Creating Maps and Materials

In this chapter, we will cover the following topics:

1 Creating a basic material with Standard Shader (Specular setup)

2 Adapting a basic material from Specular setup to Metallic

3 Applying Normal maps to a material

4 Adding Transparency and Emission maps to a material

5 Highlighting materials at mouse-over

6 Adding Detail maps to a material

7 Fading the transparency of a material

8 Playing videos inside a scene

9 Using a custom shader to make instances of a sprite color differently at Run-Time

# Introduction

There is a close relationship between Textures, Materials and Shaders:

Textures are two-dimensional images. The surface of 2D and 3D objects in Unity games are defined by meshes. The Texture images are mapped onto meshes by Materials – each point (vertex) on a mesh has to be mapped to some value in the Texture. A Texture may indicate colors, but may indicate bumps/wrinkles or transparency etc. – all of which can contribute to determining what is finally rendered for the user to see.

Materials specify which Shaders should used to render the images onto the meshes; plus values for the Shader’s parameters (e.g. which textures / parts of a texture map, colours, other values etc.)

Shaders define the method to render an object (code and math). Shaders can use multiple textures for more sophisticated results, and specify which parameters may be customized in the Material Inspector.

Unity offers Physically-Based Shaders (PBS). Physically-Based Rendering is a technique that simulates the appearance of materials based on how the light reacts with that material (more specifically, the matter from which that material is made) in the real world. Such a technique allows for more realistic and consistent materials. So, your creations in Unity should look better than ever. Creating materials in Unity has also become more efficient now. Once you have chosen between the available workflows (Metallic or Specular setup; we'll get back to that later), there is no longer the need to browse the drop-down menus in search of specific features, as Unity optimizes the shader for the created material, removing unnecessary code for unused properties once the material has been set up and the texture maps have been assigned.

For a deep understanding of Physically-Based Rendering, we recommend you to take a look at The Comprehensive PBR Guide, written by Wes McDermott from Allegorithmic, freely available in two volumes at <http://www.allegorithmic.com/pbr-guide>. Allegorithmic's guide contains invaluable information on PBR theory and techniques, having been a fundamental reference for this chapter. A great resource that we'd recommend you take a look at is Mastering Physically Based Shading in Unity 5 by Renaldas Zioma (Unity), Erland Körner (Unity), and Wes McDermott (Allegorithmic), available at <http://www.slideshare.net/RenaldasZioma/unite2014-mastering-physically-based-shading-in-unity-5>.

Another resource is Physically Based Shading in Unity by Aras Pranckevičius (Unity), available at <http://aras-p.info/texts/files/201403-GDC_UnityPhysicallyBasedShading_notes.pdf>.

## Creating and saving texture maps

The visual aspects of a material can be modified through the use of textures. In order   
to create and edit image files, you will need an image editor such as Adobe Photoshop   
(the industry standard, and has its native format supported by Unity), GIMP, and so on. In order to follow the recipes in this chapter, it's strongly recommended that you have access   
to a few pieces of software like these.

When saving texture maps, especially the ones that have an Alpha Channel, you might want to choose an adequate file format. PSD, Photoshop's native format, is practical for preserving the original artwork in many layers. The PNG format is also a great option, but please note that Photoshop doesn't handle PNG's Alpha channel independently of the transparency, possibly compromising the material's appearance. Also, PNG files don't support layers. For this chapter, we will often use the TIF format for three main reasons: (a) it's open to those not using Photoshop; (b) it uses layers; (c) it preserves the Alpha Channel information. The file size is significantly greater than in PSDs and PNGs, so feel free to save your work as PSDs (if you have Photoshop) or PNGs (if you don't need layers and, if using Photoshop, Alpha Channels).

Finally, a word of advice - although it's possible to manually create texture maps for our materials by using the traditional image editing software, new tools such as Allegorthmic's Substance Painter and Bitmap2Material make this work much more efficient, complete, and intuitive, complementing the traditional texture-making process or replacing it altogether - in a similar way to what zBrush and Mudbox did for 3D modeling. For design professionals, we strongly recommend at least trying such tools. Note, however, that products from Allegorithmic won't make use of Unity's Standard Shader, relying on the substance files (which are natively supported by Unity).

## The big picture

To understand the Standard Shaders, it's a good idea to know the workflows, their properties, and how they affect the material's appearance. There are, however, many possible ways to work with materials - texture map requirements, for instance, might change from engine to engine, or from one tool to another. Presently, Unity supports two different workflows: one based on Specular, and another based on Metallic values. Although both workflows share similar properties (such as Normal, Height, Occlusion, and Emission), they differ in the way the diffuse color and reflectance properties are set up.

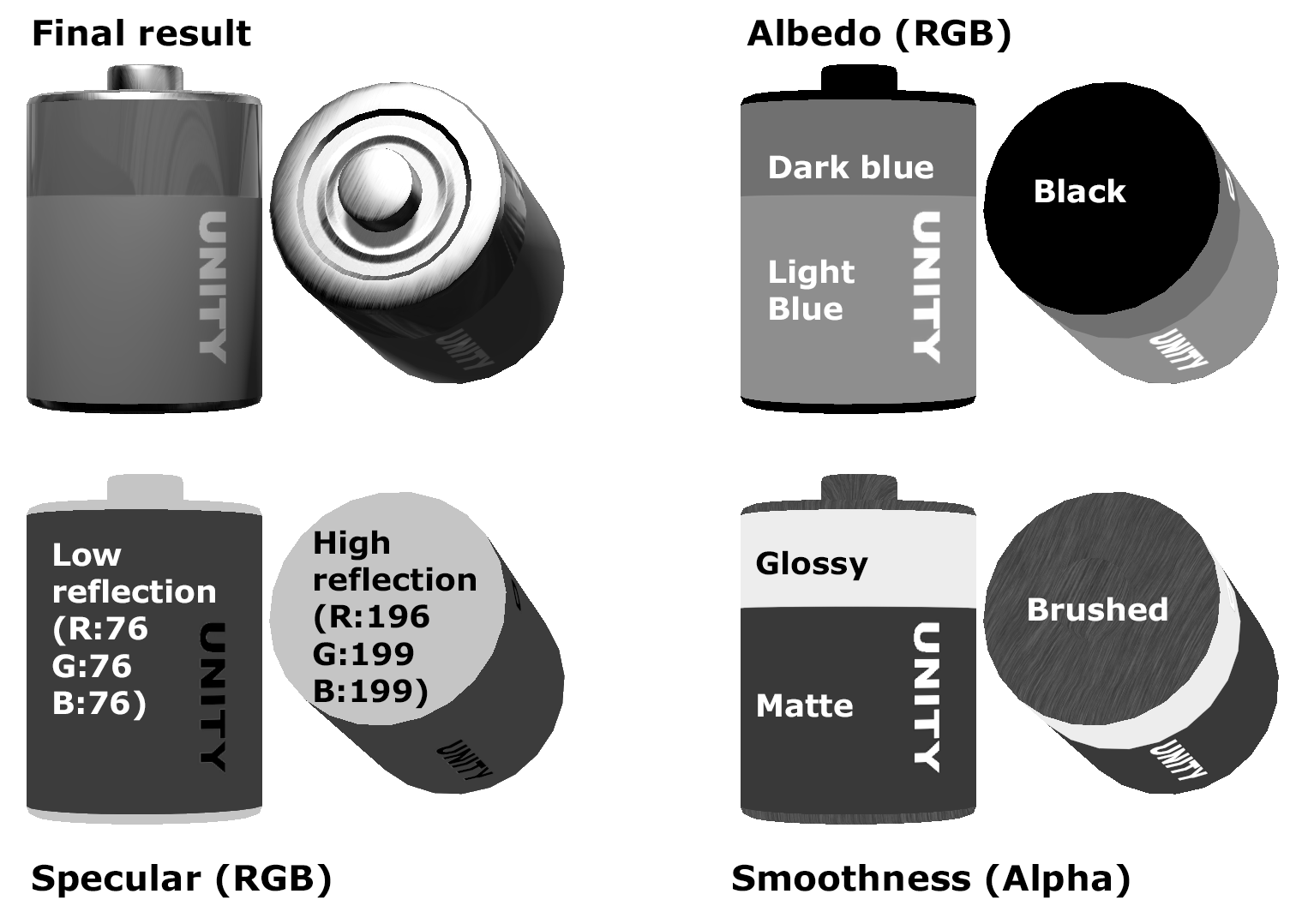
### Specular workflow

Unity's Standard Shader (Specular setup) uses Albedo and Specular/Smoothness maps, combining them to create some of the material's aspect—mainly its color and reflectance qualities. The following shows the difference between Albedo and Smoothness maps:

Albedo: This is the material's diffused color. Plainly and simply, this is how you   
usually describe the appearance of the material (the British flag is red, white and blue; Ferrari's logo is a black horse in a yellow setting; some sunglasses' lenses are semi-transparent gradients, and more). This description, however, can be deceptive. Purely metallic objects (such as aluminum, chrome, gold, and others) should have black as their diffuse color. Their colors, as we perceive them, have originated from their specular channel. Non-metallic objects (plastic, wood, and even painted or rusted metal), on the other hand, do have very distinct diffuse colors. Texture maps for the Albedo property feature RGB channels for colors and (optionally) an Alpha Channel for transparency.

Specular/Smoothness: This refers to the shininess of the material. Texture maps make use of RGB channels for specular color (which informs hue and intensity), and Alpha Channel for smoothness/gloss (dark values for less shiny surfaces and blurred reflections; light/white values for shiny, mirror-like appearance). It is important to note that non-metallic objects feature neutral, very dark specular colors (with plastic, for instance, you should work with a grey value around 59). Metallic objects, on the other hand, feature very light values, and are also a bit yellowish in hue.

To illustrate such concepts, we have created a battery object (shown below), featuring brushed metal caps and a plastic body. Observe how each map contributes to the final result:



Insert Image 8775\_05\_01.png

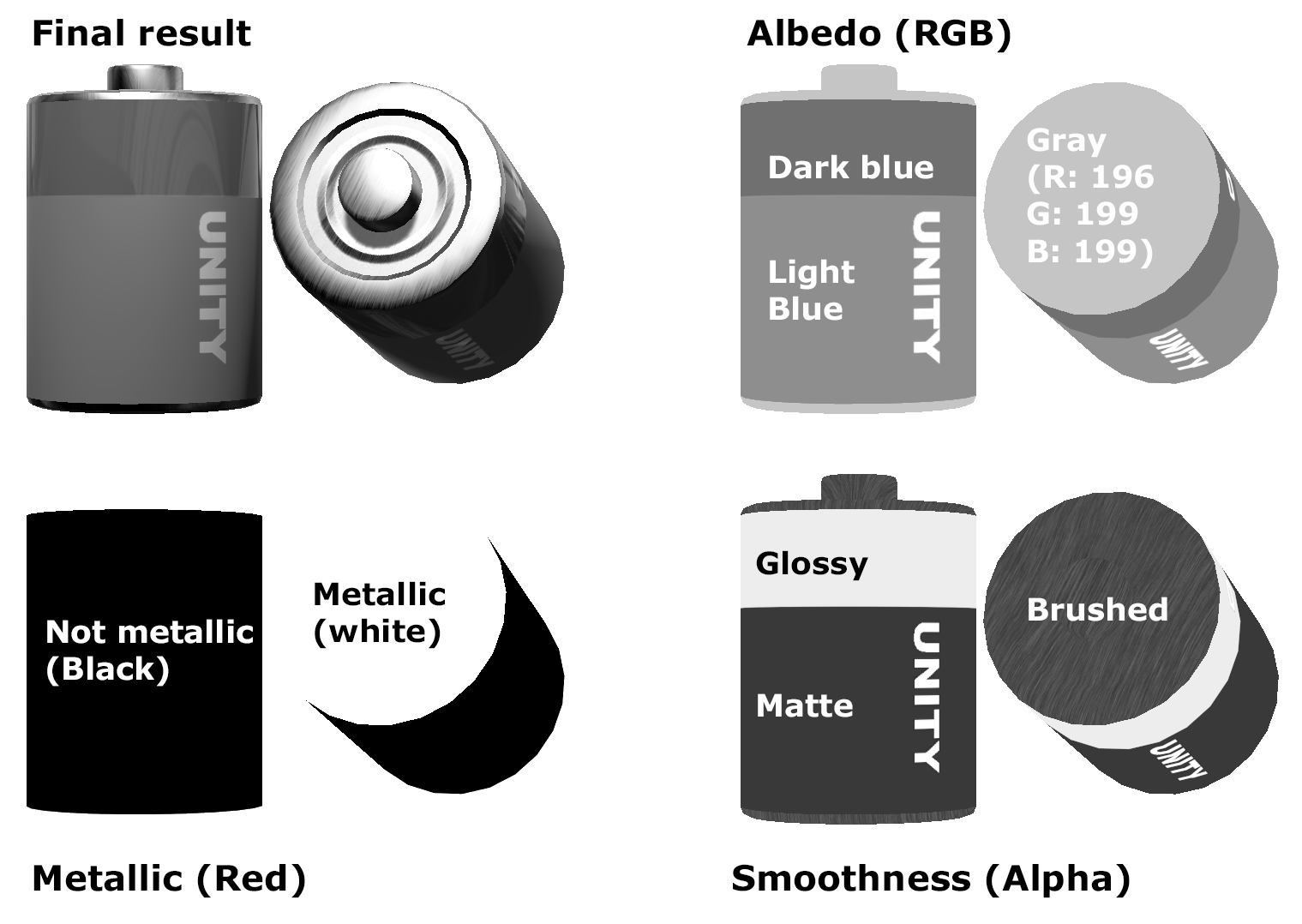
### The metallic workflow

Unity's default Standard Shader combines Albedo and Metallic/Glossiness maps to create the color and reflectance qualities of the material. The following are the differences:

Albedo: As in the Specular workflow, this is the material's diffuse color; how you would describe the material. However, Albedo maps for the Metallic workflow should be configured in a slightly different way than ones for Specular workflow. This time around, the perceived diffuse color of metallic materials (grey for iron, yellow/orange for golden, and so on) have to be present in the Albedo map. Again, Albedo maps feature RGB channels for the colors and (optionally) an Alpha channel for transparency.

Metallic/Smoothness: This refers to how metallic the material looks. Metallic texture maps make use of the Red channel for the Metallic value (black for non-metallic and white for metallic materials that are not painted or rusted) and the Alpha Channel for smoothness (in a similar way to the Specular workflow). Please note that Metallic maps do not include any information on hue, and in these cases the yellow-ish nature of the metallic gloss should be applied to the Albedo map.

To reproduce the battery that illustrated the Specular workflow by using the Metallic workflow, maps would have to be recreated as follows:



Insert Image 8775\_05\_02.png

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | You might have noticed that we've used white to convey a metallic object. Technically, since only the Red channel is relevant, we could have used red (R: 255, G: 0, B: 0), yellow (R: 255, G: 255, B: 0) or, for that matter, any color that has a red value of 255. |  |

### Other material properties

It's also worth mentioning that Unity's Standard Shaders support other maps such as:

Normal maps: The normal map adds detailed bumpiness into the material,   
simulating a more complex geometry. For instance, the internal ring on the   
positive (top) node of the battery that illustrated shader workflows is not modeled   
in the 3D object's geometry, but rather created through a simple normal map.

Occlusion maps: A greyscale map is used to simulate the dark sections of an   
object under ambient light. Usually, it is used to emphasize joints, creases,   
and other details of geometry.

Height maps: These add a displacement effect, giving the impression of depth   
without the need for complex geometry.

Emission maps: These add color emitted by the material, as if self-illuminated,   
such as fluorescent surfaces or LCDs. Texture maps for Emission feature RGB   
channels for color.

### Unity samples and documentation

Before you start, it might be a good idea to read Unity's documentation on textures and materials. They can be found online at

https://docs.unity3d.com/Manual/Textures.html

<https://docs.unity3d.com/Manual/Materials.html>

Finally, Unity has put together a great resource for those looking for some pointers regarding how to set up maps for a variety of materials: the Shader Calibration Scene, which can be downloaded (for free) from the Unity Asset Store. It is a fantastic collection, featuring sample materials (both Metallic and Specular setup) for wood, metal, rubber, plastic, glass, skin,   
mud, and much more.

# xxx

xxx

## Getting ready

We have included xxx inside the 05\_01 folder.

## How to do it...

To play xxx do the following:

1. Create a new Unity 2D project and import the sound clip files.
2. Create a C# script class, PlaySounds, in a new folder \_Scripts, containing the following code, and add an instance as a scripted component to the Main Camera:

using UnityEngine;

[RequireComponent(typeof(AudioSource))]

public class PlaySounds : MonoBehaviour

{

public AudioClip clipEatCherry;

public AudioClip clipExtraLife;

private AudioSource audioAudioSource;

void Awake()

{

audioAudioSource = GetComponent<AudioSource>();

}

void Update()

{

if (Input.GetKey(KeyCode.UpArrow))

{

audioAudioSource.PlayOneShot(clipEatCherry);

}

if (Input.GetKey(KeyCode.DownArrow))

{

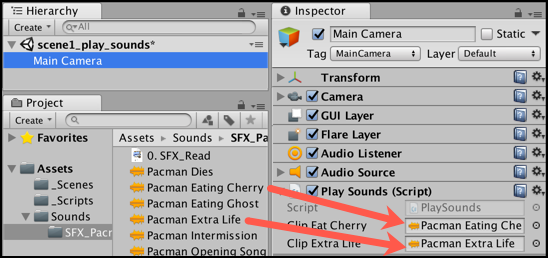
audioAudioSource.PlayOneShot(clipExtraLife);

}

}

}

1. Ensure that the MainCamera GameObject is selected in the Hierarchy. Then, in the Inspector panel, drag the Pacman Eating Cherry sound clip from the Project panel into the public AudioClip variable Pacman Eating Cherry in the PlaySounds (Script) scripted component. Repeat this procedure for the Pacman Extra Life sound clip. These steps are illustrated in the screenshot.



Insert Image 8775\_04\_03.png

1. Run the scene, and press the UP and DOWN arrow keys to play the different sound effects.

## How it works...

You created a C# script class xxx

NOTE: Cannot pause/interrogate sounds played with PlayOneShot

While great for short, one-off sound effects, a limitation of the PlayOneShot() method is that you cannot then interrogate the status of the playing sound (has it finished, at what point is it playing etc.). Nor can you pause/restart a sound played with PlayOneShot(). For such detailed control of sounds each sound needs its own AudioSource component.

Learn more about the PlayOneShot() method in the Unity documentation:

<https://docs.unity3d.com/ScriptReference/AudioSource.PlayOneShot.html>

## There's more...

There are some details that you don't want to miss.

### xxx

xxxx

# Creating a basic material with Standard Shader (Specular setup)

In this recipe, we will learn how to create a basic material using the new Standard Shader (Specular Setup), an Albedo map, and a Specular/Smoothness map. The material will feature both metallic and non-metallic parts, with various smoothness levels.

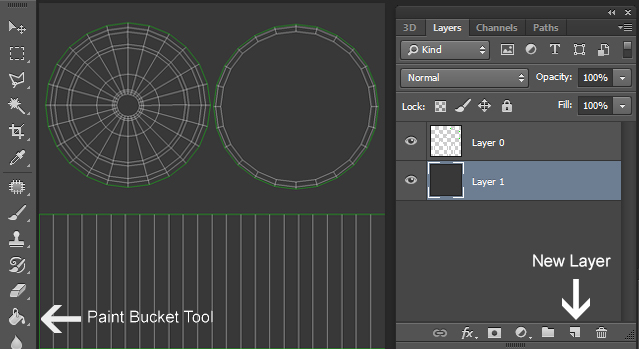
## Getting ready

Two files have been prepared to support this recipe: a 3D model (in FBX format) of a battery, and an UVW template texture (in PNG format) to guide us when creating the diffuse texture map. 3D models and UVW templates can be made with 3D modeling software, such as 3DS MAX, Maya, or Blender. All necessary files are available in the 05\_01 folder.

## How to do it...

To create a basic material, follow these steps:

1. Create a new Unity 3D project and import the battery.FBX and uvw\_template.png files to your project.
2. Place the battery model in the scene by dragging it from the Assets folder, in   
   the Project view, to the Hierarchy view. Select it on the Hierarchy view and make sure, via the Transform component on the Inspector view, that it is positioned at   
   X: 0, Y: 0, Z: 0.
3. Now, let's create a Specular/Smoothness map for our object. Open the image file called uvw\_template.png in your image editor (we'll use Adobe Photoshop to illustrate the next steps). Note that the image file has only a single layer, mostly transparent, containing the UVW mapping templates that we will use as guidelines   
   for our specular map.
4. Create a new layer and place it beneath the one with the guidelines. Fill the new   
   layer with dark gray (R: 56, G: 56, B: 56). The guidelines will be visible at the top   
   of the solid black fill:

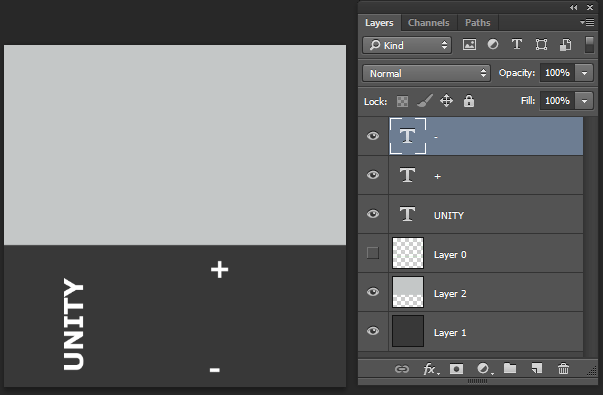


Insert Image 8775\_05\_03.png

1. Create a new layer and select the upper section of the image (the one with the circles). Then, fill that area with a slightly hued light gray (R: 196, G: 199, B: 199):

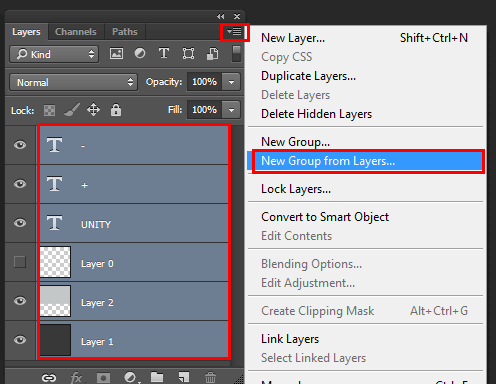
|  |  |  |  |
| --- | --- | --- | --- |
|  |  | The RGB values for our specular map are not arbitrary: Physically-Based Shading takes out most of the guesswork from the mapping process, replacing it with the research for references. In our case, we have used colors based on the reflectance values of iron (the slightly hued light gray) and plastic (the dark gray). Check out the chapter's conclusion for a list of references. |  |

1. Use the text elements in white to add a brand, size, and positive/negative indicators to the battery body. Then, hide the guidelines layer.



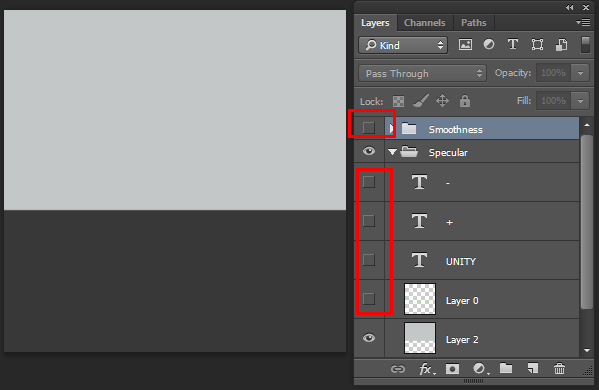
Insert Image 8775\_05\_04.png

1. Select all your layers and organize them into a group (in Photoshop, this can be   
   done by clicking on the drop-down menu in the Layers window and navigating   
   to Window | New Group from Layers…). Name the new group Specular:



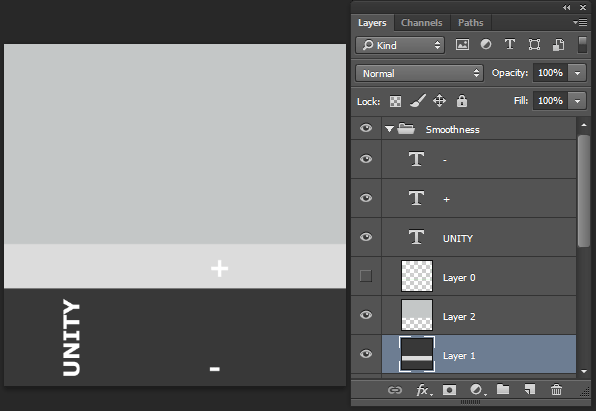
Insert Image 8775\_05\_05.png

1. Duplicate the Specular group (in the Layers window, right-click on the group's   
   name and select Duplicate Group…). Name the duplicated group Smoothness.
2. Hide the Smoothness group. Then, expand the Specular group and hide all   
   text layers:



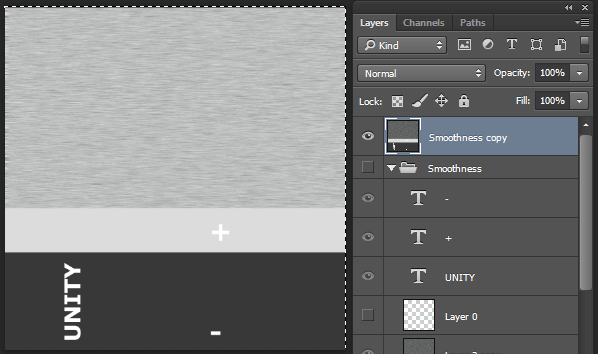
Insert Image 8775\_05\_06.png

1. Unhide the Smoothness group, and hide the Specular group. Select the dark gray layer. Then, make an area selection around the upper region of the battery body, and fill it with light gray (R: 220, G: 220, B: 220). Rescale and rearrange the Text layers   
   if needed:



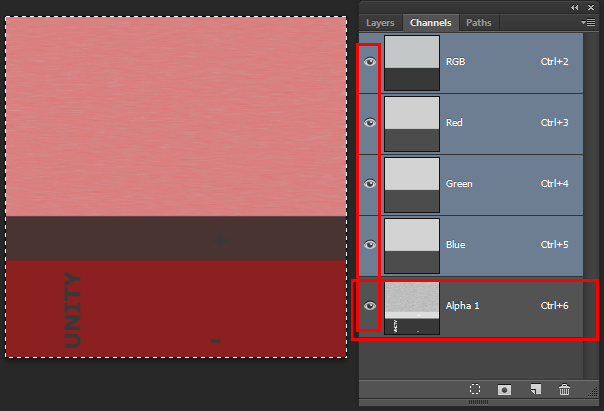
Insert Image 8775\_05\_07.png

1. Duplicate the layer that contains the gray fill for the upper section of the image   
   (the one that went over the circles).
2. To add a brushed quality to this material, add a Noise filter to the duplicated layer   
   (in Photoshop, this can be done by navigating to Filter | Noise | Add Noise...). Use 50% as the Amount and set Monochromatic to true. Then, apply a Motion Blur filter (Filter | Blur | Motion Blur...) using 30 Pixels as the Distance.
3. Duplicate the Smoothness group. Then, select the duplicated group and merge it into a single layer (on the Layers window, right-click on the group's name and select Merge Group).
4. Select the merged layer, use the CTRL + A key combination to select the entire   
   image, and copy it using the CTRL + C keys:



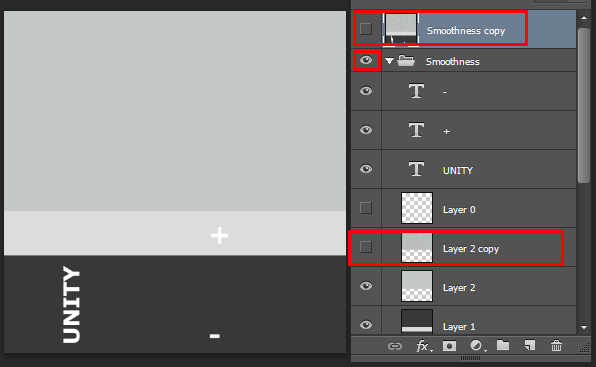
Insert Image 8775\_05\_08.png

1. Hide the merged layer and the Smoothness group. Then, unhide the Specular group.
2. In your image editor, access the image channels window (in Photoshop, this can   
   be done by navigating to Window | Channels). Create a New Channel. This will   
   be our Alpha Channel.
3. Paste the image that you previously copied (from the merged layer) in to the   
   Alpha Channel. Then, set all channels as visible:



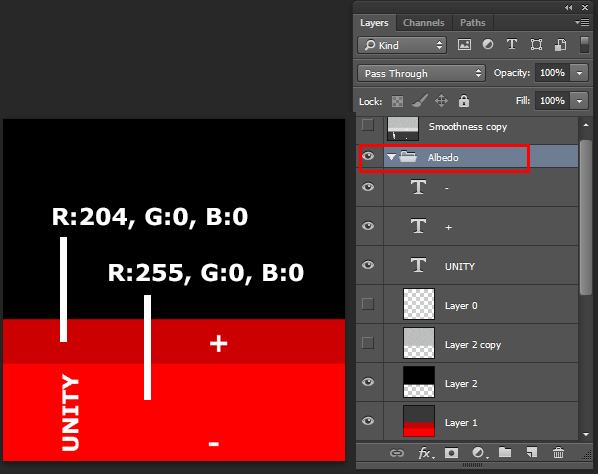
Insert Image 8775\_05\_09.png

1. Save your image in the Project's Assets folder as Battery\_specular, either in Photoshop format (PSD) or TIF format.
2. Now, let's work on the Albedo map. Save a copy of Battery\_specular as Battery\_albedo. Then, from the Channels window, delete the Alpha Channel.
3. From the Layers window, hide the Smoothness copy merged layer, and unhide   
   the Smoothness group. Finally, expand the Smoothness group, and hide the   
   layer where the Noise filter was applied:



Insert Image 8775\_05\_10.png

1. Change the color of the upper rectangle to black. Then, change the light gray area to dark red (R: 204, G: 0, B: 0), and the dark gray to red (R: 255, G: 0, B: 0). Rename the group Albedo and save the file:

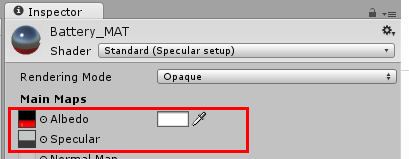


Insert Image 8775\_05\_11.png

1. Go back to Unity and make sure that both files were imported. Then, from the Project view, create a new Material. Name it Battery\_MAT.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | An easy way to create new materials is to access the Project view, click on the Create drop-down menu, and choose Material. |  |

1. Select Battery\_MAT. From the Inspector view, change the Shader to Standard (Specular setup), and make sure that the rendering mode is set to Opaque.
2. Set Battery\_specular as the Specular map, and Battery\_albedo as the Albedo map for Battery\_MAT.
3. Drag the Battery\_MAT material from the Project view and drop it into the   
   battery object in the Hierarchy view:



Insert Image 8775\_05\_12.png

1. Drag the Battery\_MAT material from the Project view and, in the Hierarchy view, drop it into the battery object:



Insert Image 8775\_05\_13.png

## How it works...

Ultimately, the visual aspect of the battery is a combination of three properties of its material: Specular, Smoothness, and Albedo.

To compose the dark red part of the plastic body, for instance, we have mixed the following:

The Specular map (RGB): Very dark grey specularity (for non-metallic appearance)

The Smoothness (the Alpha channel of the Specular map): Light gray   
(for a glossy aspect)

The Albedo map: Dark red (for a dark red color)

The light red portion, on the other hand, combines the following:

The Specular map (RGB): That same dark grey specular

The Smoothness (the Alpha Channel of the Specular map): Dark gray (for a matte aspect)

The Albedo map: Red (for a red color)

Finally, the brushed metal used for the top and bottom covers combines the following:

The Specular map (RGB): Light grey (for a metallic aspect)

The Smoothness (the Alpha Channel of the Specular map): A blurred grey noise pattern (for a brushed aspect)

The Albedo map: Black (for a red color)

Regarding how the image layers are structured, it's good practice to organize your layers   
into groups named after the property that they are related to. As texture maps get more diversified, it can be a good idea to keep a file that contains all the maps for quick   
reference and consistency.

## There's more...

A few things you should have in mind when working with Albedo maps are as follows.

### Setting the texture type for an image file

Since image files can be used for several purposes within Unity (texture maps, GUI textures, cursors, and more), it's a good idea to check if the right Texture Type is assigned to your file. This can be done by selecting the image file in the Project view, and in the Inspector view by using the drop-down menu to select the right Texture Type (in this case, Texture). Please note that other settings can be adjusted, such as Wrap Mode, Filter Mode, and Maximum Size. This last parameter is very useful if you want to keep your texture maps small in size for your game, while still being able to edit them in full size.

### Combining the map with color

When editing a material, the color picker to the right of the Albedo map slot, on the Inspector view, can be used to select the material's color, in case there is no texture map. If a texture map is being used, the selected color will be multiplied to the image, allowing variations on the material's color hue.

# Adapting a basic material from Specular setup to Metallic

For a better understanding of the differences between Metallic and Specular workflows, we will modify the Albedo and Specular/Smoothness maps that are used on a Specular setup material, in order to adapt them to the Metallic workflow. The material to be generated will feature both metallic and non-metallic parts, with various smoothness levels.

## Getting ready

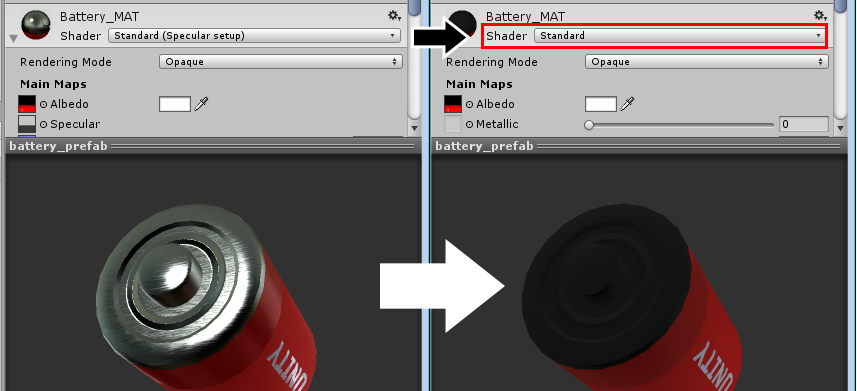
For this recipe, we have prepared a Unity package containing a battery model and its original material (made with Standard Shader—Specular setup). The package includes two image files for the original Albedo and Specular/Smoothness maps which, throughout the recipe, should be adapted for use with the Metallic setup. The package is available in the 1362\_04\_02 folder.

## How to do it...

To create a basic material, follow these steps:

Import the battery\_prefab Unity package into a new project.

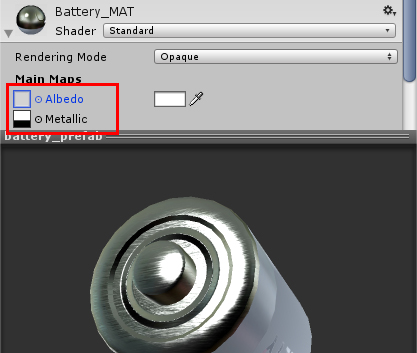
From the Project view, select the battery\_prefab element. Then, from Inspector, access its material (named Battery\_MAT) and change its Shader to Standard   
(as opposed to its current shader—Standard (Specular setup).



From the Project view, find the Battery\_specular map and rename it Battery\_metallic. Open it in your image editor (we'll use Adobe Photoshop to illustrate the following steps).

Find the layer group named Specular and rename it Metallic. Then, fill the light gray layer (named Layer 2, in the Metallic group) with white (R: 255, G: 255, B: 255), and the dark gray layer (named Layer 1, in the Metallic group) with black (R: 0, G: 0, B: 0). Save the file:

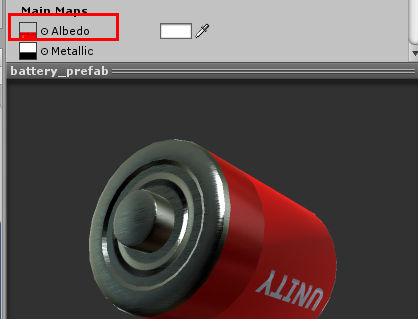
Go back to Unity. From the Inspector view, set the modified Battery\_metallic map as the Metallic map of the Battery\_MAT material. Also, set None as the Albedo map for that material. This will give you an idea of how the material is coming along:



Now, let's adjust the Albedo texture map. From the Project view, locate the Battery\_albedo map and open it in your image editor. Then, use the Paint Bucket tool to fill the black area of Layer 2, in the Albedo group, with light gray (R: 196, G: 199, B: 199). Save the file:

Go back to Unity. From the Inspector view, set the modified Battery\_albedo map as the Albedo map of the Battery\_MAT material.

Your material is ready, combining visual properties based on the different maps that you have edited and assigned.



## How it works...

The visual aspect of the battery is a combination of three properties of its material: Metallic, Smoothness, and Albedo.

To compose the dark red part of the plastic body, for instance, we have mixed the following:

The Metallic map (RGB): Black (for a non-metallic appearance)

The Smoothness (the Alpha Channel of a Metallic map): Light gray   
(for a glossy appearance)

The Albedo map: Dark red (for a dark red color)

The light red portion, on the other hand, combines the following:

The Metallic map (RGB): Black

The Smoothness (the Alpha Channel of the Metallic map): dark gray (for a matte appearance)

Albedo map: red (for a red color)

Finally, the brushed metal used for the top and bottom covers combines the following:

The Metallic map (RGB): white (for a metallic aspect)

The Smoothness (the Alpha Channel of the Metallic map): blurred grey noise pattern (for a brushed appearance);

Albedo map: light grey (for an iron-like appearance)

Remember to organize your layers in to groups named after the property that they are related to.

# Applying Normal maps to a material

Normal maps are generally used to simulate complex geometry that would be too expensive, in terms of computer processing, to be actually represented by the 3D polygons during the game's runtime. Oversimplifying, Normal maps fake complex geometry on low-definition 3D meshes. These maps can be generated either by projecting high-definition 3D meshes onto low-poly ones (a technique usually referred to as baking), or, as will be the case for this recipe, from another texture map.

## Getting ready

For this recipe, we will prepare two texture maps: the Heightmap and the Normal map. The former will be made from simple shapes in an image editor. The latter will be automatically processed from the Heightmap. Although there are a number of tools that can be used to generate Normal maps (see the There is more section of this chapter for a list of resources), we will use a free online tool, Windows and Mac compatible, to generate our texture. Developed by Christian Petry, the NormalMap Online feature can be accessed at   
<http://cpetry.github.io/NormalMap-Online/>.

To help you with this recipe, it's been provided a Unity package, containing a prefab made of  
 a 3D object and its material; and also an UVW template texture (in PNG format) to guide you when creating the diffuse texture map. All the files are in the 1362\_04\_03 folder.

## How to do it...

To apply a Normal map to a material, follow these steps:

Import the 1362\_04\_03.unitypackage file to your project. Select the batteryPrefab object from the Assets | 1362\_04\_03 folder, in the Project view. After comparing it with some reference photos, inform yourself about the features that should be reproduced by the Normal map: (A) a bumpy ring at the top; and (B) some circular creases at the bottom, shown in the following image:

In an image editor, open uvw\_template.png. Create a new layer, fill it with grey (RGB: 128), and position it below the pre-existing layer as shown:

On a separate layer, draw a white circle centered on the battery's top. Then, on another layer, draw a black circle, centered on the battery's bottom, as shown below:

If you have used vector shapes to make the circles, rasterize their layers (in Adobe Photoshop, right-click on the layer's name and select the Rasterize Layer option from the context menu).

Blur the white circle (in Photoshop, this can be done by navigating to Filter | Blur | Gaussian Blur...). Use 4,0 pixels as the Radius.

Hide the UVW template layer and save the image as Battery\_height.png.

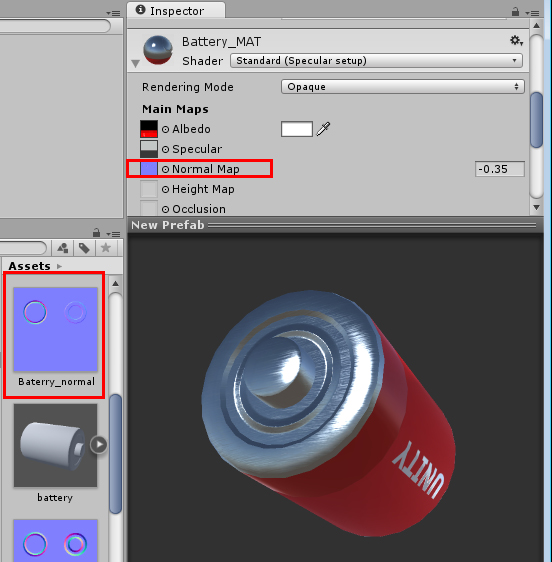
If you want to convert the Heightmap directly from Unity, import it to your project. Select it from the Project view and, from the Inspector view, change its Texture   
Type to Normal map. Check the Create from Grayscale option, adjust Bumpiness and Filtering as you like, and click on Apply to save the changes:

To convert your Heightmap externally, access the website at <http://cpetry.github.io/NormalMap-Online/>. Then, drag the HEIGHT\_battery.png   
file to the appropriate image slot. Feel free to play with the Strength, Level and   
Blur/Sharp parameters:

Save the resulting Normal map as Battery\_normal.jpg and add it to your   
Unity project.

In Unity, select Battery\_normal from the Project view. Then, from the Inspector view, change its Texture Type to Normal, leaving the Create from Grayscale box unchecked. Click on Apply to save the changes:

In the Project view, select batteryPrefab. Then, in the Inspector view, scroll down to the Material component, and assign Battery\_normal to the Normal Map slot. To adjust its intensity and direction, change its value to -0.35:



## How it works...

The Normal map was calculated from the grey values on the Heightmap, where the lighter tones were interpreted as recesses (applied to the top of the battery), and the darker tones as bulges (applied to the bottom). Since the desired output was actually the opposite, it was necessary to adjust the Normal map to a negative value (-0.35). Another possible solution   
to the issue would have been to redraw the Heightmap and switch the colors for the white   
and black circles.

## There's more...

If you wish to explore Normal mapping beyond the limitations of NormalMap Online, there is an ever-growing list of full-featured software that can produce Normal maps (and much more). Here are some resources that you might want to check out:

CrazyBump is a standalone tool for Windows and Mac, which is available at   
<http://www.crazybump.com>

nDo is a Photoshop plugin by Quixel (Windows only), available at <http://quixel.se/ndo>

GIMP normalmap Plugin, available for Windows only, is available at <http://code.google.com/p/gimp-normalmap/>

NVIDIA Texture Tools for Adobe Photoshop, available for Windows only, is available at <http://developer.nvidia.com/nvidia-texture-tools-adobe-photoshop>

Bitmap2Material is an amazing texture generating tool from Allegorithmic, which is available at <http://www.allegorithmic.com/>

# Adding Transparency and Emission maps to a material

The Emission property can be used to simulate a variety of self-illuminated objects, from the LEDs of mobile displays to futuristic Tron suits. Transparency, on the other hand, can make the diffuse color of a material more or less visible. In this recipe, you will learn how to configure these properties to produce a toy's cardboard packaging that features a plastic case and glow-in-the-dark text.

## Getting ready

For this recipe, we have prepared a Unity package containing a prefab made of a 3D object, its material, and its respective diffused texture map (in PNG format). All files are in the 1362\_04\_04 folder.

## How to do it...

To add transparency and color emissions to a material, follow these steps:

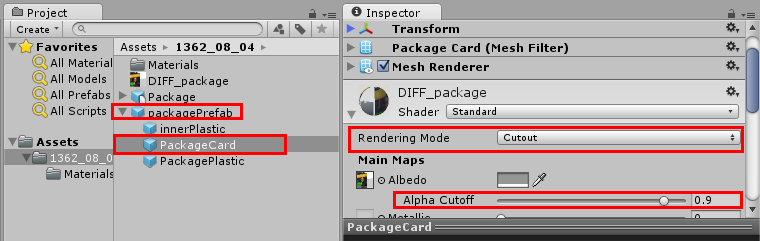
Import TransparencyEmission.unitypackage to your project. Select the DIFF\_package texture from the Assets folder, in the Project view. Then, open it in your image editor.

First, we will add transparency to the image by deleting the white areas around the package (and the hang hole). Make a selection of those areas (in Photoshop, this can be done with the Magic Wand Tool).

Make sure you unlock the Background layer by clicking on the lock icon, to the left   
of the layer's name, as shown below:

Delete the previously-made selection (this can be done in Photoshop by pressing   
the Delete key). The background of the image should be transparent, as shown below.   
Save the file:

Back in Unity, in the Assets folder, expand the packagePrefab list and select the PackageCard object. Now, in the Inspector view, scroll down to the Material component and change its Rendering Mode to Cutout, and adjust its Alpha Cutoff   
to 0.9:



|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Choosing Cutout means that your material can be either invisible or fully visible, not allowing for semi-transparency. The Alpha Cutoff is used to get rid of unwanted pixels around the transparent borders. |  |

From the expanded packagePrefab, select the PackagePlastic object. In the Inspector view, scroll down to the Material component and change its Rendering Mode to Transparent. Then, use the Diffuse color picker to change the color's RGB values to 56, and Alpha to 25. Also, change the Smoothness level to 0.9:

Now that we have taken care of our transparency needs, we need to work on the Emission map. From the Assets folder, duplicate the DIFF\_package texture, rename it EMI\_package, and open it in your image editor.

Select all the characters from the Ms. Laser inscription and the green star   
(in Photoshop, this can be done with the Magic Wand tool, keeping the Shift   
key pressed while selecting multiple areas).

Copy and paste your selection into a new layer. Then, select it and apply a Noise filter to it (in Photoshop, this can be done by navigating to Filter | Noise | Add Noise...). Use 50% as the value.

Create a new layer and, using a tool such as the Paint Bucket, fill it with black   
(R: 0, G: 0, B: 0). Place this black layer beneath the one with the colored elements.

Flatten your image (in Photoshop this can be done by navigating to Layer | Flatten Image) and save your file:



Back in Unity, in the Assets folder, expand packagePrefab and select the PackageCard object. Now, in the Inspector view, scroll down to the Material component and assign the EMI\_package texture to its Emission slot. Then, change the Emission color slot to white (R: 255; G: 255; B: 255), and turn down its intensity to 0.25, as shown in the following screenshot. Also, change its GI option to None, so that its glow won't be added to the Lightmaps or influence the illumination in real time:

Place an instance of the packagePrefab in your scene and check out the results. Your material is ready:



## How it works...

Unity is able to read four channels of a texture map: R (Red), G (Green), B (Blue) and A (Alpha). When set to Transparent or Cutout, the Alpha channel of the diffuse texture map sets the transparency of the material according to each pixel's brightness level (the Cutout mode will not render semi-transparency—only fully visible or invisible pixels). You might have noticed that we didn't add an Alpha channel—this is because Photoshop exports the PNG's Alpha map, based on its transparency. To help you visualize the Alpha map, the 1362\_04\_04 folder contains a DIFF\_packageFinal.TIF file featuring an Alpha map that works exactly in the same way as the PNG file that we have generated:



Regarding the Emission texture map, Unity assigns its RGB colors to the material, combining   
them with the appropriate color selection slot, and also allowing adjustments to the intensity of that Emission.

## There's more...

Let look at a little more information on Transparency and Emission.

### Using texture maps with Transparent Mode

Please note that you can use a bitmap texture for the Diffuse map in the Transparent   
render mode. In this case, RGB values will be interpreted as the Diffuse color, while the   
Alpha will be used to determine that pixel's transparency (in this case, semi-transparent materials are allowed).

### Avoiding issues with the semi-transparent objects

You might have noticed that the plastic case was made from two objects (PackagePlastic   
and innerPlastic). This was done to avoid z-sorting problems, where faces are rendered in front of other geometry when they should be behind it. Having multiple meshes instead of a single one allows these faces to be correctly sorted for rendering. Materials in the Cutout mode are not affected by this problem, though.

### Emitting light over other objects

The Emission value can be used to calculate the material's light projection over other objects when using Lightmaps.

# Highlighting materials at mouse over

Changing the color of an object at runtime can be a very effective way of letting players know that they can interact with it. This is very useful in a number of game genres, such as puzzles and point-and-click adventures, and it can also be used to create 3D user interfaces.

## Getting ready

For this recipe, we'll use objects created directly in Unity. Alternatively, you can use any 3D model you like.

## How to do it...

To highlight a material at mouse-over, follow these steps:

Create a new 3D project, and add a Cube to the scene (from the Hierarchy view, navigate to Create | 3D Object | Cube).

From the Project view, click the Create drop-down menu and choose Material.   
Name it HighlightMaterial.

Select HighlightMaterial, and, from the Inspector view, change its Albedo color   
to gray (R: 135, G: 135, B: 135), its Emission intensity to 1, as shown in the screenshot below, and it's Emission color to R: 1, G: 1, B: 1:

Assign HighlightMaterial to the Cube that you previously created.

From the Project view, click on the Create drop-down menu and choose   
C# Script. Rename it HighlightObject and open it in your editor.

Replace everything with the following code:

using UnityEngine;

using System.Collections;

public class HighlightObject : MonoBehaviour{

private Color initialColor;

public bool noEmissionAtStart = true;

public Color highlightColor = Color.red;

public Color mousedownColor = Color.green;

private bool mouseon = false;

private Renderer myRenderer;

void Start() {

myRenderer = GetComponent<Renderer>();

if (noEmissionAtStart)

initialColor = Color.black;

else

initialColor = myRenderer.material.GetColor("\_EmissionColor");

}

void OnMouseEnter(){

mouseon = true;

myRenderer.material.SetColor("\_EmissionColor", highlightColor);

}

void OnMouseExit(){

mouseon = false;

myRenderer.material.SetColor("\_EmissionColor",initialColor);

}

void OnMouseDown(){

myRenderer.material.SetColor("\_EmissionColor",   
mousedownColor);

}

void OnMouseUp(){

if (mouseon)

myRenderer.material.SetColor("\_EmissionColor",   
highlightColor);

else

myRenderer.material.SetColor("\_EmissionColor", initialColor);

}

}

Save your script and attach it to the Cube.

Select the Cube, and, in the Inspector view, set the Highlight Color and Mousedown Color values to any colors that you would like:

If you are using the script with your own imported 3D mesh, please make   
sure you add a Collider component to your object.

Test the scene. The Cube will be highlighted red when the mouse is over it   
(and green when clicked on).

## How it works...

The cube is automatically sent the mouse enter/exit/down/up events as the user moves the mouse pointer over and away from the part of the screen where the cube is visible. Our script adds a behavior to the cube when these events are detected. The Start() method gets a reference to the Renderer component of the GameObject that the script has been added to, and stores it in the variable myRenderer (note that 'renderer' already has a meaning in Unity so it is not appropriate as a private variable name for this script). The Boolean variable called mouseon records whether or not the mouse pointer is currently over the object. When the mouse button is released, we use the mouseon variable to decide whether to change the cube back to its initial color (mouseon FALSE, so the mouse pointer is away from the cube), or back to its highlight color (mouseon TRUE, so the mouse pointer is over the cube).

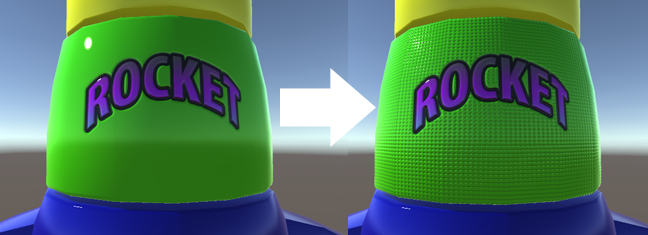
The reason we needed to change the material's original Emission color to ultra, dark gray is that leaving it black would have caused Unity to optimize the Shader by removing the Emission property from the material. Our script wouldn't have worked if this had happened.

## There's more...

You can achieve other interesting results by changing the other properties of your material   
(by changing the \_EmissionColor script to \_Color or "\_SpecularColor, for instance). For a full list of properties, select your material, and, in the Inspector view, click on the Edit button, at the side of the Shader drop-down menu.

# Adding Detail maps to a material

When creating a large object, there is not only the desire to texture it as a whole, but also to add details that can make it look better at closer distances. To overcome the need for gigantic texture maps, the use of Detail maps can make a real difference. In this recipe, we will add Detail maps to a rocket toy by applying a Detail mask and a Detail Normal map. In our case, we want to add a textured quality (and a stripe pattern) to the green plastic, except in the region where there is a battery compartment and the toy's logo:



## Getting ready

For this recipe, we have prepared a Unity package, containing the prefab for a rocket toy.   
The prefab includes the 3D model and a material, featuring a Diffuse map and a Normal   
map (made from a Heightmap). The file can be found in the 1362\_04\_06 folder.

## How to do it...

To add the Detail maps to your object, follow these steps:

Import the rocket.unitypackage file into your project. Then, select the   
prefab named rocketToy from the Assets folder, in the Project view, and   
place it in your scene.

From the Hierarchy view, expand the rocketToy GameObject and select its child called rocketLevel1. Then, scroll down the Inspector view to the Material component. Observe that it uses the DIFF\_ship texture as the Diffuse map. Duplicate this file and rename the new copy COPY\_ship.

Open COPY\_ship in your image editor. Select all the solid green pixels around   
the logo and battery compartment (in Photoshop, this can be done with the Magic Wand tool, keeping the Shift key pressed while selecting multiple areas):

Keeping your selection active, access the image Channels window (in Photoshop,   
this can be done by navigating to Window | Channels). Click on New Channel.   
This will be our Alpha channel:

Hide the Red, Green and Blue channels. Select the Alpha channel and paint the selection white. Then, select the area of the battery compartment and paint it grey   
(R, G and B: 100):

Save it in the TIFF format as MASK\_ship.TIF, in the Assets folder. Make sure that you include Alpha Channels:

Now that we have the mask, let's create a diffuse map for our detail. In your image editor, create a new image with the following dimensions: width: 64, and height: 64.

Fill the new image with grey (R, G and B: 128). Then, use shapes or rectangular fills to create a dark grey (R, G, and B: 100) horizontal line that is about 16 pixels tall:

Save the image as DIFF\_detail.PNG in the Assets folder.

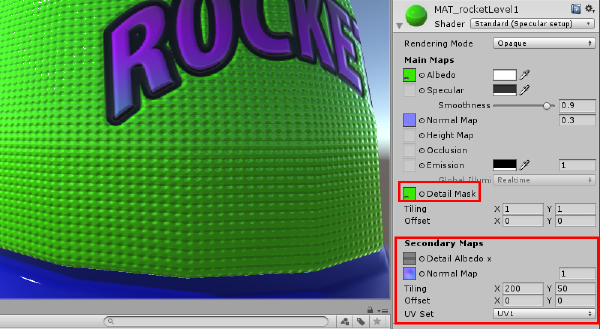
Create a new 64 x 64 image. Use a Gradient tool to create a black and white Radial Gradient (in Photoshop, this can be done with the Gradient Tool in Radial mode):

Save the image as HEIGHT\_detail.PNG in the Assets folder.

Go back to Unity. From the Assets folder, select HEIGHT\_detail. Then, from   
the Inspector view, change its Texture Type to Normal map, check the Create   
from Grayscale option, adjust Bumpiness to 0.25, and set Filtering to smooth. Click on Apply to save the changes:

From the Hierarchy view, expand the rocketToy GameObject and select its child called rocketLevel1. Then, scroll down the Inspector view to the Material component. Assign MASK\_ship to the Detail Mask slot; DIFF\_detail as Secondary Maps | Detail Diffuse x 2; and HEIGHT\_detail as Secondary Maps | Normal Map. Also, turn the Normal Map intensity down to 0.6.

In the Secondary Maps section, change the Tiling values to X: 200, and Y: 50. You might notice that the pattern is not seamless. This is because we are using the same UV Set from our Diffuse texture. However, the object has been assigned to two different UV channels (back when it was being modeled). While UV channel 1 contains the mapping for our Diffuse map, UV channel 2 uses a basic cylindrical mapping. We need to change the Secondary Maps section's UV Set from UV0 to UV1. The Detail map for your material is ready:



## How it works...

When in use, Secondary Maps are blended onto the material's primary Diffuse and Normal maps (that's why our object is green even after the Detail Diffuse is applied: the grey tones are superimposed on the original Diffuse texture). By using a Detail Mask, the artist defines which areas of the object should be affected by Secondary Maps. This is great for customization, and also for creating nuances (like the semi-bumped battery compartment   
in our example).

Another helpful feature is the possibility of using a separate UV channel for Details maps   
and Tiling. Besides adding variation to texture mapping, this allows us to paint the details   
that can be perceived even at a very close distance by dramatically enhancing the visual quality of our objects.

# Fading the transparency of a material

In this recipe, we will create an object that, once clicked, fades out and disappears. However, the script will be flexible enough to allow us adjust the initial and final alpha values. Plus, we will have the option of making the object self-destructible when turned invisible.

## How to do it...

Follow these steps:

Add a Sphere to your scene by accessing the GameObject | 3D Object | Sphere menu.

Select the Sphere and make sure it has a collider (if you are using a custom 3D object, you might have to add a collider through the Components | Physics menu).

Create a new material. The easiest way to do that is to access the Project view,   
click the Create drop-down menu, and choose Material.

Rename your new material. For this example, let's call it Fade\_MAT.

Select your material. From the Inspector view, use the drop-down menu to change   
its Rendering Mode to Fade:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | The Fade rendering mode is specifically designed for situations like this. Other rendering modes, such as Transparent, will fade turn the Albedo color transparent, but not the specular highlights nor the reflections, in which case the object will still be visible. |  |

Apply the FadeMaterial to Sphere by dragging it from the Project view into the Sphere Game Object name in the Hierarchy view.

From the Project view, click the Create drop down menu and choose C# Script. Rename it as FadeMaterial and open it in your editor.

Replace your script with the code below:

using UnityEngine;

using System.Collections;

public class FadeMaterial : MonoBehaviour {

public float fadeDuration = 1.0f;

public bool useMaterialAlpha = false;

public float alphaStart = 1.0f;

public float alphaEnd = 0.0f;

public bool destroyInvisibleObject = true;

private bool isFading = false;

private float alphaDiff;

private float startTime;

private Renderer rend;

private Color fadeColor;

void Start () {

rend = GetComponent<Renderer>();

fadeColor = rend.material.color;

if (!useMaterialAlpha) {

fadeColor.a = alphaStart;

} else {

alphaStart = fadeColor.a;

}

rend.material.color = fadeColor;

alphaDiff = alphaStart - alphaEnd;

}

void Update () {

if(isFading){

var elapsedTime = Time.time - startTime;

if(elapsedTime <= fadeDuration){

var fadeProgress = elapsedTime / fadeDuration;

var alphaChange = fadeProgress \* alphaDiff;

fadeColor.a = alphaStart - alphaChange;

rend.material.color = fadeColor;

} else {

fadeColor.a = alphaEnd;

rend.material.color = fadeColor;

if(destroyInvisibleObject)

Destroy (gameObject);

isFading = false;

}

}

}

void OnMouseUp(){

FadeAlpha();

}

public void FadeAlpha(){

isFading = true;

startTime = Time.time;

}

}

Save your script and apply it to the Sphere Game.

Play your scene and click on the Sphere to see it fade away and self-destruct.

## How it works...

Since the opaqueness of the material using a transparent Shader is determined by the alpha value of its main color, all we need to do in order to fade it is changing that value over a given amount of time. This transformation is expressed, in our script, on the following lines of code:

var fadeProgress = elapsedTime / fadeDuration;

var alphaChange = fadeProgress \* alphaDiff;

fadeColor.a = alphaStart - alphaChange;

rend.material.color = fadeColor;

## There's more...

You could call the FadeAlpha function in other circumstances (such as a Rigidbody collision, for instance). In fact, you could even call it from another Game Object's script   
by using the GetComponent command. The script would be something like:

GameObject.Find("Sphere").GetComponent<FadeMaterial>().FadeAlpha();

# Playing videos inside a scene

TV sets, projectors, monitors.... If you want complex animated materials in your level, you can play video files as texture maps. In this recipe, we will learn how to apply a video texture to a cube. We will also implement a simple control scheme that plays or pauses the video whenever that cube is clicked on.

## Getting ready

Unity imports video files through Apple Quicktime. If you don't have it installed in your machine, please download it at <http://www.apple.com/quicktime/download/>.

Also, if you need a video file to follow this recipe, please use the videoTexture.mov included in the folder 1632\_04\_08.

## How to do it...

Follow these steps:

Add a cube to the scene through the GameObject | 3D Object | Cube menu.

Import the provided videoTexture.mov file.

From the Project view, use the Create drop-down menu to create a new   
Material. Rename it Video\_MAT and, from the Inspector view, change its   
Shader to Unlit/Texture:

Apply videoTexture to the texture slot of Video\_MAT by dragging it from the Project view into the appropriate slot.

Apply the Video\_MAT to the Cube you have previously created.

Expand videoTexture on the Project view to reveal its correspondent Audio Clip. Then, apply that audio clip to the Cube (you can do it by dragging it from the Project view to the Cube in the Hierarchy view, or a Scene view).

Select the Cube. Make sure there is a Collider component visible from the Inspector view. In case there isn't one, add it via the Component | Physics | Box Collider menu. Colliders are needed for mouse collision detection.

Now we need to create a script for controlling the movie texture and associated audio clip. From Project view, use the Create drop-down menu to add a C# Script. Name it PlayVideo.

Open the script and replace it with the following code:

using UnityEngine;

using System.Collections;

[RequireComponent(typeof(AudioSource))]

public class PlayVideo : MonoBehaviour {

public bool loop = true;

public bool playFromStart = true;

public MovieTexture video;

public AudioClip audioClip;

private AudioSource audio;

void Start () {

audio = GetComponent<AudioSource> ();

if (!video)

video = GetComponent<Renderer>().material.mainTexture as   
MovieTexture;

if (!audioClip)

audioClip = audio.clip;

video.Stop ();

audio.Stop ();

video.loop = loop;

audio.loop = loop;

if(playFromStart)

ControlMovie();

}

void OnMouseUp(){

ControlMovie();

}

public void ControlMovie(){

if(video.isPlaying){

video.Pause();

audio.Pause();

} else {

video.Play();

audio.Play();

}

}

}

Save your script and attach it to the Cube.

Test your scene. You should be able to see the movie being played in the cube face, and also pause/play it by clicking on it.

## How it works...

By default, our script makes the movie texture play in loop mode. There is, however, a Boolean variable than can be changed through the Inspector panel, where it is represented by a check box. Likewise, there is a check box that can be used to prevent the movie from playing when the level starts.

## There's more...

There are some other movie texture commands and parameters that can be played with. Don't forget to check out Unity's scripting guide at <http://docs.unity3d.com/Documentation/ScriptReference/MovieTexture.html>.

## Getting ready

We have included xxx inside the 05\_01 folder.

## How to do it...

To play xxx do the following:

1. Create a new Unity 2D project and import the sound clip files.
2. Create a C# script class, PlaySounds, in a new folder \_Scripts, containing the following code, and add an instance as a scripted component to the Main Camera:

using UnityEngine;

[RequireComponent(typeof(AudioSource))]

public class PlaySounds : MonoBehaviour

{

public AudioClip clipEatCherry;

public AudioClip clipExtraLife;

private AudioSource audioAudioSource;

void Awake()

{

audioAudioSource = GetComponent<AudioSource>();

}

void Update()

{

if (Input.GetKey(KeyCode.UpArrow))

{

audioAudioSource.PlayOneShot(clipEatCherry);

}

if (Input.GetKey(KeyCode.DownArrow))

{

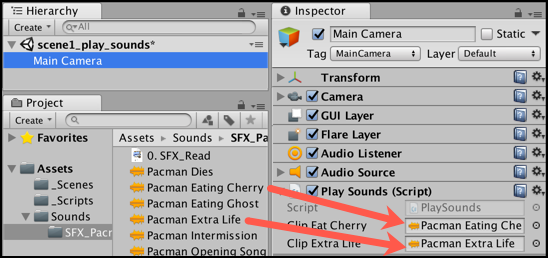
audioAudioSource.PlayOneShot(clipExtraLife);

}

}

}

1. Ensure that the MainCamera GameObject is selected in the Hierarchy. Then, in the Inspector panel, drag the Pacman Eating Cherry sound clip from the Project panel into the public AudioClip variable Pacman Eating Cherry in the PlaySounds (Script) scripted component. Repeat this procedure for the Pacman Extra Life sound clip. These steps are illustrated in the screenshot.



Insert Image B08775\_04\_03.png

1. Run the scene, and press the UP and DOWN arrow keys to play the different sound effects.

## How it works...

You created a C# script class xxx

NOTE: Cannot pause/interrogate sounds played with PlayOneShot

While great for short, one-off sound effects, a limitation of the PlayOneShot() method is that you cannot then interrogate the status of the playing sound (has it finished, at what point is it playing etc.). Nor can you pause/restart a sound played with PlayOneShot(). For such detailed control of sounds each sound needs its own AudioSource component.

Learn more about the PlayOneShot() method in the Unity documentation:

<https://docs.unity3d.com/ScriptReference/AudioSource.PlayOneShot.html>

## There's more...

There are some details that you don't want to miss.

### xxx

xxxx

# xxx

xxx

## Getting ready

We have included xxx inside the 05\_01 folder.

## How to do it...

To play xxx do the following:

1. Create a new Unity 2D project and import the sound clip files.
2. Create a C# script class, PlaySounds, in a new folder \_Scripts, containing the following code, and add an instance as a scripted component to the Main Camera:

using UnityEngine;

[RequireComponent(typeof(AudioSource))]

public class PlaySounds : MonoBehaviour

{

public AudioClip clipEatCherry;

public AudioClip clipExtraLife;

private AudioSource audioAudioSource;

void Awake()

{

audioAudioSource = GetComponent<AudioSource>();

}

void Update()

{

if (Input.GetKey(KeyCode.UpArrow))

{

audioAudioSource.PlayOneShot(clipEatCherry);

}

if (Input.GetKey(KeyCode.DownArrow))

{

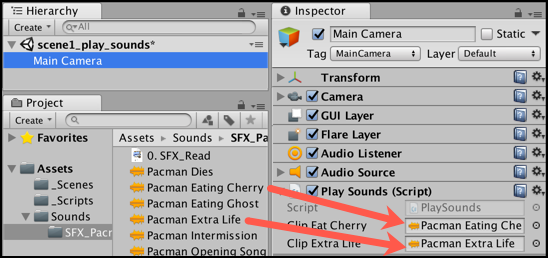
audioAudioSource.PlayOneShot(clipExtraLife);

}

}

}

1. Ensure that the MainCamera GameObject is selected in the Hierarchy. Then, in the Inspector panel, drag the Pacman Eating Cherry sound clip from the Project panel into the public AudioClip variable Pacman Eating Cherry in the PlaySounds (Script) scripted component. Repeat this procedure for the Pacman Extra Life sound clip. These steps are illustrated in the screenshot.



Insert Image B08775\_04\_03.png

1. Run the scene, and press the UP and DOWN arrow keys to play the different sound effects.

## How it works...

You created a C# script class xxx

NOTE: Cannot pause/interrogate sounds played with PlayOneShot

While great for short, one-off sound effects, a limitation of the PlayOneShot() method is that you cannot then interrogate the status of the playing sound (has it finished, at what point is it playing etc.). Nor can you pause/restart a sound played with PlayOneShot(). For such detailed control of sounds each sound needs its own AudioSource component.

Learn more about the PlayOneShot() method in the Unity documentation:

<https://docs.unity3d.com/ScriptReference/AudioSource.PlayOneShot.html>

## There's more...

There are some details that you don't want to miss.

### xxx

xxxx

# xxx

xxx

## Getting ready

We have included xxx inside the 05\_01 folder.

## How to do it...

To play xxx do the following:

1. Create a new Unity 2D project and import the sound clip files.
2. Create a C# script class, PlaySounds, in a new folder \_Scripts, containing the following code, and add an instance as a scripted component to the Main Camera:

using UnityEngine;

[RequireComponent(typeof(AudioSource))]

public class PlaySounds : MonoBehaviour

{

public AudioClip clipEatCherry;

public AudioClip clipExtraLife;

private AudioSource audioAudioSource;

void Awake()

{

audioAudioSource = GetComponent<AudioSource>();

}

void Update()

{

if (Input.GetKey(KeyCode.UpArrow))

{

audioAudioSource.PlayOneShot(clipEatCherry);

}

if (Input.GetKey(KeyCode.DownArrow))

{

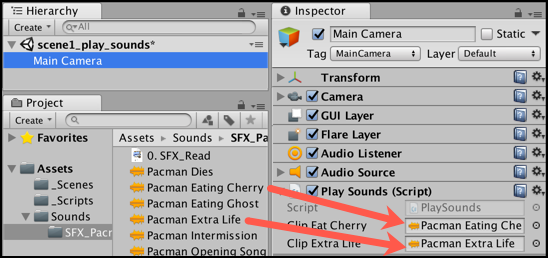
audioAudioSource.PlayOneShot(clipExtraLife);

}

}

}

1. Ensure that the MainCamera GameObject is selected in the Hierarchy. Then, in the Inspector panel, drag the Pacman Eating Cherry sound clip from the Project panel into the public AudioClip variable Pacman Eating Cherry in the PlaySounds (Script) scripted component. Repeat this procedure for the Pacman Extra Life sound clip. These steps are illustrated in the screenshot.



Insert Image B08775\_04\_03.png

1. Run the scene, and press the UP and DOWN arrow keys to play the different sound effects.

## How it works...

You created a C# script class xxx

NOTE: Cannot pause/interrogate sounds played with PlayOneShot

While great for short, one-off sound effects, a limitation of the PlayOneShot() method is that you cannot then interrogate the status of the playing sound (has it finished, at what point is it playing etc.). Nor can you pause/restart a sound played with PlayOneShot(). For such detailed control of sounds each sound needs its own AudioSource component.

Learn more about the PlayOneShot() method in the Unity documentation:

<https://docs.unity3d.com/ScriptReference/AudioSource.PlayOneShot.html>

## There's more...

There are some details that you don't want to miss.

### xxx

xxxx

# Conclusion

This chapter has covered a number of techniques used to create, often manually and sometimes automatically, texture maps that are capable of giving distinctive features to materials. Hopefully, you are now more confident about working with Unity's new Physically-Based Shading, which is capable of understanding differences between available workflows, is aware of the role of each material property, and is ready to make better-looking materials for your games. We have also explored ways of changing the properties of materials during runtime by accessing an object's material via script.

## Resources

Physically-Based Rendering is a complex (and current) topic, so it's a good idea to study it a bit by familiarizing yourself with the tools and concepts behind it. To help you with this task, we have included a non-exhaustive list of resources below that you should take a look at.

### References

Here's a list of interesting, detailed material on Physically-Based Rendering (within and outside Unity):

The Comprehensive PBR Guide Volumes 1 and 2 by Wes McDermott (Allegorithmic), available at <http://www.allegorithmic.com/pbr-guide>. This guide takes an in-depth look at the practical and theoretical aspects of PBR, including great analysis of possible workflows.

Mastering Physically Based Shading in Unity 5 by Renaldas Zioma (Unity), Erland Körner (Unity), and Wes McDermott (Allegorithmic), is available at <http://www.slideshare.net/RenaldasZioma/unite2014-mastering-physically-based-shading-in-unity-5>. This is a detailed presentation about using PBS   
in Unity. Originally presented at the Unite 2014 conference, it contains some   
out-of-date information, but, nevertheless, it is still worth taking a look at.

Physically Based Shading in Unity 5 by Aras Pranckevičius, from Unity, is available   
at <http://aras-p.info/texts/talks.html>. Slides and notes from a presentation on the subject are given at the GDC.

Tutorial: Physically Based Rendering, And You Can Too! by Joe "EarthQuake" Wilson   
is available at <http://www.marmoset.co/toolbag/learn/pbr-practice>.   
It is a great overview from the makers of Marmoset Toolbag and Skyshop.

Polycount PBR Wiki, which is available at <http://wiki.polycount.com/wiki/PBR>, is a list of resources compiled by the Polycount community.

### Tools

This is a new generation of texturing software for you to check out, in case you haven't yet:

Substance Painter is a 3D painting application from Allegorithmic. It is available at <http://www.allegorithmic.com/products/substance-painter>. Again, it's worth mentioning that Allegorithmic products won't make use of Unity's Standard Shader, relying instead on substance files that are natively supported by Unity.

Bitmap2Material creates full-featured materials (including normal maps,   
specular maps, and more) from a single bitmap image. Also, it is from   
Allegorithmic, and it is available at <http://www.allegorithmic.com/products/bitmap2material>

Quixel DDO is a plugin for creating PBR-ready textures in Adobe Photoshop.   
From Quixel, it is available at <http://www.quixel.se/ddo>

Quixel NDO is a plugin for creating Normal maps in Adobe Photoshop.   
From Quixel, it is available at <http://www.quixel.se/ndo>

Mari is a 3D painting tool from The Foundry. It is available at   
<http://www.thefoundry.co.uk/products/mari/>