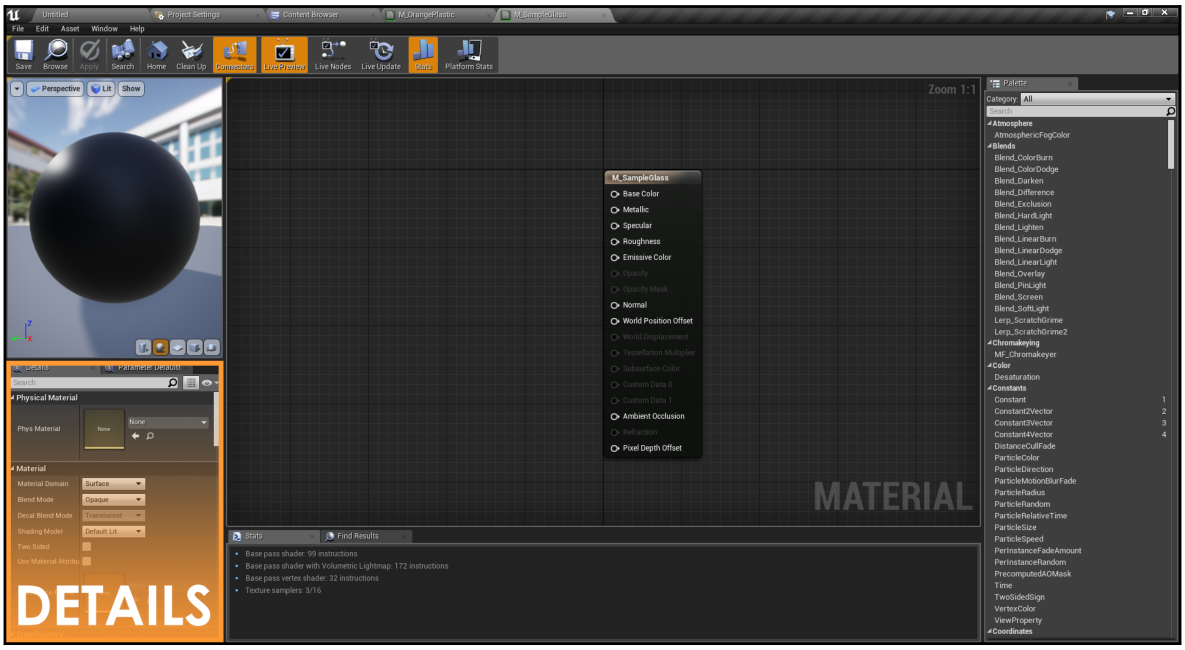
Getting ready

In order to start this recipe, you are not going to need a lot of anything. The sample Unreal project we have previously created will serve us fine, but feel free to create a new one if you are starting in this section of the book. It is completely fine to use standard assets, such as the ones included with the engine, but I've also prepared a few of them that you can download if you want to closely follow this book.

How to do it...

The first example that we are going to create is going to be some simple glass. As before, right-click in the appropriate subfolder of your **Content Browser** and create a new material. Here's how we go about it:

1. Let's name it with a pertinent name, something like `M_SampleGlass`, as that's what we'll be creating!
2. Open up the material editor, and focus on the **Details** panel. That's the first area we are going to operate on. Make sure you have the main material node selected—if you haven't created anything else, that's the only element that should exist on the main editor graph:

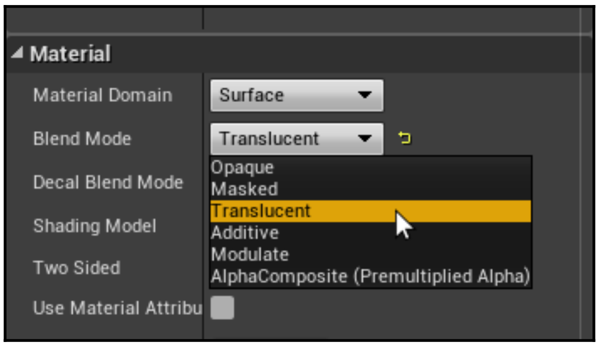


Physical Material
Phys Material: None
Material
Material Domain: Surface
Blend Mode: Opaque
Decal Blend Mode: Translucent
Shading Model: Default Lit
Two Sided
Use Material Attribu

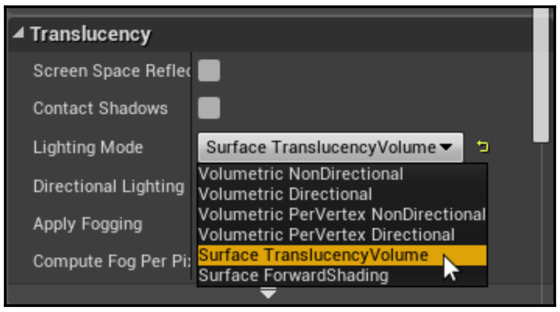
Stats

- Base pass shader: 99 instructions
- Base pass shader with Volumetric Lightmap: 172 instructions
- Base pass vertex shader: 32 instructions
- Texture samplers: 3/16

3. Having the main node selected, you'll be able to see that the second editable attribute under the **Material** section of the **Details** panel is the **Blend Mode**. Let's change that from the default value of **Opaque** to the more appropriate **Translucent** one as follows:



4. After this change has happened, you'll note that several options have been grayed out inside of the main material node. We'll come back to this shortly.
5. Without leaving the **Details** panel, you can now scroll down to the **Translucency** section of the main material node. You should be able to find a drop-down menu named **Lighting Mode**, which we'll need to change from the default value of **Volumetric NonDirectional** to the one named **Surface Translucency Volume**, as shown in the following screenshot:

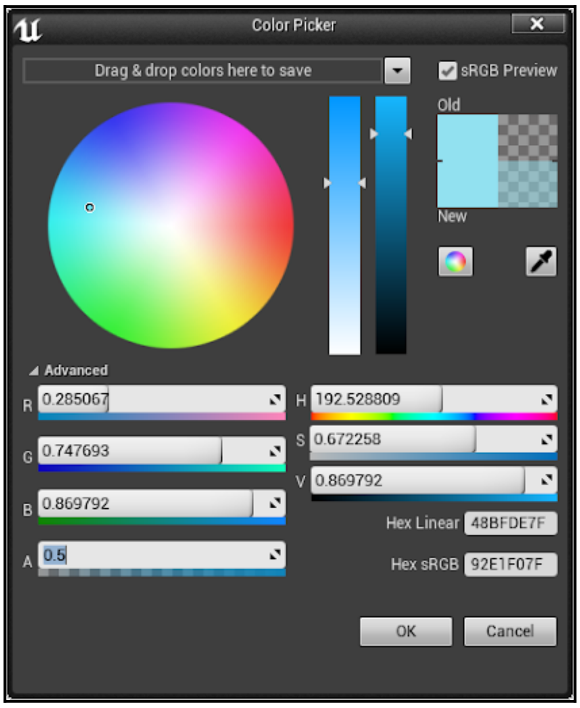


If you hover over each of the options inside of the **Lighting Mode** drop-down menu, you should be able to take a look at their description. You'll note that some of the options are meant to be used with particles, while others are meant for 3D models. That's the reason why some of the material attributes were previously grayed out— some options don't make sense to be used if we are going to be applying the material to a particle, for example, so these are left out.

6. With that out of the way, let's now attach a **Constant4Vector** to the **Base Color** node and give it an interesting value. I'm going with a bluish tone, as we'll be creating a glass and they usually have that kind of tint.

Why a Constant4Vector and not a Constant3Vector, as we used last time? This new type that we are using includes a fourth parameter, which can be used as an alpha value, something very useful for glass-like materials as you'll see for yourself in a moment.

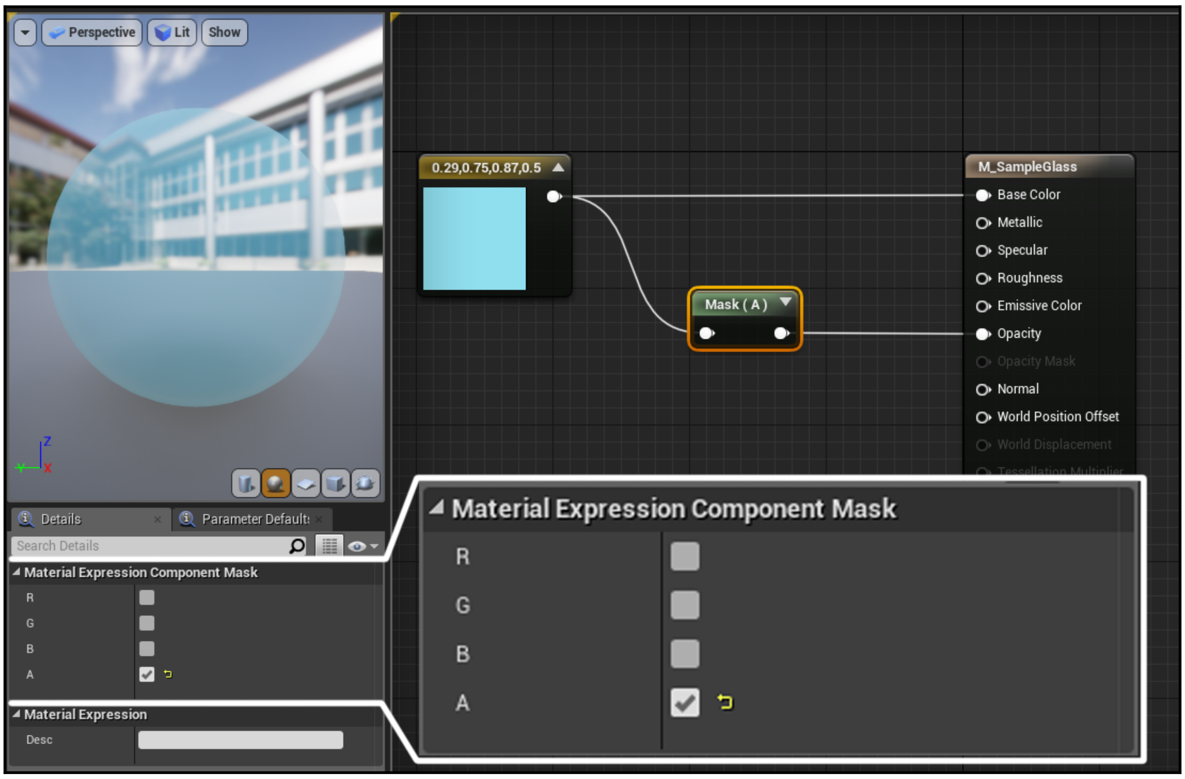
7. Without leaving the Constant4Vector behind, set the *alpha value* to something like **0.5**. Don't go all the way with this parameter! Setting it either as a **0** or a **1** would make our future material fully transparent or opaque, so choose something in between. Plug the value into the **Base Color** material node as follows:



R 0.285067
G 0.747693
B 0.869792
A 0.5
H 192.528809
S 0.672258
V 0.869792
Hex Linear 48BFDE7F
Hex sRGB 92E1F07F

8. Now it's time to plug in the alpha value of our **Constant4Vector** into the **Opacity** slot of our material. Drag from the pin of the **Constant4Vector** into an empty space in the main graph and release the left mouse button. A contextual menu should now appear, and you want to type **mask**. Selecting **ComponentMask** is what we want to be doing now!

9. With the component mask selected, let's take a look at the details panel. In there you'll be able to select which of the four components from the Constant4Vector node you want to use. For our case, as we'll be driving the opacity through the alpha, let's just tick the last option.
10. Finally, connect the mask to the **Opacity** pin. Click on the **Apply** button and save the material. The preview window may take a moment to update itself, but once it does we should be looking at a translucent material like the following:


Perspective
Lit
Show
0.29,0.75,0.87,0.5
Mask (A)
M_SampleGlass
Base Color
Metallic
Specular
Roughness
Emissive Color
Opacity
Opacity Mask
Normal
World Position Offset
World Displacement
Tessellation Multiplier
Details
Parameter Defaults
Search Details
Material Expression Component Mask
R
G
B
A
Material Expression
Desc