HOUR 3

Models, Materials, and Textures

What You'll Learn in This Hour:

- The fundamentals of models
- How to import custom and premade models
- How to work with materials and shaders

In this hour, you'll learn all about models and how they are used in Unity. You'll start by looking at the fundamental principles of meshes and 3D objects. From there, you'll learn how to import your own models or use ones acquired from the Asset Store. You'll finish this hour by examining Unity's material and shader functionality.

The Basics of Models

Video games wouldn't be very *video* without the graphical components. In 2D games, the graphics consist of flat images called *sprites*. In 2D games, all you need to do is change the x and y positions of sprites and flip several of them in sequence, and the viewer's eye is fooled into believing that it sees true motion and animation. In 3D games, however, things aren't so simple. In worlds with a third axis, objects need to have volume to fool the eye. Games use a large number of objects; therefore, they need to process things quickly. Meshes make that possible. A mesh, at its most simple, is a series of interconnected triangles. These triangles build off each other in strips to form basic to very complex objects. These strips provide the 3D definitions of a model and can be processed very quickly. Don't worry, though: Unity handles all this for you so that you

don’t have to manage it yourself. Later in this hour, you’ll see just how triangles can make up various shapes in the Unity Scene view.

NOTE

Why Triangles?

You might be wondering why 3D objects are made up entirely of triangles. The answer is simple: Computers process graphics as a series of points, otherwise known as *vertices*. The fewer vertices an object has, the faster it can be drawn. Triangles have two properties that make them desirable. The first is that whenever you have a single triangle, you need only one more vertex to make another. So to make one triangle, you need three vertices, to make two triangles you need only four, and to make three triangles you need only five. This makes triangles very efficient. The second property that makes triangles desirable is that by making strips of triangles, you can model any 3D object. No other shape affords you that level of flexibility and performance.

NOTE

Terminology: Model or Mesh?

The terms *model* and *mesh* are similar, and you can often use them interchangeably. There is a difference, however. A mesh contains all the points and lines that define the 3D shape of an object. When you refer to the shape or form of a model, you are really referring to a mesh. A mesh can also be called a *model’s geography*, or its *geo*. A model, therefore, is an object that contains a mesh.

A model has a mesh to define its dimensions, but it can also contain animations, textures, materials, shaders, and other meshes. A good general rule is this: If the item in question contains anything other than vertex information, it is a model; otherwise, it is a mesh.

NOTE

What About 2D?

This hour covers a lot of information about rendering 3D objects. While that is useful, what if you only want to make 2D games? Should you still bother with this lesson (or any others that don’t cover 2D)? Of course! As

mentioned in Hour 2, “Game Objects,” there aren’t really 2D games anymore. Sprites are just textures applied to flat 3D objects. Lighting can be applied to 2D objects. Even the cameras used for 2D and 3D games are identical. All the concepts learned here can be applied directly to your 2D games because 2D games are really 3D games. As you practice more with Unity, you will quickly come to see that the line between 2D and 3D is very blurred indeed!

Built-in 3D Objects

Unity comes with a few basic built-in meshes (or primitives) for you to work with. They tend to be simple shapes that serve simple utilities or can be combined to make more complex objects. Figure 3.1 shows the available built-in meshes. (You worked with the cube and sphere in the previous hours.)

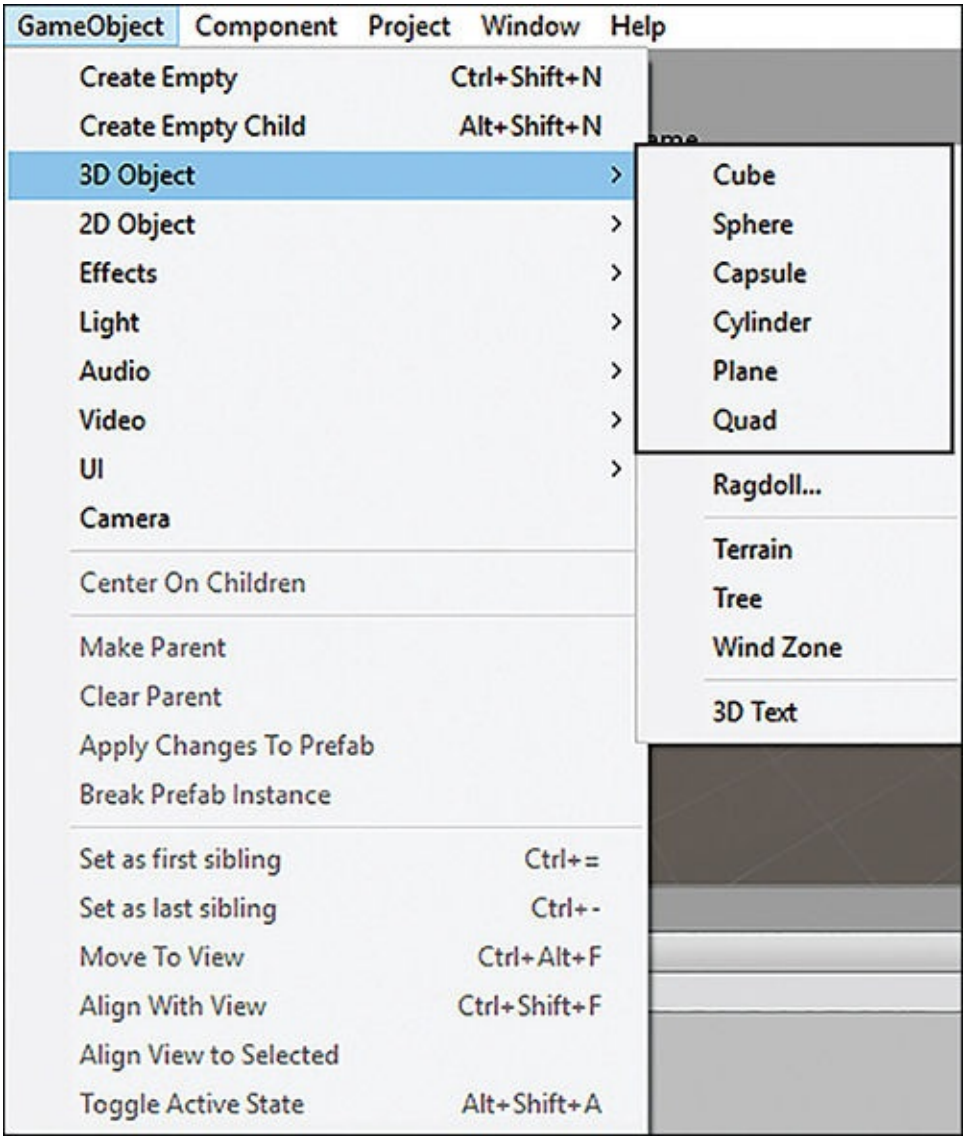

FIGURE 3.1
The built-in meshes in Unity.

TIP

Modeling with Simple Meshes

Do you need a complex object in your game but can't find the right type of model to use? By nesting objects in Unity, you can easily make simple models using the built-in meshes. Just place the meshes near each other so that they form the rough look you want. Then nest all the objects under one central object. This way, when you move the parent, all the children move, too. This might not be the prettiest way to make models for a game, but it will do in a pinch and is useful for prototyping!

Importing Models

Having built-in models is nice, but most of the time your games will require art assets that are a little more complex. Thankfully, Unity makes it rather easy to bring your own 3D models into your projects. Just placing a file containing a 3D model in your Assets folder is enough to bring it into the project. From there, you can drag it into the scene or hierarchy to build a game object around it. Natively, Unity supports .fbx, .dae, .3ds, .dxf, and .obj files. This enables you to work with just about any 3D modeling tool.

▼ TRY IT YOURSELF

Importing Your Own 3D Model

Follow these steps to bring a custom 3D modes into a Unity project:

1. Create a new Unity project or scene.
2. In the Project view, create a new folder named **Models** under the Assets folder by right-clicking the Assets folder and selecting **Create > Folder**.
3. Locate the Torus.fbx file provided for you in the Hour 3 folder of the book files.
4. With the operating system's file browser and the Unity editor open and side-by-side, click and drag the Torus.fbx file from the file browser into the Models folder that you created in step 2. In Unity, click the **Models** folder to see the new Torus file. If you do this correctly, your Project view should resemble Figure 3.2. (*Note:* If you are using an earlier version of Unity or have changed your editor settings, you may also have a Materials folder automatically created. If this is the case, it is not a problem. Materials are discussed in greater detail later in this hour.)

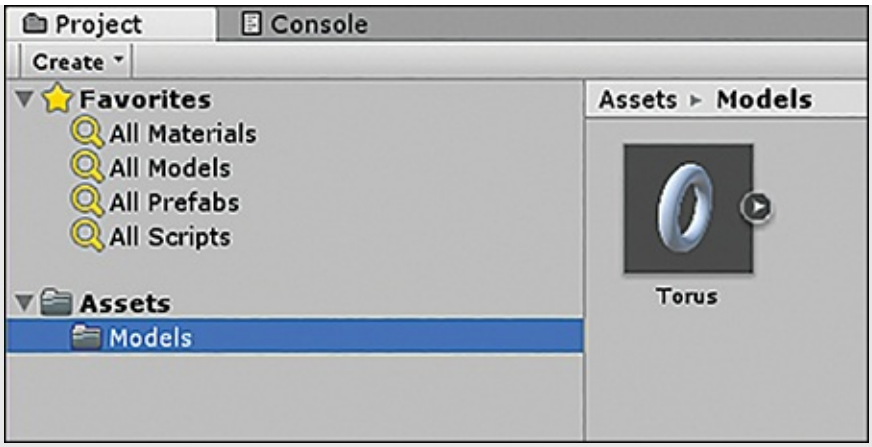

FIGURE 3.2
The Project view after the Torus model is added.

5. Drag the Torus asset from the Models folder onto the Scene view. Notice how a Torus game object is added to the scene containing a mesh filter and mesh renderer. These allow the Torus to be drawn to the screen.

6. Currently the torus in the scene is very small, so select the Torus asset in the project view and look at the Inspector view. Change the value of the scale factor from 1 to 100 and click the **Apply** button at the bottom right of the Inspector.

CAUTION

Default Scaling of Meshes

Most of the Inspector view options for meshes are advanced and are not covered right now. The property you are interested in is the scale factor. 3D software deals in generic units, and each software can use a different scale for units. By default, Unity treats one generic unit as one meter; another software, such as Blender, might treat one unit as a centimeter. Normally when you import a model, based on the file type, Unity can guess the scale at which to import. Sometimes, however (as in this case), this doesn't work, and you can fine-tune it with the scale factor. By changing the value of the scale factor from 1 to 100, you are telling Unity to import the model 100 times larger, which converts the centimeters of the model into the meters of Unity.

Models and the Asset Store

You don’t have to be an expert modeler to make games with Unity. The Asset Store provides a simple and effective way to find premade models and import them into your projects. Generally speaking, models in the Asset Store are either free or paid and come alone or in a collection of similar models. Some of the models come with their own textures, and some of them are simply the mesh data.

TRY IT YOURSELF

Downloading Models from the Asset Store

Let’s look at how to find and download models from Unity’s Asset Store. Follow these steps to acquire a model named Robot Kyle and import it into your scene:

1. Create a new scene (by selecting **File > New Scene**). In the Project view, type **Robot Kyle t: Model** in the search bar (see Figure 3.3).

FIGURE 3.3
Locating a model asset.

2. In the search filter section, click the **Asset Store** button (refer to Figure 3.3). If these words aren’t visible, you may need to resize your editor window or Project view window. You also need to be connected to the Internet.

3. Locate Robot Kyle, published by Unity Technologies. You can see the publisher by clicking on an asset and checking the Inspector view (see Figure 3.4).

FIGURE 3.4
The asset Inspector view.

4. In the Inspector view, click **Import Package**. At this point, you may be prompted to provide your Unity account credentials. In case you have trouble with this, I have also provided the file in this hour's book files as a Unity package; simply double-click it to import it.

5. When the Importing Package dialog opens, leave everything checked

and click **Import**.

6. Locate the robot model under Assets\Robot Kyle\Model and drag it into the Scene view. Note that the model will be fairly small in the Scene view; you might need to move closer to see it.

Textures, Shaders, and Materials

Applying graphical assets to 3D models can be daunting for beginners who are not familiar with the process. Unity uses a simple and specific workflow that gives you a lot of power in determining exactly how you want things to look. Graphical assets are broken down into textures, shaders, and materials. Each of these is covered individually in its own section, and Figure 3.5 shows how they fit together. Notice that textures are not applied directly to models. Instead, textures and shaders are applied to materials. Those materials are in turn applied to the models. This way, the look of a model can be swapped or modified quickly and cleanly without a lot of work.

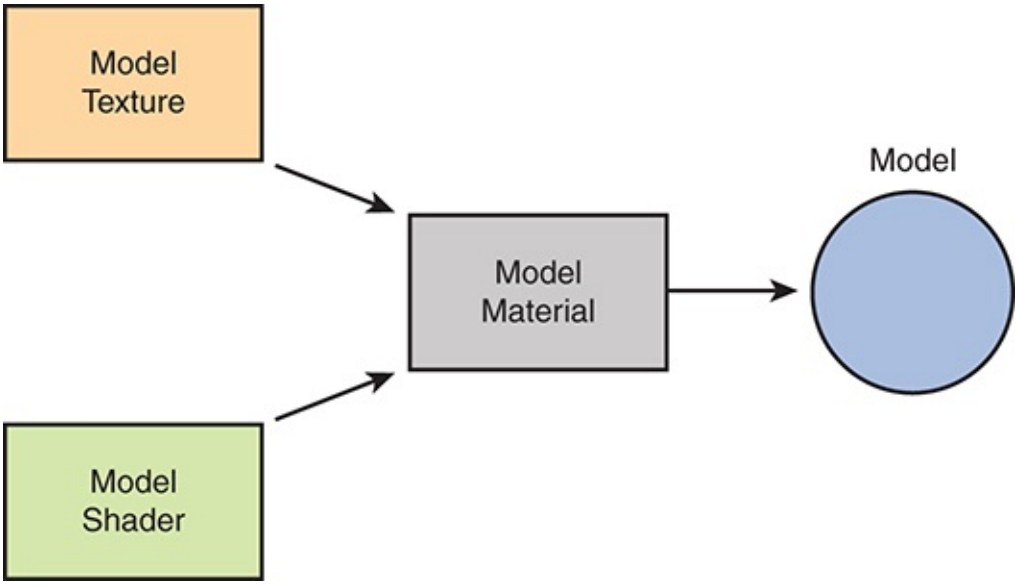

FIGURE 3.5
The model asset workflow.

Textures

Textures are flat images that get applied to 3D objects. They are responsible for models being colorful and interesting instead of blank and boring. It can be strange to think that a 2D image can be applied to a 3D model, but it is a fairly

strange to think that a 2D image can be applied to a 3D model, but it is a fairly straightforward process once you are familiar with it. Think about a soup can for a moment. If you were to take the label off the can, you would see that it is a flat piece of paper. That label is like a texture. After a label is printed, it is wrapped around a 3D can to provide a more pleasing look.

Just as with all other assets, adding textures to a Unity project is easy. Start by creating a folder for your textures; a good name would be Textures. Then drag any textures you want in your project into the Textures folder you just created. That's it!

NOTE

That's an Unwrap!

Imagining how textures wrap around cans is fine, but what about more complex objects? When creating an intricate model, it is common to generate something called an *unwrap*. The unwrap is somewhat akin to a map that shows exactly how a flat texture will wrap back around a model. If you look in the Textures folder under the Robot Kyle folder from earlier this hour, you see the Robot_Color texture. It looks strange, but that is the unwrapped texture for the model. The generation of unwraps, models, and textures is an art form and is not covered in this text. A preliminary knowledge of how it works should suffice at this level.

CAUTION

Weird Textures

Later in this hour, you will apply some textures to models. You might notice that the textures warp a bit or get flipped in the wrong direction. Just know that this is not a mistake or an error. This occurs when you take a basic rectangular 2D texture and apply it to a model. The model has no idea which way is correct, so it applies the texture however it can. If you want to avoid this issue, use textures specifically designed for (that is, unwrapped for) the model you are using.

Shaders

Whereas the texture of a model determines what is drawn on its surface, the shader determines *how* it is drawn. Here's another way to look at this: A material

is the interface between you and a shader. The material tells you what the shader needs to render an object, and you provide those items to make it look the way you want. This might seem nonsensical right now, but later, when you create materials, you will begin to understand how they work. Much of the information about shaders is covered later this hour because you cannot use a shader without a material. In fact, much of the information to be learned about materials is actually about materials' shaders.

TIP

Thought Exercise

If you are having trouble understanding how a shader works, consider this scenario: Imagine you have a piece of wood. The physicality of the wood is its mesh; the color, texture, and visible element are its texture. Now take that piece of wood and pour water on it. The wood still has the same mesh. It is still made of the same substance (wood). It looks different, though. It is slightly darker and shiny. In this example, you have two "shaders": dry wood and wet wood. The wet wood "shader" had something added that made it look a little different without actually changing it.

Materials

As mentioned earlier, materials are not much more than containers for shaders and textures that can be applied to models. Most of the customization of materials depends on which shader is chosen for it, although all shaders have some common functionality.

To create a new material, start by making a Materials folder. Then right-click the folder and select **Create > Material**. Give your material some descriptive name, and you are done. Figure 3.6 shows two materials with different shader settings. Notice how they both use the same Standard shader. Each has a base albedo color of white (more about albedo later), but they have different Smoothness settings. The Flat material has a low Smoothness, so the lighting looks very flat because the light bounces in a lot of directions. The Shiny material has a higher Smoothness, which creates a more focused light bounce. For both materials, there is also a preview of the material, so you can see what it will look like once it is on a model.

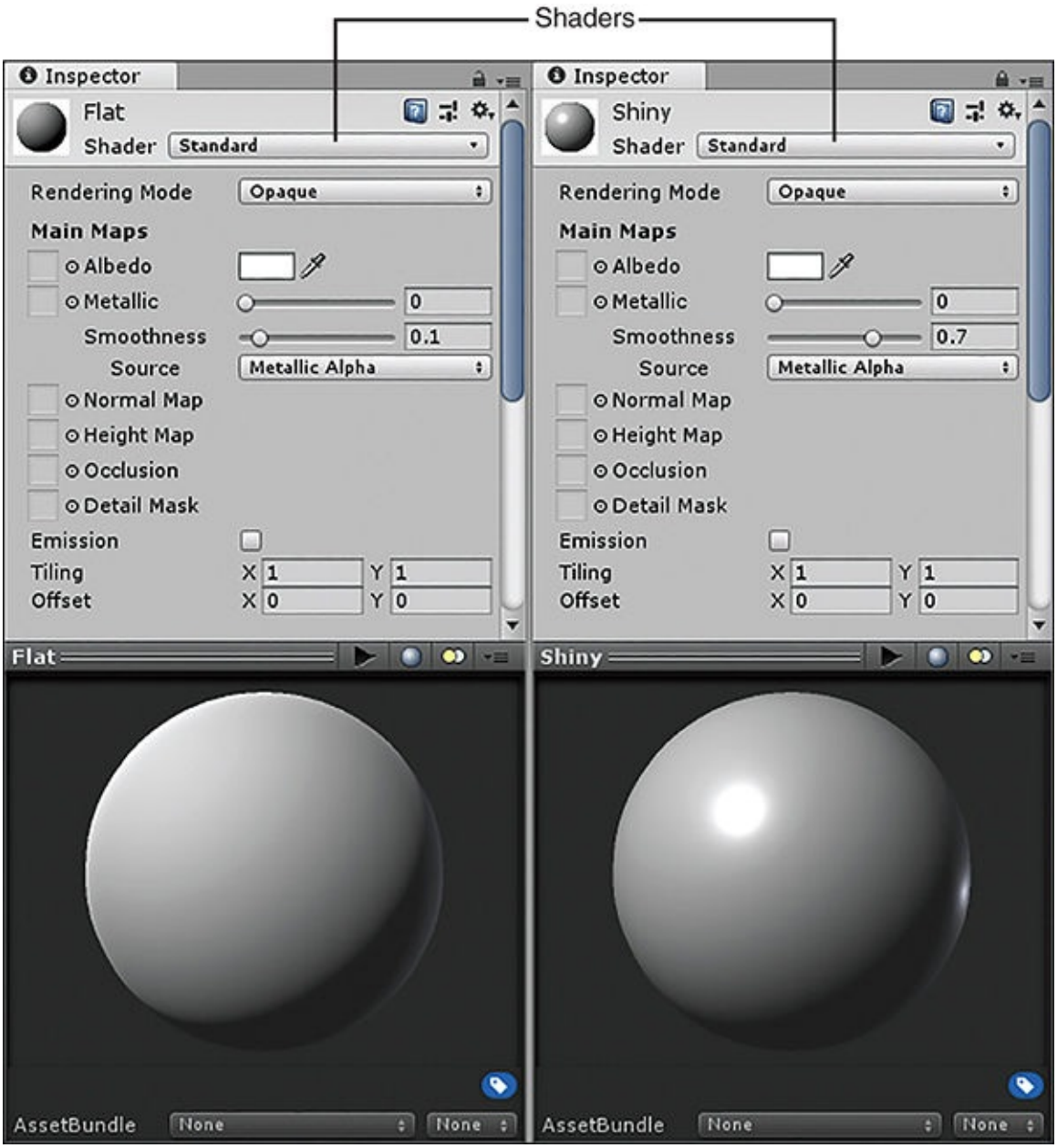

FIGURE 3.6
Two materials with different shader settings.

Shaders Revisited

Now that you have been introduced to textures, models, and shaders, it is time to look at how it all comes together. Unity has a very powerful Standard shader, which this book focuses on. Table 3.1 describes the common shader properties. In addition to the settings listing in Table 3.1, there are many other settings of the Standard shader; however, this book focuses on mastering the ones listed in

the table.

TABLE 3.1 Common Shader Properties

Property	Description
Albedo	The Albedo property defines the base color of the object. With Unity's powerful physically based rendering (PBR) system, this color interacts with light as a real object would. For example, a yellow albedo will look yellow in white light but green under blue light. This is where you would apply a texture that contains the color information about your model.
Metallic	This setting does exactly what it sounds like it does: It changes how metallic the material looks. This setting can also take in a texture as a "map" of how metallic different parts of the model appear. For realistic results, set this property to either 0 or 1.
Smoothness	Smoothness is a key element in PBR as it represents various micro imperfections, details, marks, and age that control how smooth (or rough) a surface is. The result is that it makes a model look more or less shiny. This property uses the same texture map as the Metallic property. For realistic results, avoid extreme values like 0 and 1.
Normal Map	The Normal Map property contains the normal map that will be applied to the model. A normal map can be used to apply relief, or bumps, to a model. This is useful when calculating lighting to give the model more detail than it would otherwise have.
Tiling	The Tiling property defines how often a texture can repeat on a model. It can repeat in both the x and y axes. (Remember that textures are flat, which is why there is no z tiling axis.)
Offset	The Offset property defines where in the x and y axes of a texture to begin applying.

This might seem like a lot of information to take in, but once you become more familiar with the few basics of textures, shaders, and materials, you'll find that the Smoothness value of your understanding goes to 1 (classic comedy).

Unity also has several other shaders that this book doesn't cover. The Standard shader is very flexible and should work for most of your basic needs.

TRY IT YOURSELF

Applying Textures, Shaders, and Materials to Models

Follow these steps to put all your knowledge of textures, shaders, and materials together to create a decent-looking brick wall:

1. Start a new project or scene. Note that creating a new project causes the editor to close and reopen.
2. Create a **Textures** folder and a **Materials** folder.
3. Locate the Brick_Texture.png file in the book files and drag it into the Textures folder created in step 2.
4. Add a cube to the scene. Position it at (0, 1, -5). Give it a scale of (5, 2, 1). Figure 3.7 shows the cube's properties.

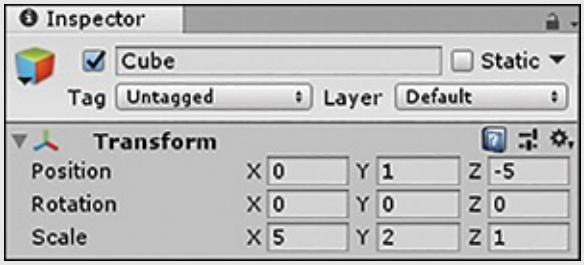

FIGURE 3.7
The properties of the cube.

5. Create a new material (by right-clicking the Materials folder and selecting **Create** > **Material**) and name it **BrickWall**.
6. Leave the shader as Standard, and under Main Maps click the circle selector (the little circle icon) to the left of the word Albedo. Select **Brick_Texture** from the pop-up window.
7. Click and drag the brick wall material from the Project view onto the cube in the Scene view.
8. Notice that the texture is stretched across the wall a little too much. With the material selected, change the value of the x tiling to be **3**. Make sure you do this in the Main Maps section, not the Secondary Maps section. Now the wall looks much better. You now have a textured brick wall in your scene. Figure 3.8 shows the final product.

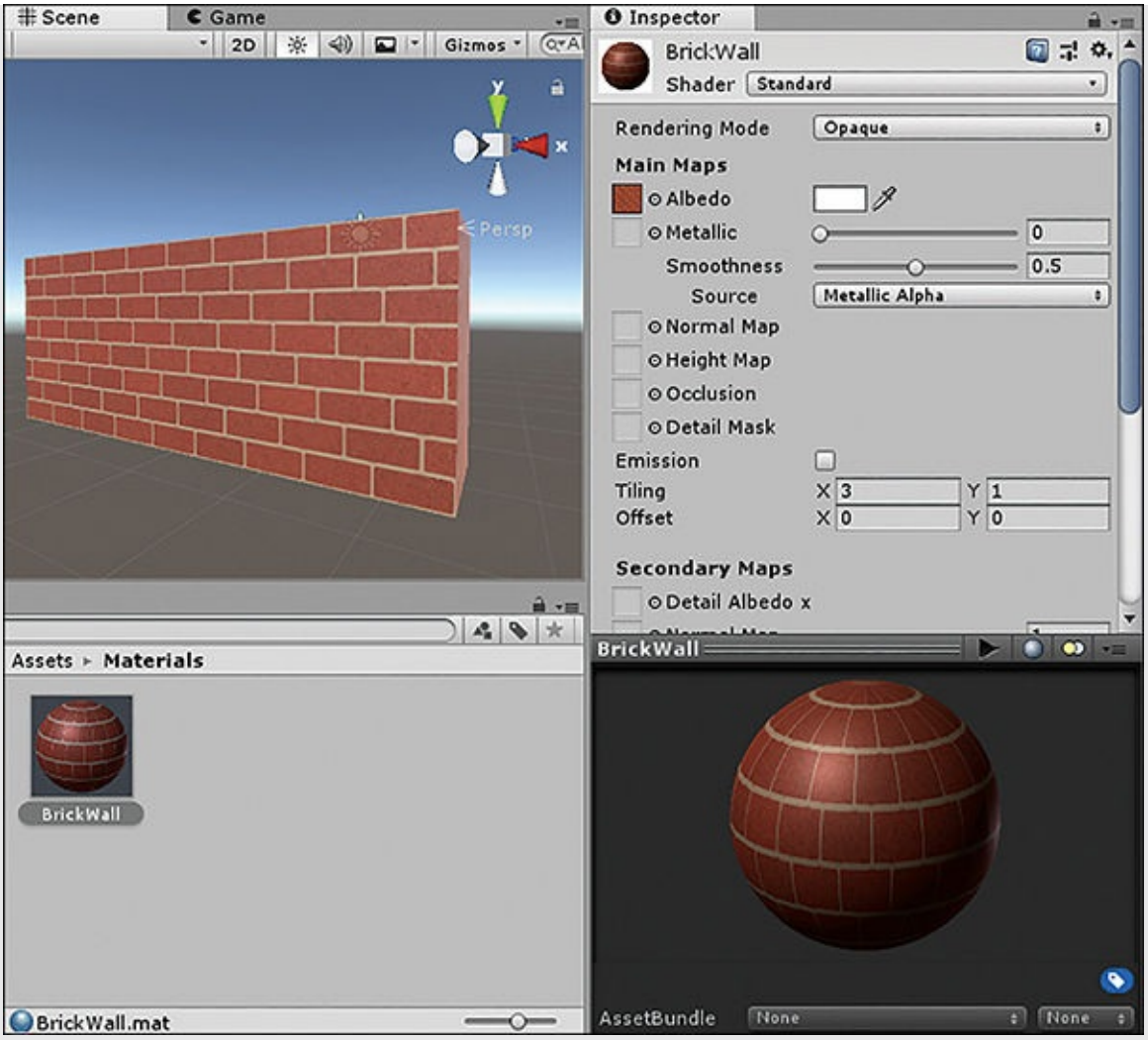

FIGURE 3.8
The final product of this Try It Yourself.

Summary

In this hour, you have learned all about models in Unity. You started by learning about how models are built with collections of vertices called meshes. Then you discovered how to use the built-in models, import your own models, and download models from the Asset Store. You then learned about the model art workflow in Unity. You experimented with textures, shaders, and materials. You finished by creating a textured brick wall.

Q&A

Q. Will I still be able to make games if I'm not an artist?

A. Absolutely. Using free online resources and the Unity Asset Store, you can find various art assets to put in your games.

Q. Do I need to know how to use all the built-in shaders?

A. Not necessarily. Many shaders are very situational. Start with the Standard shader covered in this lesson and learn more if a game project requires it.

Q. If there are paid art assets in the Unity Asset Store, does that mean I can sell my own art assets?

A. Yes, it does. In fact, the Asset Store is not limited to only art assets. If you can create high-quality assets, you can certainly sell them in the store.

Workshop

Take some time to work through the questions here to ensure that you have a firm grasp of the material.

Quiz

1. True or False: Because of their simple nature, squares make up meshes in models.
2. What file formats does Unity support for 3D models?
3. True or False: Only paid models can be downloaded from the Unity Asset Store.
4. Explain the relationship between textures, shaders, and materials.

Answers

1. False. Meshes are made up of triangles.
2. .fbx, .dae, .3ds, .dxf, and .obj files
3. False. There are many free models.
4. Materials contain textures and shaders. Shaders dictate the properties that can be set by a material and how the material gets rendered.

Exercise

In this exercise, you'll experiment with the effects shaders have on the way

models look. You will use the same mesh and texture for each model; only the shaders will be different. The project created in this exercise is named Hour 3_Exercise and is available in the Hour 3 book files.

1. Create a new scene or project.
2. Add a **Materials** folder and a **Textures** folder to your project. Locate the files Brick_Normal.png and Brick_Texture.png in the Hour 3 book files and drag them into the Textures folder.
3. In the Project view, select Brick_Texture. In the Inspector view, change the aniso level to **3** to increase the texture quality for curves. Click **Apply**.
4. In the Project view, select Brick_Normal. In the Inspector view, change the texture type to **Normal Map**. Click **Apply**.
5. Select Directional Light in the Hierarchy view and give it the position (0, 10, -10) and the rotation (30, -180, 0).
6. Add four spheres to your project. Scale them each to (2, 2, 2). Spread them out by giving them the positions (1, 2, -5), (-1, 0, -5), (1, 0, -5), and (-1, 2, -5).
7. Create four new materials in the Materials folder. Name them **DiffuseBrick**, **SpecularBrick**, **BumpedBrick**, and **BumpedSpecularBrick**. Figure 3.9 shows all the properties of the four materials. Go ahead and set them as shown.

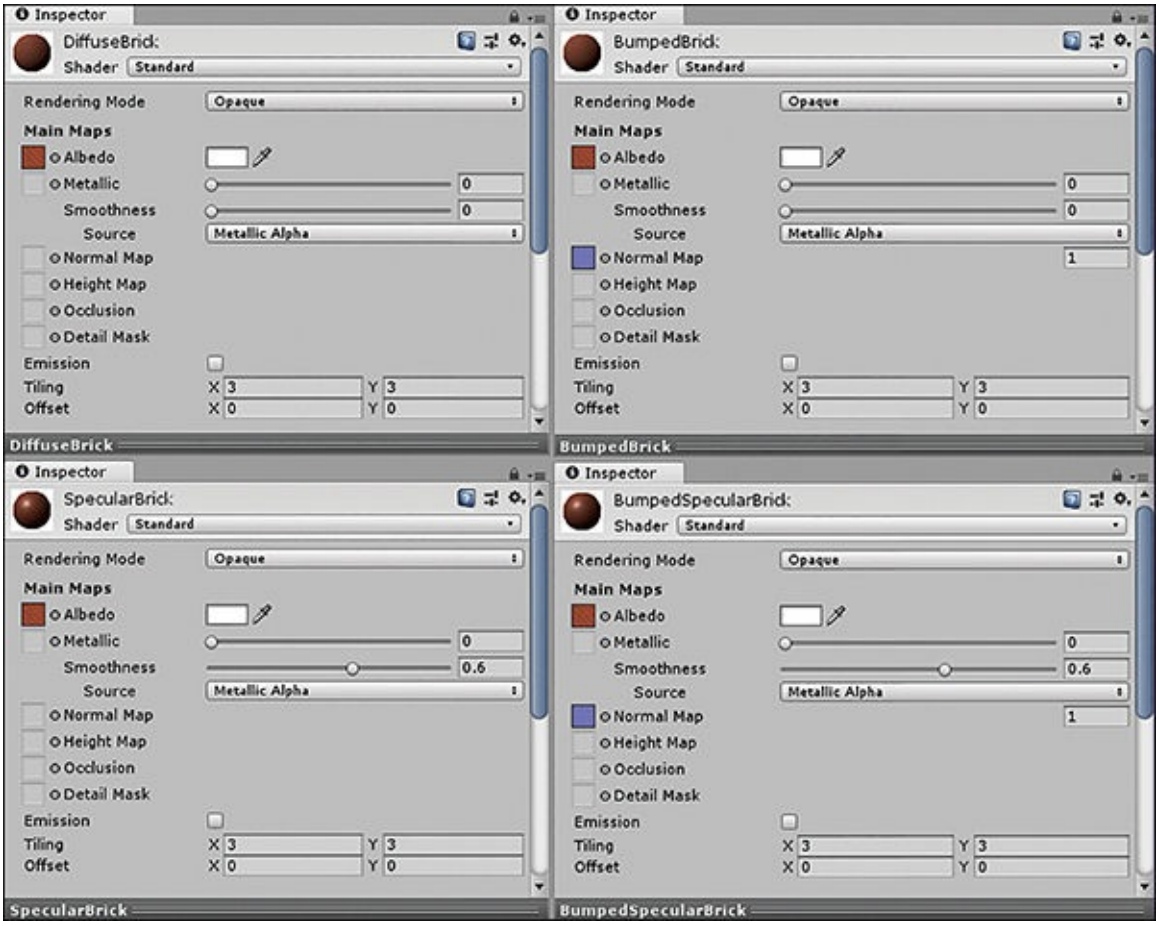

FIGURE 3.9
Material properties.

8. Click and drag each of the materials onto one of the four spheres. Notice how the light and the curvature of the spheres interact with the different shaders. Remember that you can move about the Scene view to see the spheres from different angles.