4

Creating Maps and Materials

In this chapter, we will cover:

- Creating a basic material with Standard shader (Specular Setup)
- Adapting a basic material from Specular setup to Metallic
- Applying Normal maps to a material
- Adding Transparency and Emission maps to a material
- Highlighting materials at mouse over
- Adding Detail maps to a material
- Fading the transparency of a material
- Playing videos inside a scene

Introduction

Unity 5 brings on new **Physically Based Shaders**. Physically Based Rendering is a technique that simulates the appearance of materials based on how light reacts with that material (more specifically, the *matter* from which that material is made of) in the real world. Such technique allows for more realistic and consistent materials – so your creations in Unity should look better than ever. Creating materials in Unity also 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 for browsing dropdown menus in search for 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 texture maps have been assigned.

For a deep understanding on Physically Based Rendering, we recommend you take a look at **The Comprehensive PBR Guide** written by Wes McDermott from Allegorithmic, freely available in two volumes at: `www.allegorithmic.com/pbr-guide`. Allegorithmic's guide contains invaluable information on PBR theory and techniques,

having been a fundamental reference for this chapter. Other great resources we´d recommend you to take a look at are:

Mastering Physically Based Shading in Unity 5, by Renaldas Zioma (Unity), Erland Körner (Unity) and Wes McDermott (Allegorithmic), available at www.slideshare.net/RenaldasZioma/unite2014-mastering-physically-based-shading-in-unity-5.

Physically Based Shading in Unity, by Aras Pranckevičius (Unity), available at aras-p.info/texts/files/201403-GDC_UnityPhysicallyBasedShading_notes.pdf.

Creating and saving texture Maps

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

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. **TIF** is also a good option for preserving the alpha channel -- although the image will be flattened. **PNG** is also a great option, but please note that Photoshop doesn't handle PNGs Alpha channel independently of the transparency, possibly compromising the material's appearance.

Finally, a word of advice: although it's possible to manually create texture maps for our materials using traditional image editing software, new tools such as Allegorithmic'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 cannot recommend enough that you at least try such tools. Note, however, that products from **Allegorithmic** won't make use of Unity's Standard shader, relying on **substance** files (which are natively supported by Unity)

The Big Picture

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

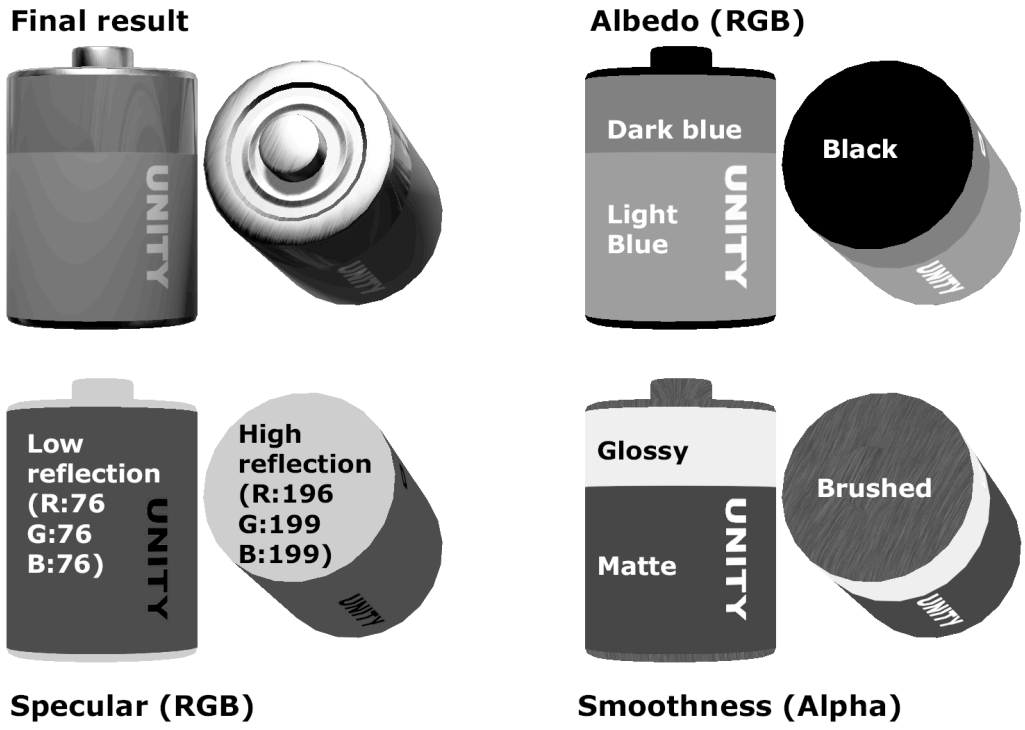
Specular workflow

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.

Albedo: The material's diffuse color. Plain and simple, this is how you usually describe the appearance of the material (the British flag is blue, red and white; Ferrari's logo is a black horse in a yellow setting; some sunglasses lenses are semi-transparent gradients, etc.). This description, however, can be deceptive. Purely metallic objects (such as aluminum, chrome, gold, etc.) should have black as their diffuse color – their colors, as we perceive them, are 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: 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 (for plastic, for instance, you should work with a grey value around 59). Metallic objects, on the other way, feature very light values, and also a bit yellow-ish in hue.

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



Insert image 1362OT_04_39.png

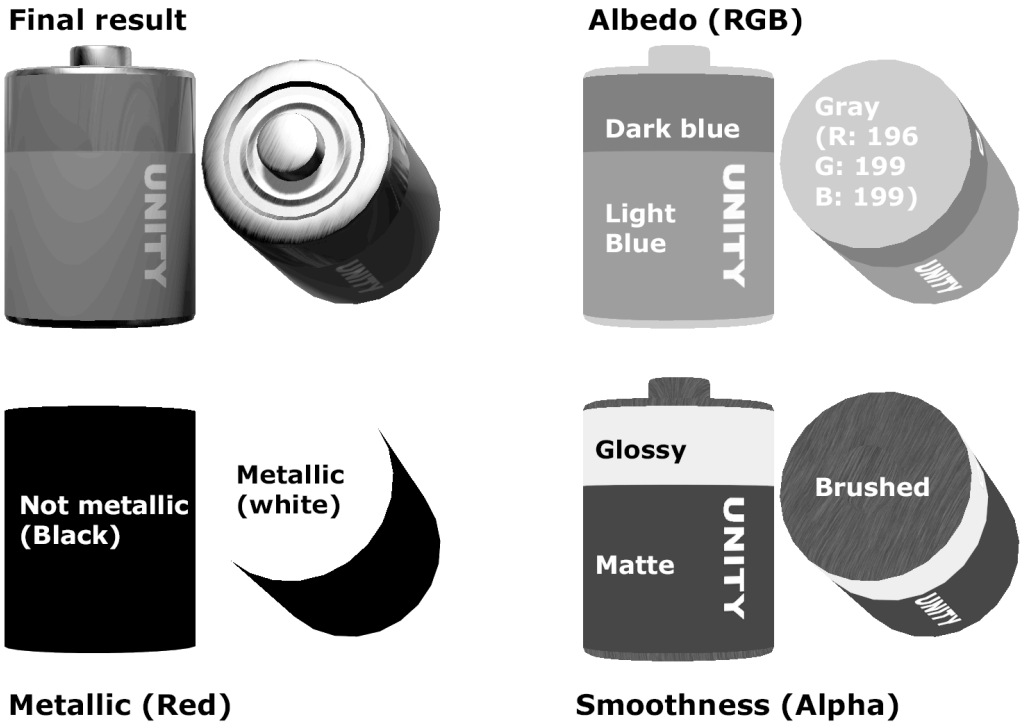
Metallic workflow

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

Albedo: As in the Specular workflow, this is the material's diffuse color, how you would describe the material, etc. 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, etc.) 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: How metallic the material looks. Metallic texture maps make use of the Red channel for Metallic value (black for non-metallic, white for metallic materials that are not painted or rusted) and Alpha Channel for smoothness (in a similar way to the Specular workflow) . Please note that Metallic maps do not include any information on hue, in which case the yellow-ish nature of metallic gloss should be applied to the Albedo map.

To reproduce the battery that illustrated the Specular workflow using the Metallic workflow, maps would have to be re-created as follows:



Insert image 1362OT_04_40.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 map: 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 map: A greyscale map used to simulate dark sections of an object under ambient light. Normally used to emphasize joints, creases and other details in geometry.

Height map: Adds a displacement effect, giving the impression of depth without the need for complex geometry.

Emission map: 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 a good idea to read Unity's documentation on textures. It can be found online at `unity3d.com/support/documentation/Manual/Textures.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 **Unity Asset Store**, is a fantastic collection, featuring sample materials (both Metallic and Specular setup) for wood, metal, rubber, plastic, glass, skin, mud, and much more.

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

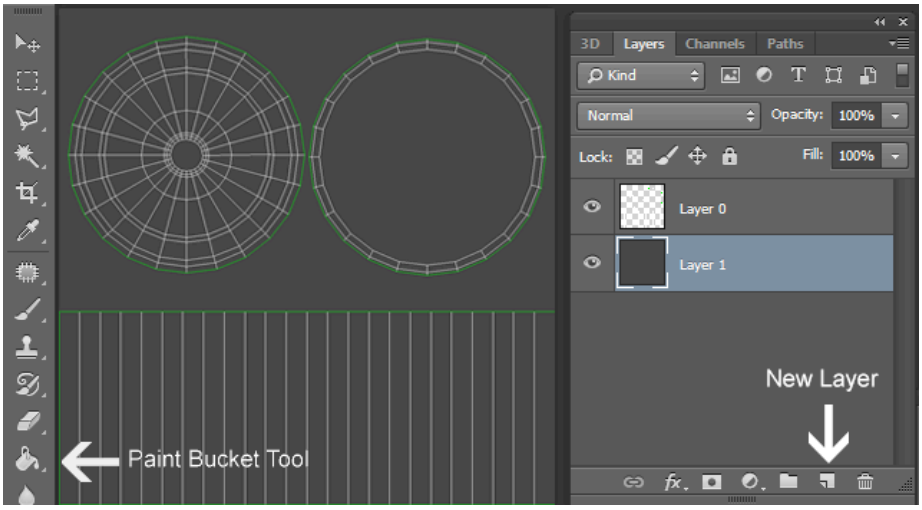
Two files have been previously prepared to support this recipe: a 3D model (in FBX format) of a battery, and a UVW template texture (in PNG format) to guide us when creating the diffuse texture map. 3D models and UVW templates can be made with 3D modeling software such as 3DS MAX, Maya or Blender. All necessary files are available inside the `1362_04_01` folder.

How to do it...

To create a basic material, follow these steps:

1. Import the files `battery.FBX` and `uvw_template.png` into 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 `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 should be visible on the top of the solid black fill.

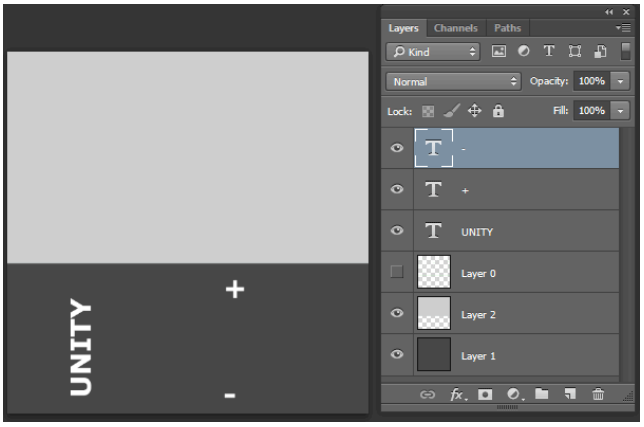


Insert image 1362OT_04_02.png

5. 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).**

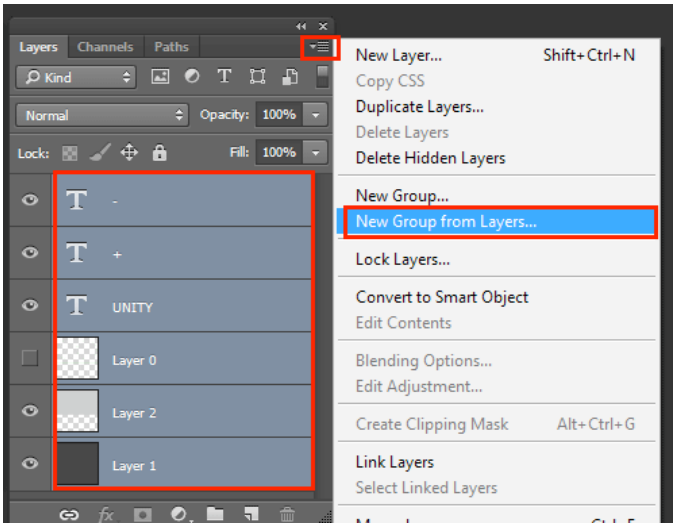
> RGB values for our specular map are not arbitrary: Physically Based Shading takes out most of the guessing from the mapping process, replacing it with research for references. In our case, we have used colors based on the reflectance values of iron (the slightly hued light gray) and plastic (dark gray).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

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



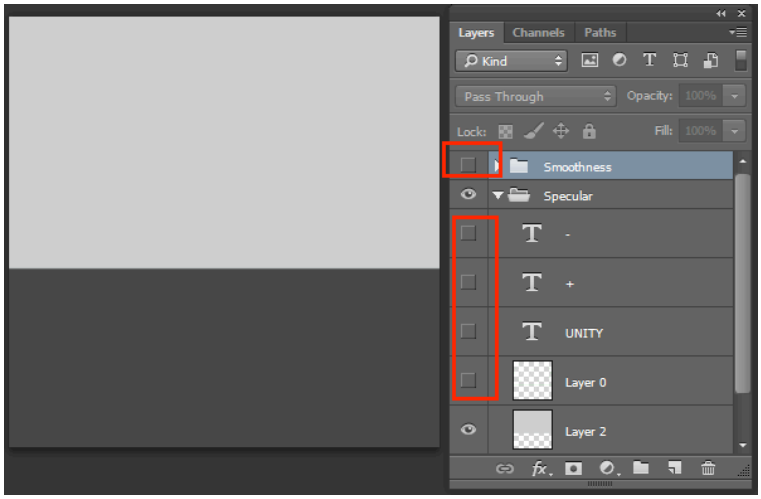
Insert image 1362OT_04_41.png

7. Select all you layers and organize them into a Group (in Photoshop this can be done by clicking the dropdown menu on the **Layers** window and navigating to **Window | New Group from Layers…**). Name the new group `Specular`.



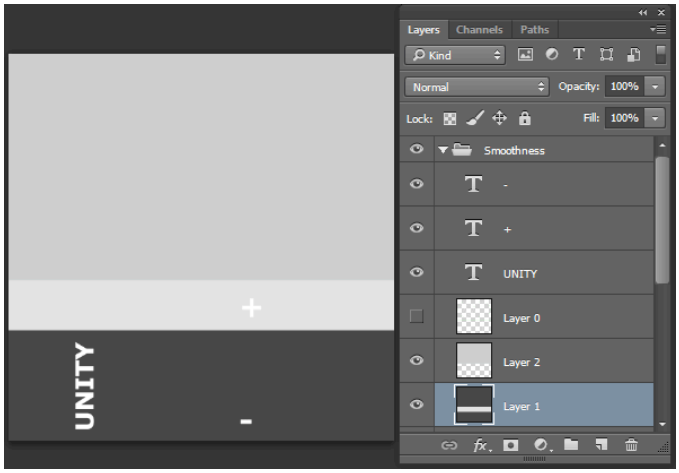
Insert image 1362OT_04_42.png

8. Duplicate the `Specular` group (on the **Layers** window, right-click the group's name and select **Duplicate Group…**). Name the duplicated group as `Smoothness`.
9. Hide the `Smoothness` group. Then, expand the `Specular` group and hide all text layers.



Insert image 1362OT_04_43.png

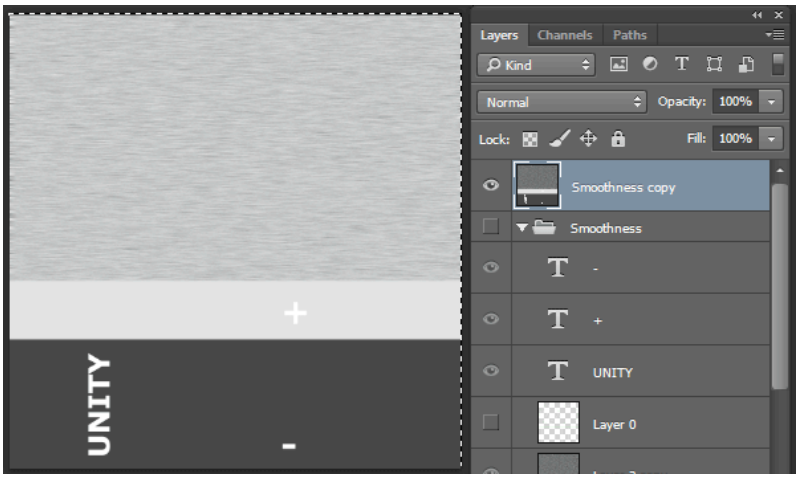
10. Unhide the `Smoothness` group and hide the `Specular`. Select the dark grey layer. Then, make an area selection around the upper region of the battery body and fill it with light grey (**R: 220**, **G: 220**, **B: 220**). Rescale and rearrange the Text layers, if you have to.



Insert image 1362OT_04_44.png

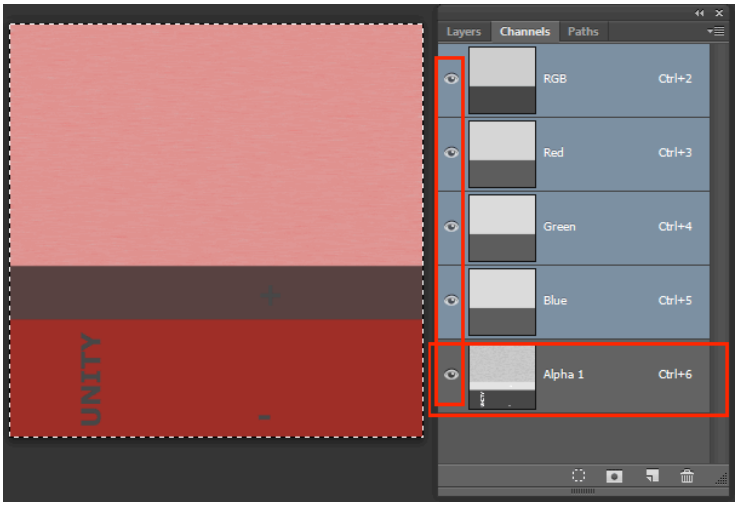
11. Duplicate the layer that contains the grey fill for the upper section of the image (the one that went over the circles).

12. To add a brushed quality to this specular map, a **Noise** filter to the duplicated layer (in Photoshop this can be done by accessed via the menu **Filter | Noise | Add Noise...**). Use **50%** as the **Amount** and set **Monochromatic** as true. Then, apply a **Motion Blur** filter (**Filter | Blur | Motion Blur...)** using **30 Pixels** as **Distance**.
13. Duplicate the `Smoothness` group. Then, select the duplicated group and merge it into a single layer (on the **Layers** window, right-click the group's name and select **Merge Group**).
14. Select the merged layer, use the *CTRL* + *A* keys to select the entire image, and copy it using the *CTRL* + *C* keys.



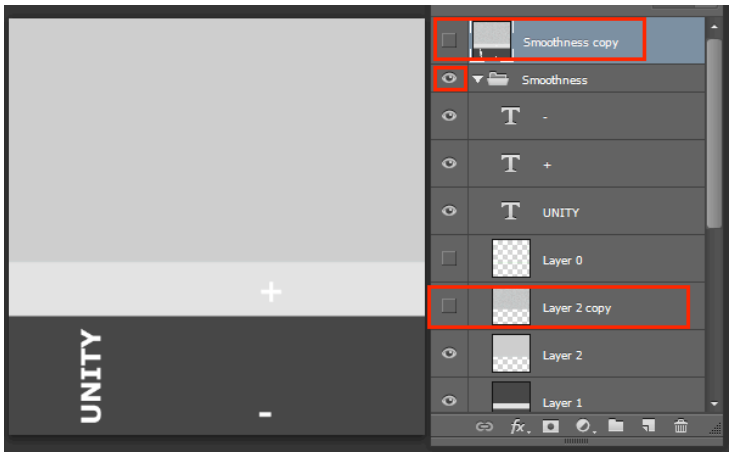
Insert image 1362OT_04_45.png

15. Hide the merged layer and the `Smoothness` group. Then, unhide the `Specular` group.
16. In your image editor, access the image Channels window (in Photoshop this can be done by navigating to **Window | Channels**). Create a New Channel. That will be our **Alpha** channel.
17. Paste the image that you have previously copied (from the merged layer) into the **Alpha** channel. Then set all channels as visible.



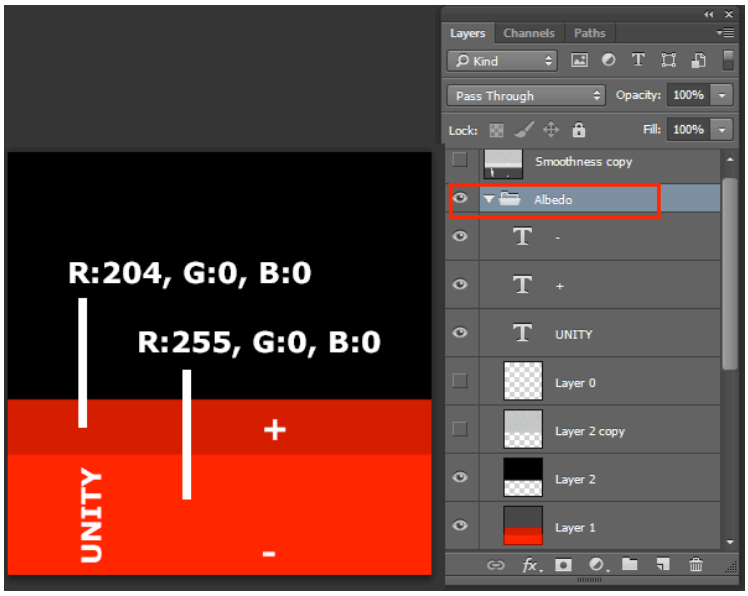
Insert image 1362OT_04_46.png

18. Save your image into the Project's Assets folder as `Battery_specular` either in Photoshop format (.PSD) or .TIF.
19. 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.
20. 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 1362OT_04_47.png

21. Change the color of the upper rectangle to black. Then, change the light grey area to dark red (**R: 204**, **G: 0**, **B: 0**) and the dark grey to red (**R: 255**, **G: 0**, **B: 0**). Rename the Group as `Albedo` and save the file.



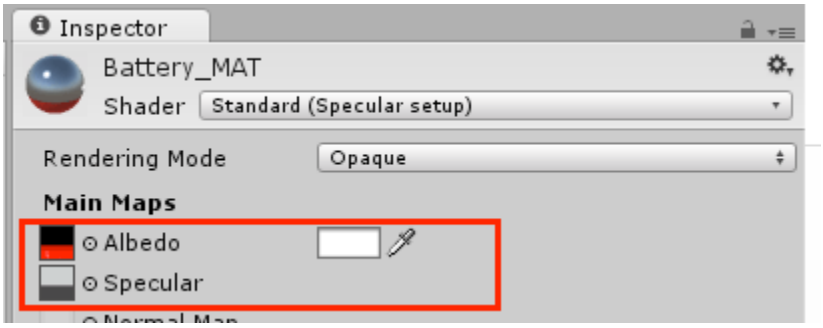
Insert image 1362OT_04_48.png

22. Go back to Unity and make sure 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 the **Create** drop-down menu and choose **Material**.

---

23. Select `Battery_MAT`. From the **Inspector** view, change the Shader to **Standard (Specular setup),** and make sure the rendering mode is set as **Opaque**.
24. Set `Battery_specular` as the **Specular** map, and `Battery_albedo` as the **Albedo** map, for `Battery_MAT`.
25. Drag the `Battery_MAT` material from the **Project** view and drop it onto the **battery** object, in the **Hierarchy** view.



Insert image 1362OT_04_49.png

26. Drag the `Battery_MAT` material from the **Project** view and, in the **Hierarchy** view, drop it into the **battery** object.

Insert image 1362OT_04_50.png

How it works...

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

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

- **Specular** map (RGB): Very dark grey specular (for non-metallic appearance);
- **Smoothness** (Alpha channel of Specular map): Light gray (for glossy aspect);
- **Albedo** map: Dark red (for dark red color).

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

- **Specular** map (RGB): that same dark grey specular;
- **Smoothness** (Alpha channel of Specular map): dark gray (for matte aspect);
- **Albedo** map: red (for red color).

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

- **Specular** map (RGB): light grey (for a metallic aspect)
- **Smoothness** (Alpha channel of Specular map): blurred grey noise pattern (for brushed aspect);
- **Albedo** map: black (for red color).

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

There's more...

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

Setting the Texture type for an image file

Since image files can be used for several purposes within Unity (texture maps, GUI textures, cursors, etc.), it's a good idea to check if the right **Texture Type** is assigned to your file. That can be done by selecting the image file in the **Project** view and, in the **Inspector** view, using the dropdown menu to select the right **Texture Type** (in this case, Texture). Please note 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, but still be able to edit them in full size.

Combining the map with color

When editing a material, the Color picker on the right to 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 a Specular/Smoothness maps 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.

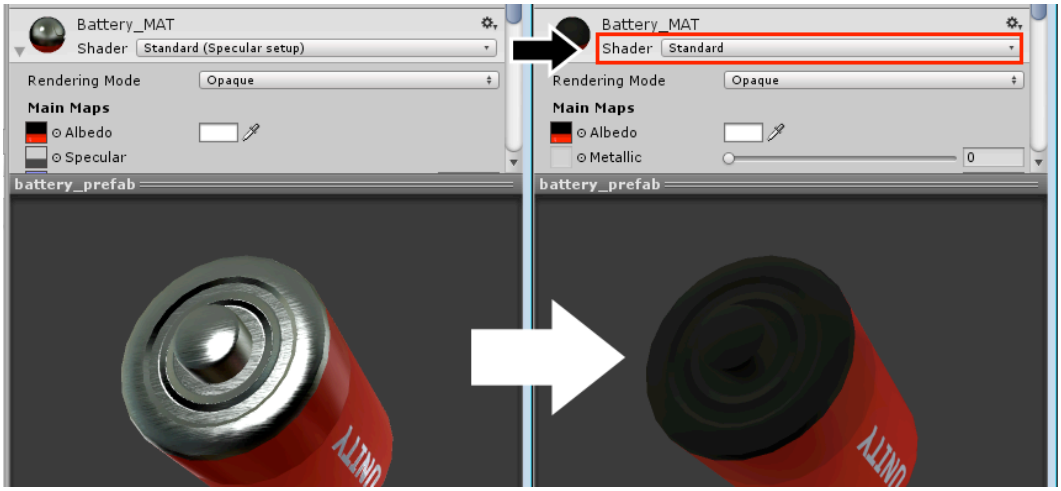
Getting ready

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 inside the `1362_04_02` folder.

How to do it...

To create a basic material, follow these steps:

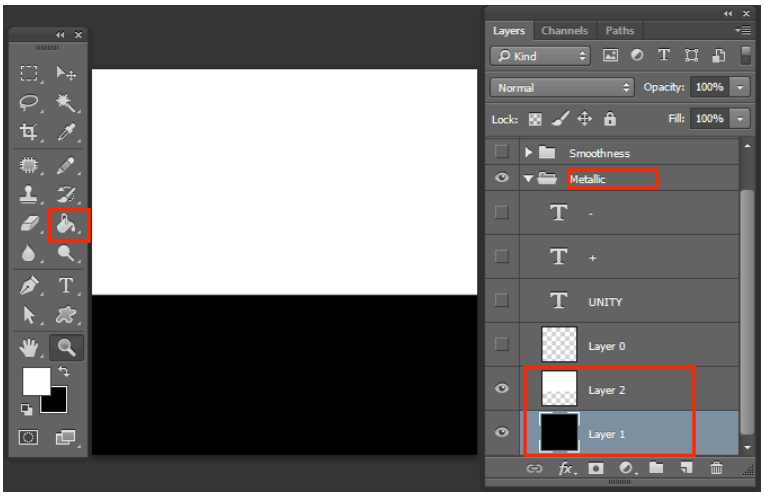
1. Import the `battery_prefab` Unity package into a new project.
2. From the **Project** view, select the **battery_prefab**. Then, from the **Inspector**, access its material (named **Battery_MAT**) and change its **Shader** to **Standard** (as opposed to its current shader, **Standard (Specular setup)**.



Insert image 1362OT_04_51.png

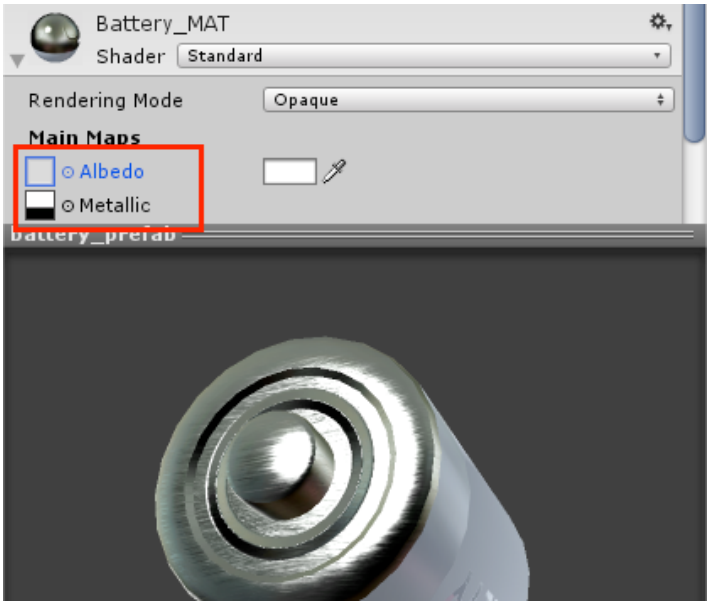
3. From the **Project** view, find the `Battery_specular` map and rename it `Battery_metallic`. Open it in your image editor (we'll use Adobe Photoshop to illustrate the next steps).
4. Find the layer Group named **Specular** and rename it as **Metallic**. Then, fill the light gray layer (named **Layer 2**, inside the **Metallic** group) with white (**R: 255**,

G: 255, **B: 255)** and the dark gray layer (named **Layer 1**, inside the **Metallic** group) with black (**R: 0**, **G: 0**, **B: 0)**. Save the file.



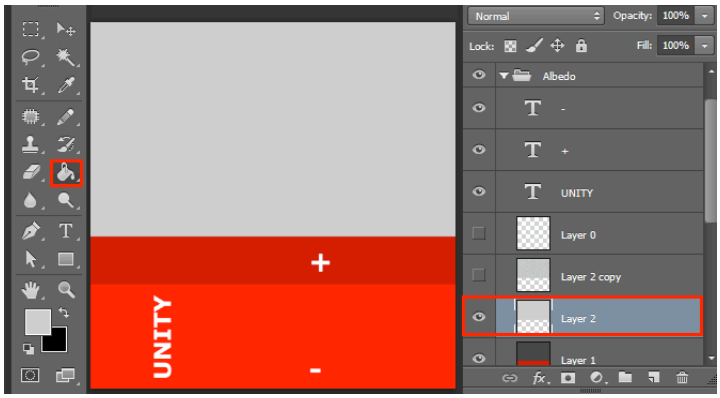
Insert image 1362OT_04_52.png

5. 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. That should give you an idea on how the material is coming along.



Insert image 1362OT_04_53.png

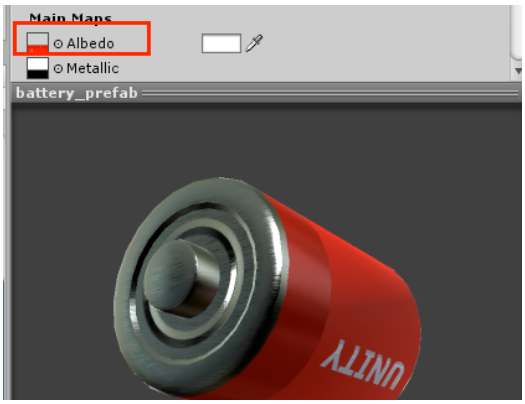
6. Now, let's adjust the **Albedo** texture map. From the **Project** view, locate the `Battery_albedo` map and open it in your image editor. Then, use the **Paint Bucket** tool to fill the black area of **Layer 2**, inside the **Albedo** group, with light gray (**R: 196**, **G: 199**, **B: 199)**. Save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



Insert image 1362OT_04_54.png

7. 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 you have edited and assigned.



Insert image 1362OT_04_55.png

How it works...

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

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

- **Metallic** map (RGB): Black (for non-metallic appearance);
- **Smoothness** (Alpha channel of Specular map): Light gray (for glossy aspect);
- **Albedo** map: Dark red (for dark red color).

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

- **Metallic** map (RGB): Black;
- **Smoothness** (Alpha channel of Specular map): dark gray (for matte aspect);
- **Albedo** map: red (for red color).

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

- **Metallic** map (RGB): white (for a metallic aspect)
- **Smoothness** (Alpha channel of Specular map): blurred grey noise pattern (for brushed aspect);
- **Albedo** map: light grey (for iron-like aspect).

Remember to organize your layers in groups named after the property 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 3D polygons during game runtime. Oversimplifying, Normal maps fake complex geometry onto low-definition 3D meshes. Those maps can be generated either from projecting high-definition 3D meshes onto low-poly ones (a technique usually referred to as *bake*), or, as it will be the case for this recipe, generated from another texture map.

Getting ready

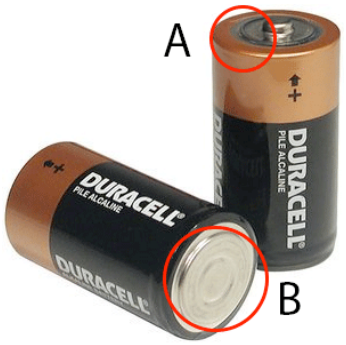
For this recipe, we will prepare two texture maps: the **Heightmap** and the **Normal map**. The former will be made from simple shapes, in an image editor. The latter, automatically processed from the heightmap. Although there is a number of tools that can be used to generate normal maps (see *There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

To help us with this recipe, it's been provided a Unity package containing a prefab made of a 3D object and its material, and also a UVW template texture (in PNG format) to guide us when creating the diffuse texture map. All files are inside the `1362_04_03` folder.

How to do it...

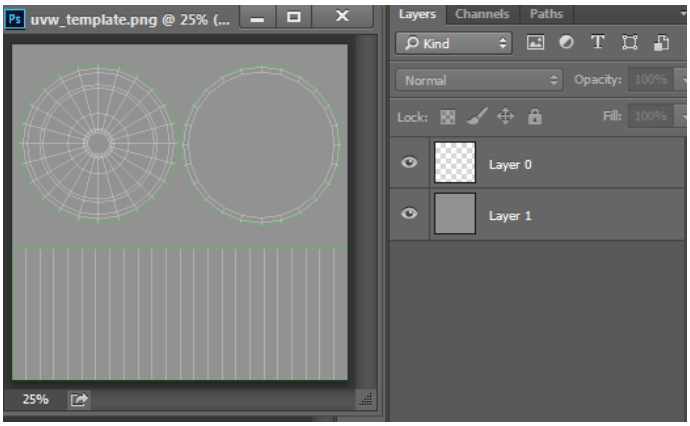
To apply a Normal map to a material, follow these steps:

1. Import `1362_04_03.unitypackage` into your project. Select the `batteryPrefab` from the **Assets | 1362_04_03** folder, in the **Project** view. Comparing it to some reference photos, inform yourself about the features that should be reproduced by the Normal map: (**A**) a bumpy ring on top; (**B**) circular creases on the bottom.



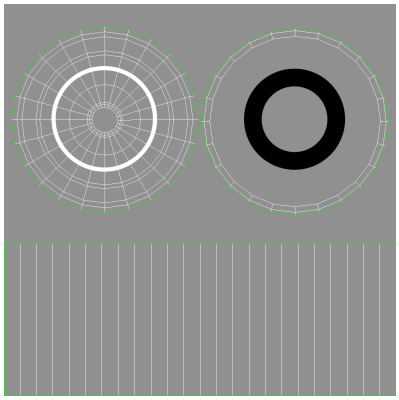
Insert image 1362OT_04_14.png

2. In an image editor, open `uvw_template.png`. Create a new layer, fill it with grey (RGB: 128), and position it below the preexisting layer.



Insert image 1362OT_04_15.png

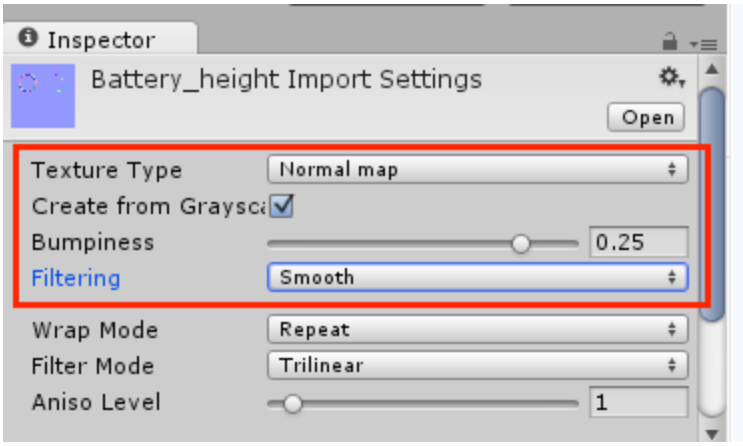
3. On a separate layer, draw a white circle centralized to the battery's top. Then, on another layer, draw a black circle, centralized to the battery's bottom.



Insert image 1362OT_04_16.png

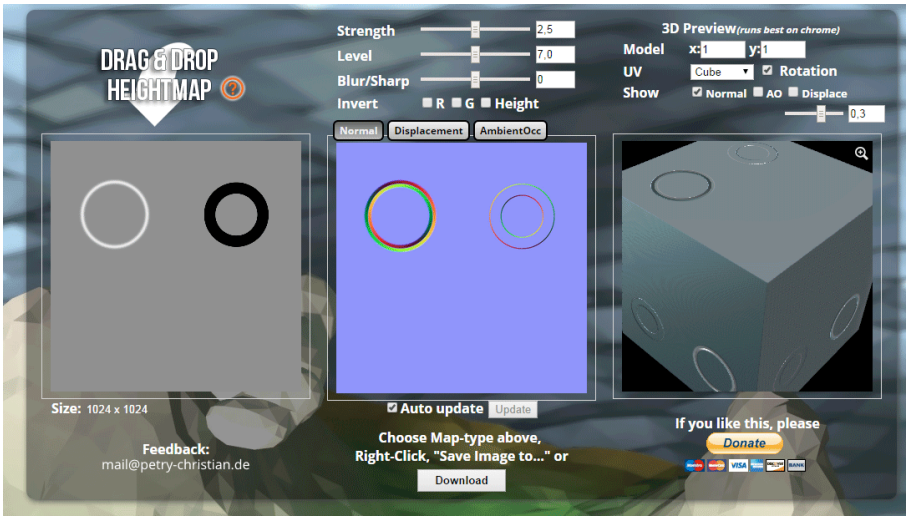
4. In case you have used vector shapes to make the circles, rasterize their layers (in Photoshop, right-click the layer's name and select the option **Rasterize Layer** from the context menu).
5. Blur the white circle (in Photoshop this can be done by accessed via the menu **Filter | Blur | Gaussian Blur...**). Use **4,0** pixel as **Radius**.

6. Hide the uvw template layer and save the image as `Battery_height.png`.
7. In case you want to convert the heightmap directly from Unity, import it into your project, select it from the **Project** view and, from the **Inspector** view, change its **Texture Type** to **Normal map**, check the option **Create from Grayscale**, adjust **Bumpiness** and **Filtering** as you like and click **Apply** to save changes.



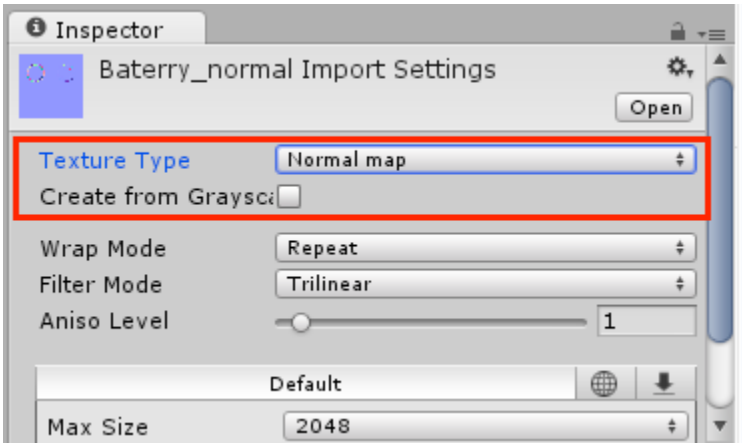
Insert image 1362OT_04_17.png

8. To convert your heightmap externally, access the website `http://cpetry.github.io/NormalMap-Online/`. Then, drag the file `HEIGHT_battery.png` into the appropriate image slot. Feel free to play with the **Strength**, **Level** and **Blur/Sharp** parameters.



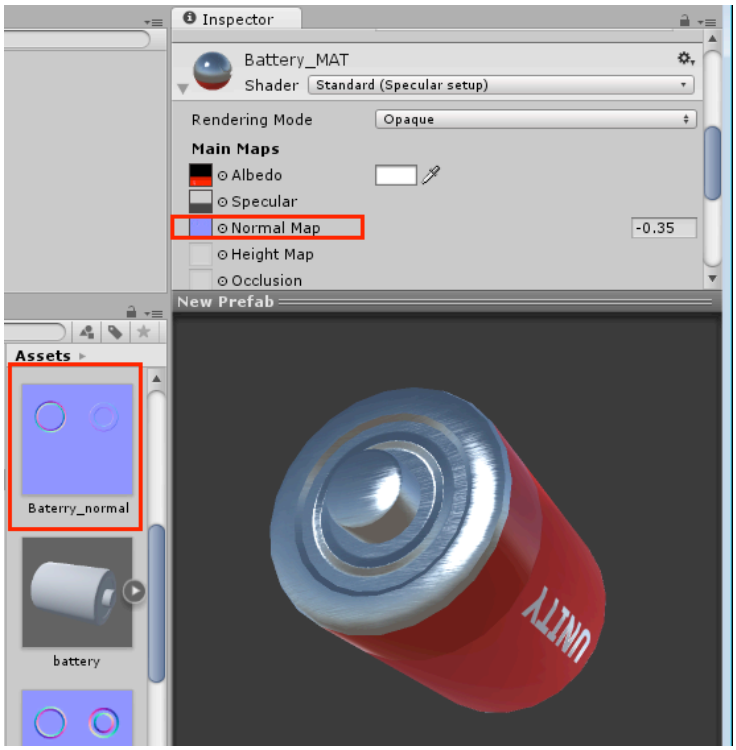
Insert image 1362OT_04_18.png

9. Save the resulting Normal map as `Battery_normal.jpg` and add it to your Unity project.
10. In Unity, select `Battery_normal` from the **Project** view. Then, from the **Inspector** view, change its **Texture Type** to **Normal**, leaving the box **Create from Grayscale** unchecked. Click **Apply** to save changes.



Insert image 1362OT_04_19.png

11. 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`.



Insert image 1362OT_04_20.png

How it works...

The **Normal** map was calculated from the grey values on the **Heightmap**, where lighter tones were interpreted as recesses (applied on the top of the battery) and darker tones, bulges (applied on 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 redrawing the Heightmap, switching 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 you might want to check out:

CrazyBump, standalone tool for Windows and Mac, available at `www.crazybump.com`.

nDo, Photoshop plugin by Quixel (Windows only), available at `quixel.se/ndo`.

GIMP normalmap Plugin, Windows only, available at `code.google.com/p/gimp-normalmap/`.

NVIDIA Texture Tools for Adobe Photoshop, Windows only, available at `developer.nvidia.com/nvidia-texture-tools-adobe-photoshop`.

Bitmap2Material, amazing texture generating tool from Allegorithmic, 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 LED 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, we 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

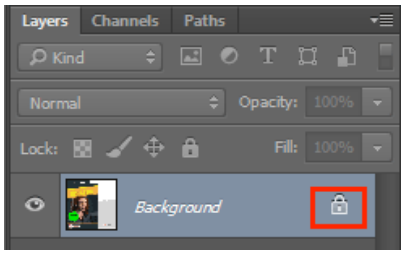
For this recipe, we have prepared a Unity package containing a prefab made of a 3D object, its material, and its respective diffuse texture map (in .PNG format). All files are inside the `1362_04_04` folder.

How to do it...

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

1. Import `1362_04TransparencyEmission.unitypackage` into your project. Select the `DIFF_package` texture from the **Assets 1362_04** folder, in the **Project** view. Then, open it in your image editor.
2. First, we will add transparency to the image by deleting the white areas around the package (and the hang hole). Make a selection for those areas (in Photoshop, than can be done with the **Magic Wand Tool**).

3. Make sure to unlock the **Background** layer by clicking on the lock icon on the left to the layer's name.

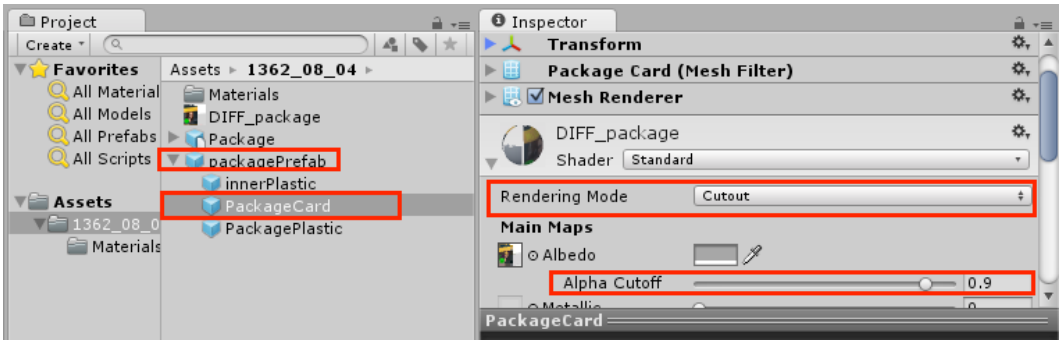


Insert image 1362OT_04_21.png

4. Delete the previously made selection (that can be done in Photoshop by pressing the *Delete* key). The background of the image should be transparent. Save the file.

Insert image 1362OT_04_22.png

5. Back in Unity, in the **Assets** folder, expand the **packagePrefab** 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**.

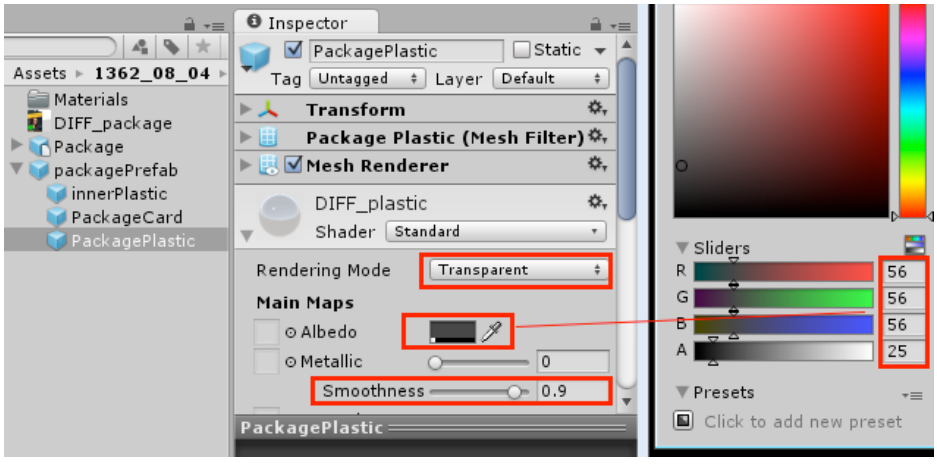


Insert image 1362OT_04_23.png

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.

---

6. 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** down to `25`. Also, change the **Smoothness** level to `0.9`.

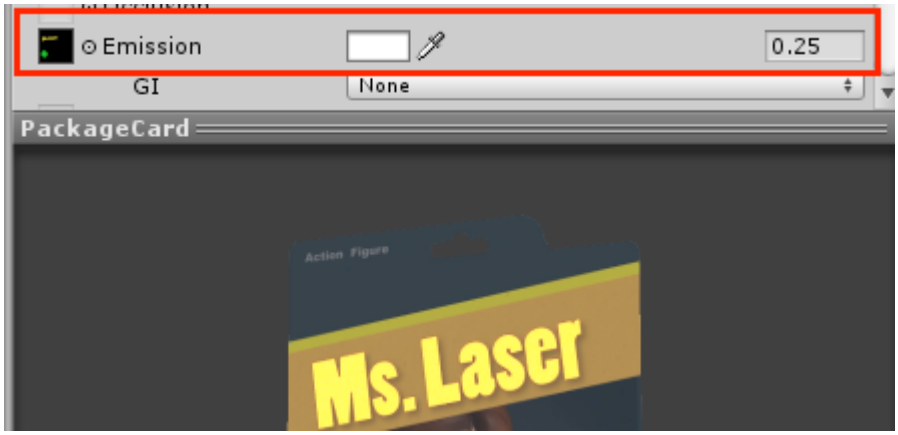


Insert image 1362OT_03_24.png

7. Now that we have taken care of our transparency needs, we need to work out on the **Emission** map. From the **Assets** folder, duplicate the `DIFF_package` texture, rename it `EMI_package` and open it in your image editor.
8. Select all characters from the inscription **"Ms. Laser"** and the green star (in Photoshop, than can be done with the **Magic Wand Tool**, keeping the *Shift* key pressed while selecting multiple areas).
9. 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 accessed via the menu **Filter | Noise | Add Noise...**). Use **50%**.
10. 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.
11. Flatten your image (in Photoshop this can be done by accessed via the menu **Layer | Flatten Image)** and save your file.

Insert image 1362OT_03_25.png

12. Back in Unity, in the **Assets** folder, expand the **packagePrefab** and select the **PackageCard** object. Now, in the **Inspector** view, scroll down to the **Material** component and assign `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`. Also, change its **GI** option to **None**, so its glow won't be added to lightmaps nor influence illumination in realtime.



Insert image 1362OT_03_26.png

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

Insert image 1362OT_03_27.png

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 we didn't add an Alpha channel – that's 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 as the .PNG we have generated.

Insert image 1362OT_03_28.png

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

There's more...

A few more information on Transparency and Emission:

Using texture maps with Transparent Mode

Please note that you could use a bitmap texture for the **Diffuse** map in **Transparent** Render Mode. In this case, RGB values would be interpreted as the Diffuse color, while the Alpha would be used to determine that pixel`s transparency (in this case, semi-transparent materials are allowed).

Avoiding issues with semi-transparent objects

You might have noticed that the plastic case was made from two objects (**PackagePlastic** and **innerPlastic**). That 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 those faces to be correctly sorted for rendering. Materials in **Cutout** mode are not affected by this problem, though.

Emitting light over other objects

The **Emission** value is also 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 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 applied to create 3D user interfaces.

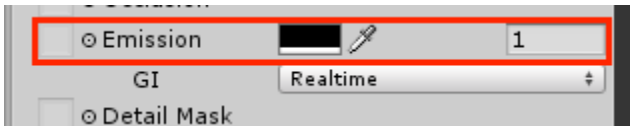
Getting ready

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

How to do it...

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

1. Create a new 3D project, and add a **Cube** to the scene (from the **Hierarchy** view, choose **Create | 3D Object | Cube**).
2. From the **Project** view, click the **Create** dropdown menu and choose **Material**. Name it `HighlightMaterial`.
3. Select **HighlightMaterial** and, from the **Inspector** view, change its **Albedo** color to gray (**R:** `135`, **G:** `135`, **B:** `135`), its **Emission** intensity to `1` and **Emission** color to **R:** `1`, **G:** `1`, **B:** `1`.



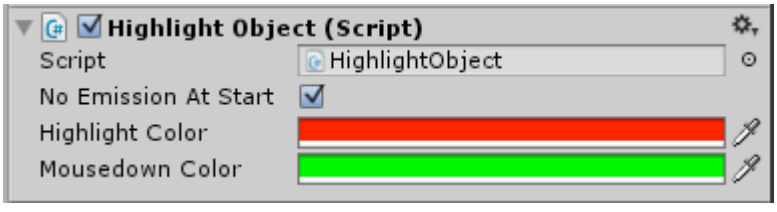
Insert image 1362OT_03_56.png

4. Assign the **HighlightMaterial** to the **Cube** you have previously created.
5. From the **Project** view, click the **Create** dropdown menu and choose **C# Script**. Rename it `HighlightObject` and open it in your editor.
6. 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("_Emission
Color");
    }

    void OnMouseEnter(){
        mouseon = true;
        myRenderer.material.SetColor("_EmissionColor", highlightCo
lor);
    }

    void OnMouseExit(){
        mouseon = false;
        myRenderer.material.SetColor("_EmissionColor",initialColor
);
    }

    void OnMouseDown(){
        myRenderer.material.SetColor("_EmissionColor", mousedownCo
lor);
    }

    void OnMouseUp(){
        if (mouseon)
            myRenderer.material.SetColor("_EmissionColor", highlig
htColor);
        else
            myRenderer.material.SetColor("_EmissionColor", initial
Color);
    }
}
```

7. Save your script and attach it to the **Cube**.
8. Select the cube and, in the **Inspector** view, set the **Highlight Color** and **Mousedown Color** to any colors you would like.



Insert image 1362OT_04_29.png

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

10. Test the scene. The **Cube** should 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 is adding a behavior to the cube when these events are detected. The `Start()` method gets a referent to the Renderer component of the GameObject the script has been added to, and stores it in variable `myRenderer` (note – 'renderer' already has a meaning in Unity so it is not appropriate as a private variable name for this script). The boolean variable `mouseon` records if the mouse pointer is currently over the object. When the mouse button is released we use variable `mouseon` to decide whether to change the cube's back to its initial color (`mouseon` FALSE so mouse pointer away from cube) or back to its highlight color ((`mouseon` TRUE so mouse pointer over cube).

The reason we needed to change the material`s original **Emission** color to ultra-dark gray is because leaving it black would cause Unity to optimize the Shader by removing the **Emission** property from the material – our script wouldn't have worked, then.

There's more...

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

Adding Detail maps to a material

When creating large objects, there is often the desire to texture it as a whole, but also add details that make it look good at closer distances. To overcome the need for gigantic texture maps, the use of Detail can make a real difference. In this recipe, we will add Detail to a rocket toy by applying a Detail mask and a Detail Normal map. In our case, we want to add a textured quality (also adding a stripe pattern) to the green plastic, except in the region where the battery compartment and the toy's logo are.

Insert image 1362OT_04_30.png

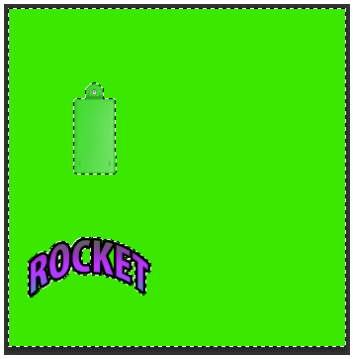
Getting ready

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 inside the `1362_04_06` folder.

How to do it...

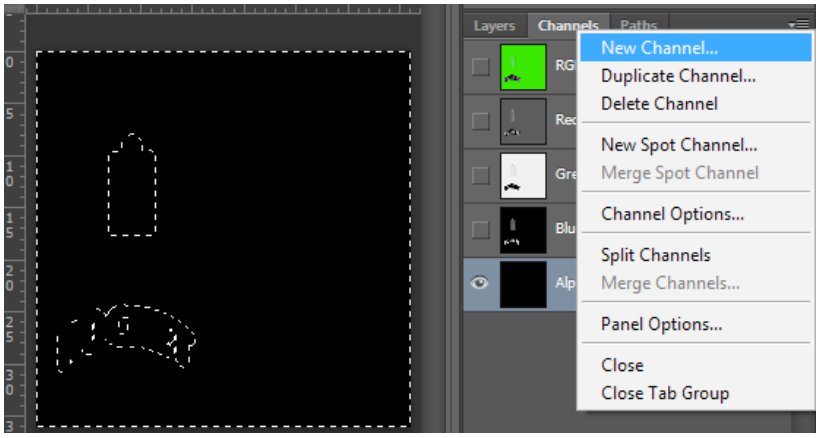
To add Detail maps to your object, follow these steps:

1. 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.
2. From the **Hierarchy** view, expand the **rocketToy** game object and select its child **rocketLevel1**. Then, scroll down the **Inspector** view down to the **Material** component. Observe that it uses the texture `DIFF_ship` as **Diffuse** map. Duplicate this file and rename the new copy `COPY_ship`.
3. Open the `COPY_ship` in your image editor. Select all the solid green pixels around the logo and battery compartment (in Photoshop, than can be done with the **Magic Wand Tool**, keeping the *Shift* key pressed while selecting multiple areas).



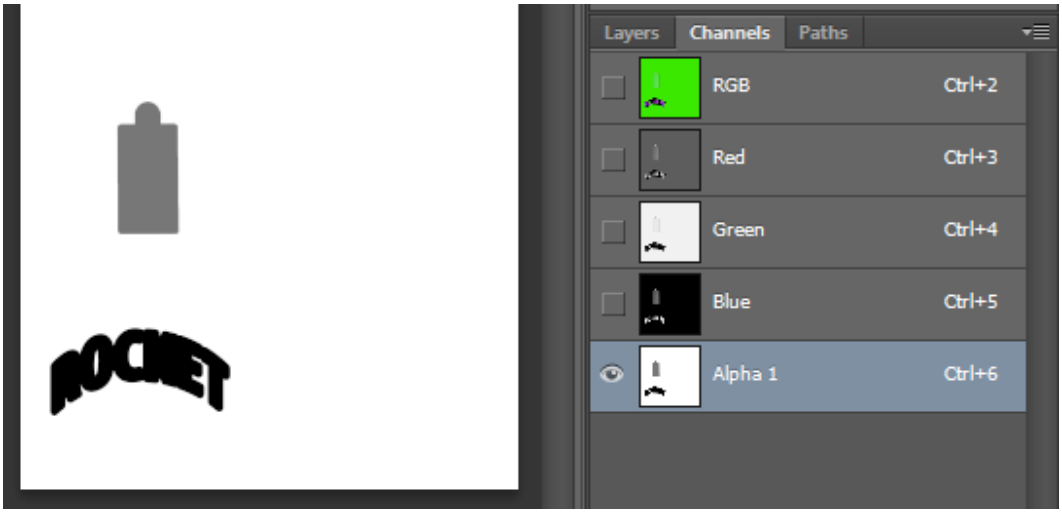
Insert image 1362OT_04_31.png

4. Keeping your selection active, access the image Channels window (in Photoshop this can be done by navigating to **Window | Channels**). Create a New Channel. That will be our **Alpha** channel.



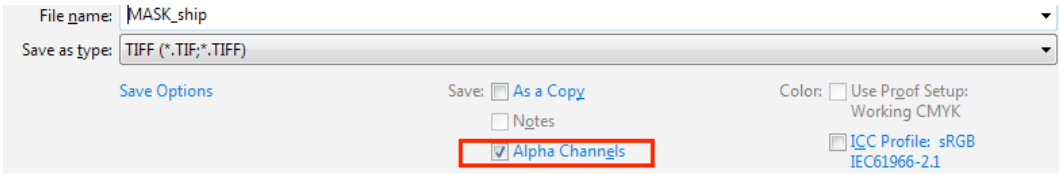
Insert image 1362OT_04_32.png

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



Insert image 1362OT_04_33.png

6. Save it in the TIFF format as `MASK_ship.TIF`, inside the `Assets` folder. Make sure to include the **Alpha Channels**.



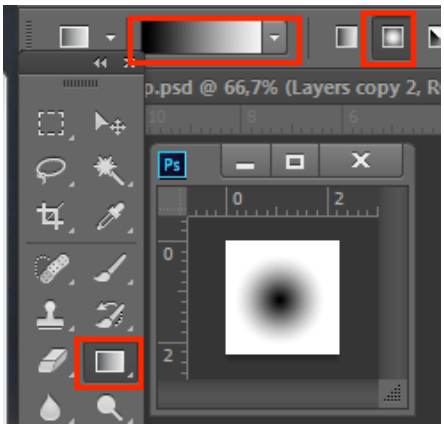
Insert image 1362OT_04_34.png

7. Now that we have the mask, let's create a diffuse map for our Detail. In your image editor, create a new image with dimensions **width: 64**, **height: 64**.
8. 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 about 16 pixels tall.

Insert image 1362OT_04_35.png

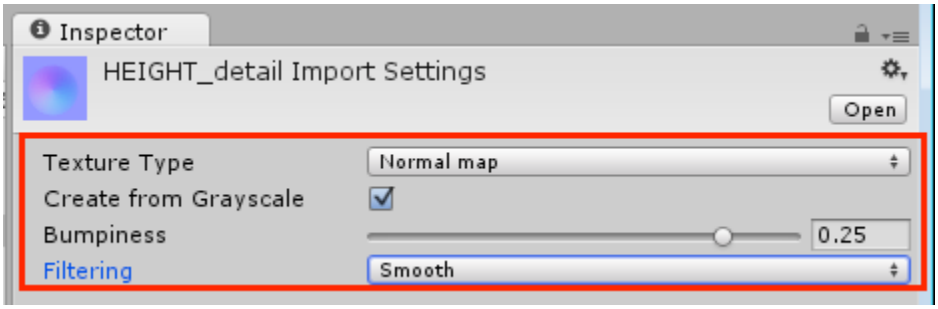
9. Save the image as `DIFF_detail.PNG`, inside the `Assets` folder.

10. Create a new 64 x 64 image. Use a **Gradient** tool to create a Black and White **Radial Gradient** (in Photoshop, than can be done with the **Gradient Tool** in **Radial** mode).



Insert image 1362OT_04_36.png

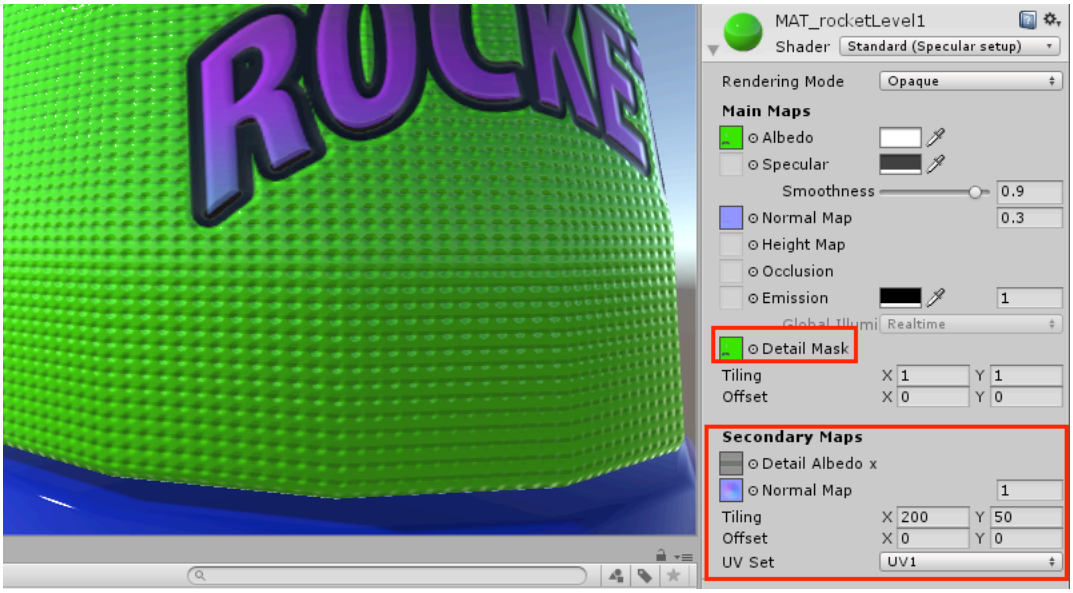
11. Save the image as `HEIGHT_detail.PNG`, inside the `Assets` folder.
12. Go back to Unity. From the Assets folder, select `HEIGHT_detail`. Then, from the Inspector view, change its **Texture Type** to **Normal map**, check the option **Create from Grayscale**, adjust **Bumpiness** to 0.25 and **Filtering** to smooth. Click **Apply** to save changes.



Insert image 1362OT_04_37.png

13. From the Hierarchy view, expand the **rocketToy** game object and select its child **rocketLevel1**. Then, scroll down the **Inspector** view down 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**.

14. In the **Secondary Maps** section, change **Tiling** values to **X: 200**, **Y:50**. You might notice that the pattern is not seamless. That is because we are using the same **UV Set** from our **Diffuse** texture. However, the object has been assigned 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, in the **Secondary Maps** section, its **UV Set** from **UV0** to **UV1.** The Detail for your material is ready.



Insert image 1362OT_04_38.png

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 to the original **Diffuse** texture). By using a **Detail Mask**, the artist defines which areas of the object should be affected by Secondary Maps. That is great for customization, and also for creating nuances (like the semi-bumped battery compartment, in our example).

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

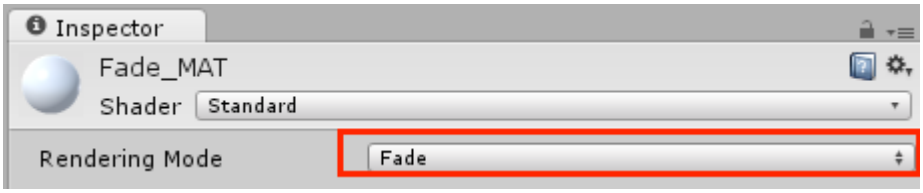
Fading the transparency of a material

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:

1. Add a **Sphere** to your scene by accessing the **GameObject | 3D Object | Sphere** menu.
2. 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).
3. 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**.
4. Rename your new material. For this example, let's call it `Fade_MAT`.
5. Select your material. From the **Inspector** view, use the drop-down menu to change its **Rendering Mode** to **Fade**.



Insert image 1362OT_04_57.png

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.

---

6. Apply the **FadeMaterial** to **Sphere** by dragging it from the **Project** view into the **Sphere** Game Object name in the **Hierarchy** view.
7. From the **Project** view, click the **Create** drop down menu and choose **C# Script**. Rename it as `FadeMaterial` and open it in your editor.
8. 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;
    }
}
```

9. Save your script and apply it to the **Sphere** Game.
10. 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

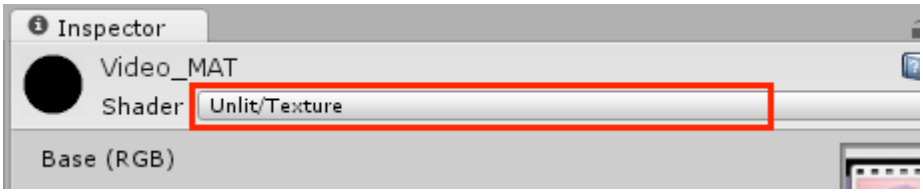
Unity imports video files through Apple Quicktime. If you don't have it installed in your machine, please download it at `http://www.apple.com/quicktime/download/`.

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...

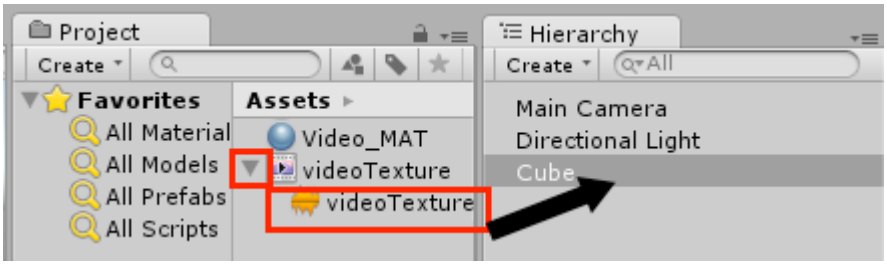
Follow these steps:

1. Add a cube to the scene through the **GameObject | 3D Object | Cube** menu.
2. Import the provided `videoTexture.mov` file.
3. 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**.



Insert image 1362OT_04_58.png

4. Apply `videoTexture` to the texture slot of `Video_MAT` by dragging it from the **Project** view into the appropriate slot.
5. Apply the `Video_MAT` to the **Cube** you have previously created.
6. 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).



Insert image 1362OT_04_59.png

7. 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.
8. 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`.
9. 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();
        }
    }
}
```

10. Save your script and attach it to the **Cube**.
11. 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** window, 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.

Conclusion

This chapter has covered a number of techniques used to create, often manually, sometimes automatically, texture maps capable of giving distinctive features to materials. Hopefully, you are now more confident to work under Unity's new Physically Based Shading, capable of understanding differences between available workflows, aware of the role of each material property, and ready to make better looking materials for your games. We have also explored ways of changing material properties 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, getting in touch with the tools and concepts behind it. To help you on this task, here is a non-exhaustive list of resources you should take a look at:

References

A list of interesting, detailed material on Physically Based Rendering (within and without Unity).

The Comprehensive PBR Guide Volumes 1 and 2, Wes McDermott (Allegorithmic), available at www.allegorithmic.com/pbr-guide. This guide takes an in-depth look at 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), available at www.slideshare.net/RenaldasZioma/unite2014-mastering-physically-based-shading-in-unity-5. A detailed presentation about using PBS in Unity. Originally presented at the Unite 2014 conference, it contains some out-of-date information, but is still worth taking a look at.

Physically Based Shading in Unity 5, by Aras Pranckevičius, from Unity. Available at aras-p.info/texts/talks.html. Slides and notes from a presentation on the subject given at GDC.

Tutorial: Physically Based Rendering, And You Can Too!, by Joe "EarthQuake" Wilson. Available at www.marmoset.co/toolbag/learn/pbr-practice. A great overview from the makers of **Marmoset Toolbag** and **Skyshop**.

Polycount PBR Wiki, Available at wiki.polycount.com/wiki/PBR. A list of resources compiled by the **Polycount** community.

Tools

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

Substance Painter, 3D painting application from **Allegorithmic**. Available at www.allegorithmic.com /products/substance-painter. Again, it's worth mentioning that **Allegorithmic** products won't make use of Unity's Standard shader, relying on **substance** files, natively supported by Unity.

Bitmap2Material, creates full-featured materials (including normal map, specular map, etc.) from a single bitmap image. Also from **Allegorithmic**. Available at www.allegorithmic.com /products/bitmap2material.

Quixel DDO, A plugin for creating PBR-ready textures inside Adobe Photoshop. From **Quixel**, it is available at quixel.se/ddo.

Quixel NDO, A plugin for creating normal maps inside Adobe Photoshop. From **Quixel**, it is available at quixel.se/ndo.

Mari, 3D painting tool from **The Foundry**. Available at www.thefoundry.co.uk/products/mari/