Mike Geig

Sams Teach Yourself

Unity® Game Development in 24 Hours

SAMS

Mike Geig

Sams **Teach Yourself**

Unity® Game Development

in 24 Hours

SAMS 800 East 96th Street, Indianapolis, Indiana, 46240 USA

Sams Teach Yourself Unity® Game Development in 24 Hours

ISBN-13: 978-0-672-33696-6
ISBN-10: 0-672-33696-6

Library of Congress Control Number: 2013950040

Printed in the United States of America

Second Printing: November 2014

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Bulk Sales

Sams Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales
1-800-382-3419
corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact

International Sales
international@pearsoned.com

Editor-in-Chief
Mark Taub

Executive Editor
Laura Lewin

Senior Development Editor
Chris Zahn

Managing Editor
Kristy Hart

Project Editor
Andy Beaster

Copy Editor
Keith Cline

Indexer
Brad Herriman

Proofreader
Sheri Cain

Technical Editors
Tim Harrington
Valerie Shipbaugh
Jeff Somers

Publishing Coordinator
Olivia Basegio

Interior Designer
Gary Adair

Cover Designer
Mark Shirar

Compositor
Gloria Schurick

Contents at a Glance

Table of Contents

Preface

The Unity game engine is an incredibly powerful and popular choice for professional and amateur game developers alike. This book has been written to get readers up to speed and working in Unity as fast as possible (about 24 hours to be exact) while covering fundamental principles of game development. Unlike other books that only cover specific topics or spend the entire time teaching a single game, this book covers a large array of topics while still managing to contain four games! Talk about a bargain. By the time you are done reading this book, you won't have just theoretical knowledge of the Unity game engine. You will have a portfolio of games to go with it.

Who Should Read This Book

This book is for anyone looking to learn how to use the Unity game engine. Whether you are a student or a development expert, there is something to learn in these pages. It is not assumed that you have any prior game development knowledge or experience, so don't worry if this is your first foray into the art of making games. Take your time and have fun. You will be learning in no time.

How This Book Is Organized and What It Covers

Following the Sam's Teach Yourself approach, this book is organized into 24 chapters that should take approximately 1 hour each to work through. The chapters include the following:

- Hour 1, "Introduction to Unity": This hour gets you up and running with the various components of the Unity game engine.
- Hour 2, "Game Objects": Hour 2 teaches you how to use the fundamental building blocks of the Unity game engine: the game object. You also learn about coordinate systems and transformations.
- Hour 3, "Models, Materials, and Textures": In this hour, you learn to work with Unity's graphical asset pipeline as you apply shaders and textures to materials. You also learn how to apply those materials to a variety of 3D objects.
- Hour 4, "Terrain": In Hour 4, you learn to sculpt game worlds using Unity's terrain system. Don't be afraid to get your hands dirty as you dig around and create unique and stunning landscapes.

- Hour 5, "Environments": In this hour, you learn to apply environmental effects to your sculpted terrain. Time to plant some trees!
- Hour 6, "Lights and Cameras": Hour 6 covers lights and cameras in great detail.
- Hour 7, "Game 1: *Amazing Racer*": Time for your first game. In Hour 7, you create Amazing Racer, which requires you to take all the knowledge you have gained so far and apply it.
- Hour 8, "Scripting Part 1": In Hour 8, you begin your foray into scripting with Unity. If you've never programmed before, don't worry. We go slowly as you learn the basics.
- Hour 9, "Scripting Part 2": In this hour, you expand on what you learned in Hour 8. This time, you focus on more advanced topics.
- Hour 10, "Collision": Hour 10 walks you through the various collision interactions that are common in modern video games. You learn about physical as well as trigger collisions. You also learn to create physical materials to add some variety to your objects.
- Hour 11, "Game 2: *Chaos Ball*": Time for another game! In this hour, you create Chaos Ball. This title certainly lives up to its name as you implement various collisions, physical materials, and goals. Prepare to mix strategy with twitch reaction.
- Hour 12, "Prefabs": Prefabs are a great way to create repeatable game objects. In Hour 12, you learn to create and modify prefabs. You also learn to build them in scripts.
- Hour 13, "Graphical User Interfaces": In Hour 13, you learn to implement graphical user interfaces (GUIs) in Unity. You learn the various components and how to position them on a 2D interface.
- Hour 14, "Character Controllers": In this hour, you learn how to create your own character controllers. You finish up the chapter by building your own custom controller.
- Hour 15, "Game 3: *Captain Blaster*": Game number 3! In this hour, you make Captain Blaster, a retro-style spaceship shooting game.
- Hour 16, "Particle Systems": Time to learn about particle effects. In this chapter, you experiment with Unity's legacy particle system and its new Shuriken particle system. You learn how to create cool effects and apply them to your projects.
- Hour 17, "Animations": In Hour 17, you get to learn about animations and Unity's legacy animation system. You experiment with bringing models to life using assets from the Asset Store.
- Hour 18, "Animators": Hour 18 is all about Unity's new Mecanim animation system. You learn to remap model riggings and apply universal animations to them.
- Hour 19, "Game 4: *Gauntlet Runner*": Lucky game number 4 is called Gauntlet Runner. This game explores a new way to scroll backgrounds and how to implement animator controllers to build complex blended animations.

- Hour 20, "Audio": Hour 20 has you adding important ambient effects via audio. You learn about 2D and 3D audio and their different properties.
- Hour 21, "Mobile Development": In this hour, you learn how to build games for mobile devices. You also learn to utilize a mobile device's built-in accelerometer and multi-touch display.
- Hour 22, "Game Revisions": It's time to go back and revisit the four games you have made. This time you modify them to work on a mobile device. You get to see which control schemes translate well to mobile and which don't.
- Hour 23, "Polish and Deploy": Time to learn how to add multiple scenes and persist data between scenes. You also learn about the deployment settings and playing your games.
- Hour 24, "Wrap Up": Here, you look back and summarize the journey you went on to learn Unity. This hour provides useful information about what you have done and where to go next.

Unity Engine Versions

This book was made with the Unity engine version 4.1 and 4.2. The two different versions are nearly identical for your purposes, but do note that some visual elements might have shifted place. For example, in some of the screen images you may note a Terrain menu item in the menu bar at the top of the Unity editor. In version 4.2, that has been moved. Do not worry. All explanations involving the creation and management of terrain have been updated to illustrate the new process. I am just writing this here so that you are not confused if a couple of things look slightly different.

Thank you for reading my preface! I hope you enjoy this book and learn much from it. Good luck on your journey with the Unity game engine!

About the Author

Mike Geig is both an experienced teacher and game developer, with a foot firmly in both camps. He is currently teaching game design and development at Stark State College and the Cleveland Institute of Art. Mike also works as a screencaster for Unity Technologies and is a member of Unity's Learn department. His Pearson video, *Game Development Essentials with Unity 4 LiveLessons*, is a key title on Unity. Mike was once set on fire and has over a million "likes" on Facebook.

Dedication

To Dad: Everything worth learning, I learned from you.

Acknowledgments

A big "thank you" goes out to everyone who helped me write this book.

First and foremost, thank you Kara for keeping me on track. I don't know what we'll be talking about when this book comes out, but whatever it is, you are probably right. Love ya, babe.

Link and Luke: We should take it easy on mommy for a little while. I think she's about to crack.

Thanks to my parents. As I am now a parent myself, I recognize how hard it was for you not to strangle or stab me. Thanks for not strangling or stabbing me.

Thanks to Angelina Jolie. Due to your role in the spectacular movie Hackers (1995), I decided to learn how to use a computer. You underestimate the impact you had on 10-year-olds at the time. You're elite!

To the inventor of beef jerky: History may have forgotten your name, but definitely not your product. I love that stuff. Thanks!

Thank you to my technical editors: Valerie, Jim, and Tim. Your corrections and insights played a vital role in making this a better product.

Thank you, Laura, for convincing me to write this book. Also thank you for buying me lunch at GDC. I feel that lunch, the best of all three meals, specifically enabled me to finish this.

Finally, a "thank you" is in order for Unity Technologies. If you never made the Unity game engine, this book would be very weird and confusing.

We Want to Hear from You!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

When you write, please be sure to include this book's title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

Email: consumer@samspublishing.com

Mail: Sams Publishing
ATTN: Reader Feedback
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

This page intentionally left blank

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 learn all about models and how they are used in Unity. You start by looking at the fundamental principles of meshes and 3D objects. From there, you learn how to import your own models or use ones acquired from the Asset Store. You 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*. All you needed to do was change the x and y positions of these sprites and flip several of them in sequence and the viewer's eye was fooled into believing that it saw 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. Because games use a large number of objects, the need to process things quickly was very important. Enter the mesh. A mesh, at its most simple, is a series of interconnected triangles. These triangles build off of 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 of 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 asking yourself why 3D objects are made up entirely of triangles. The answer is simple. Computers process graphics as a series of point, 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. To make one triangle, you need three vertices, two triangles take only four, and three triangles require only five. This makes them very efficient. The second is that by using this practice of making strips of triangles, you can model any 3D object. No other shape affords you that level of flexibility and performance.

NOTE

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 vertex information that defines 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 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.

Built-In 3D Objects

Unity comes with a few basic built-in meshes (or primitives) for you work with. These 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.)

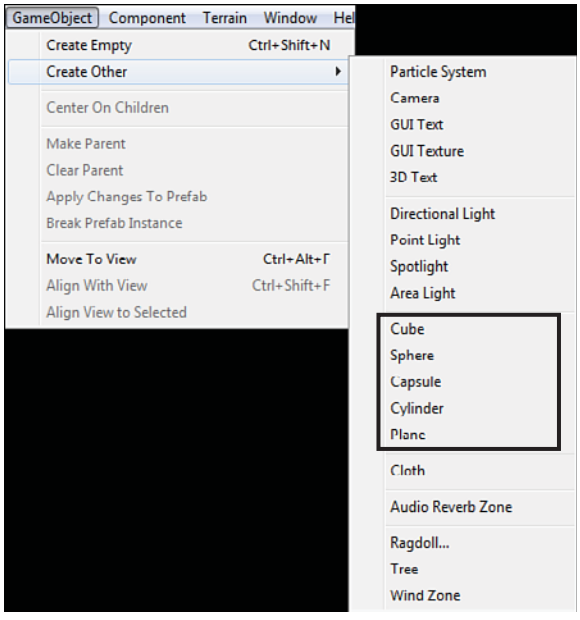

FIGURE 3.1
The built-in meshes in Unity.

TIP

Modeling with Simple Meshes

Do you need a complex object in your game but you can't find the right type of model to use? Nesting objects in Unity enables you to 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 your game, but it will do in a pinch!

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 the file containing the 3D model in your Assets folder is enough to bring it into the project. From there, dragging it into the scene or hierarchy builds 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

Let's walk through the steps required to bring custom 3D models 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. (Right-click the Assets folder and select **Create > Folder**.)
3. Locate the Torus.fbx file provided for you in the Hour 3 folder of the book files.
4. With both 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 done correctly, your Project view will resemble Figure 3.2. Notice the Materials folder that was added for you. You will learn more about this later.

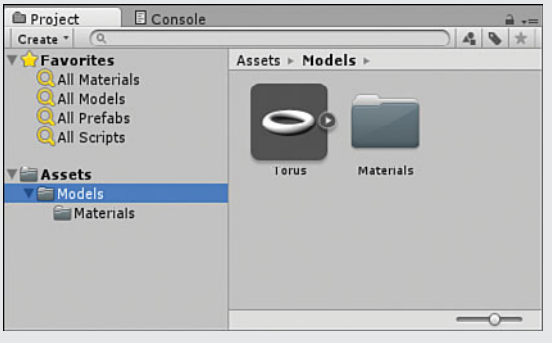

FIGURE 3.2
The Project view after the Torus model was added.

5. Click the **Torus** asset in the Models folder and look at the Inspector view. Change the value of the scale factor from 0.01 to 1 and click **Apply**.
6. Drag the Torus asset from the Models folder onto the Scene view. Notice how a Torus game object was added to the scene containing a mesh filter and mesh rendered. These allow the Torus to be drawn to the screen. If you click the **Torus** object, you see how it is made up of many connected triangles.

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. By default, Unity imports meshes scaled down. By changing the value of the scale factor from 0.01 to 1, you are telling Unity to allow the model to enter the scene as the same size as it was created.

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 project. Generally speaking, models on 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 learn how to find and download models from Unity's Asset Store. We will be acquiring a model named Robot Kyle and importing it into our scene:

1. Create a new scene (click **File > New Scene**). In the Project view, type **t:Model** into the search bar (see Figure 3.3).
2. In the search filter section, click the **Asset Store: 999+/999+** button (see Figure 3.3). If these words aren't visible, you may need to resize your editor window or Project view window.
3. Locate **Robot Kyle** and select it.

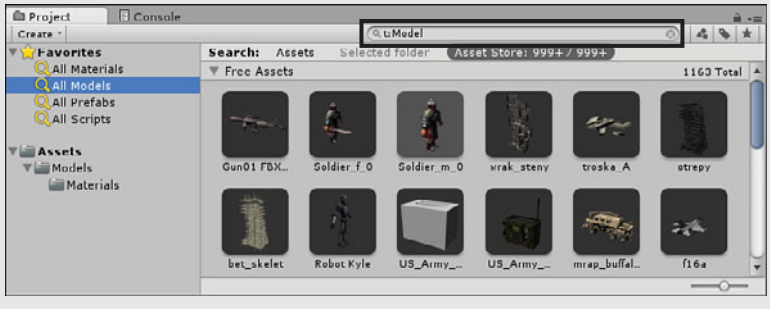

FIGURE 3.3
Steps to locate a model asset.

4. In the Inspector view, click **Import Package**. At this point, you may be prompted to provide your Unity account credentials.
5. When the Importing Package dialog opens, leave everything checked and select **Import**.
6. There will now be a new asset folder called Robot Kyle. Locate the robot model under **Assets > Robot Kyle > Model** and drag it into the Scene view (see Figure 3.4). Note that the model will be fairly small in the Scene view; you might need to move closer to see it.

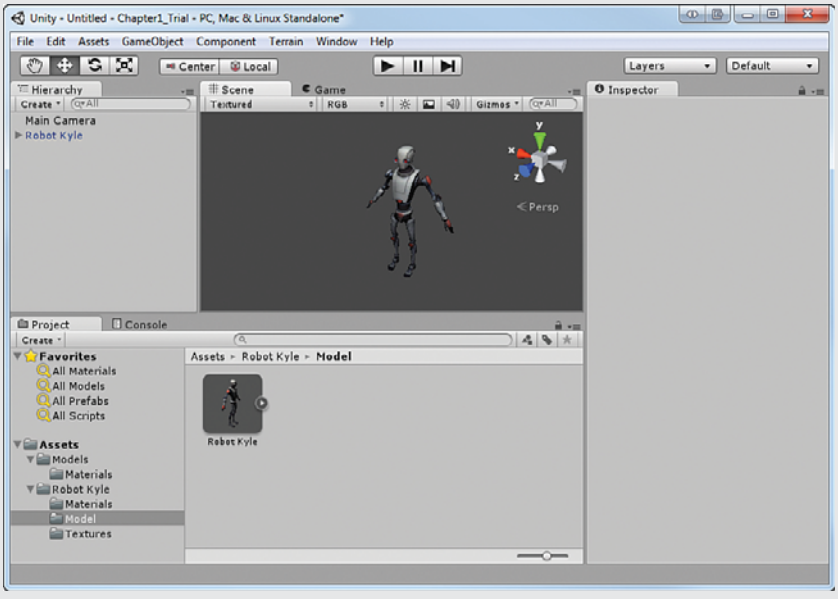

FIGURE 3.4
The Unity project with Robot Kyle Added.

Textures, Shaders, and Materials

Applying graphical assets to 3D models can be daunting if you are not familiar with it. Unity uses a simple and specific workflow that gives you a lot of power when 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, but Figure 3.5 shows you 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.

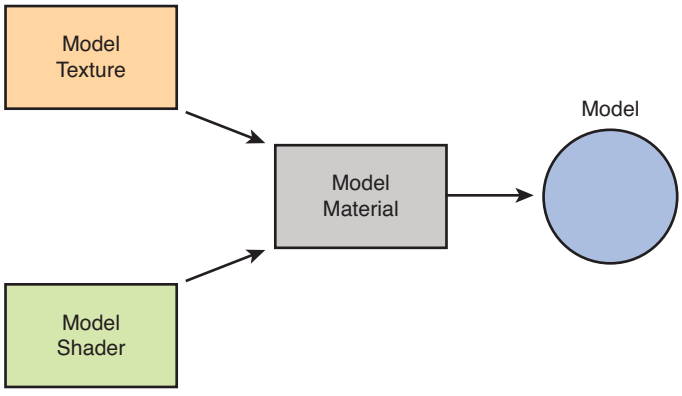

FIGURE 3.5
The model asset workflow.

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

Just like 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 you exactly how a flat texture will wrap back around a model. If you look in the Robot Kyle > Textures folder from earlier this hour, you notice 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 to itself 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 problem 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 (unwrapped for) the model that you are using.

Shaders

If the texture of a model determines what is drawn on its surface, the shader is what determines *how* it is drawn. Here's another way to look at this: A material contains properties and textures, and shaders dictate what properties and textures a material can have. This might seem nonsensical right now, but later when we create materials you will begin to understand how they work. Much of the information about shaders is covered later this hour, because you cannot create a shader without a material. In fact, much of the information to be learned about materials is actually about the material's shader.

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 still is made of the same substance (wood). It looks different, though. It is slightly darker and shiny. The water in this example is the shader. The shader took something and 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 shaders selected. Notice how they each have a base texture, main color, tilling and offsets, and a preview of the material (blank now because there is no texture). The Shiny material, however, uses a specular shader and comes with properties for specular color and shininess. All these properties are covered later in this hour.

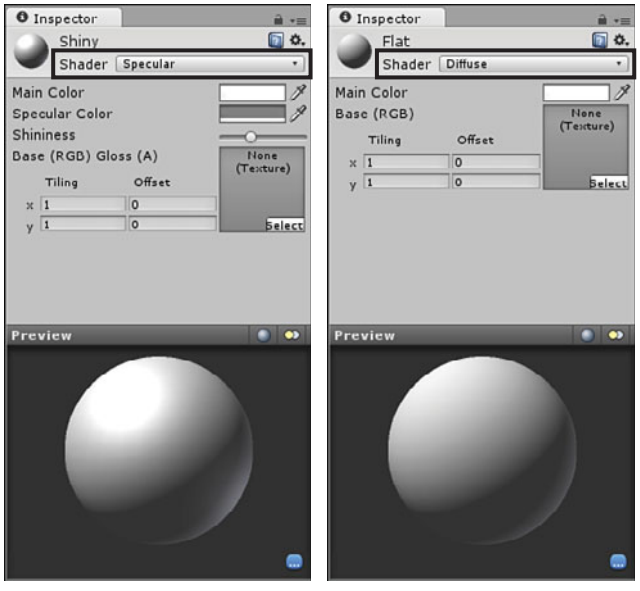

FIGURE 3.6
Two materials with different shaders.

Shaders Revisited

Now that you understand textures, models, and shaders, it is time to look at how it all comes together. Unity has a lot of built-in shaders, but this book is concerned with only a few of the Normal family of shaders. These shaders are the most basic and should be useful for everyone. Table 3.1 lists some of the basic shaders and describes them.

TABLE 3.1 Basic Normal Family of Shaders

Shader	Description
Diffuse	Diffuse is the default shader for materials and is also the most basic. Light is evenly distributed across the diffuse object's surface.
Specular	Specular textures make an object look shiny. If you want to make an object seem to reflect a lot of light, this is the shader to use.
Bumped	Bumped shaders are generally used in conjunction with other shaders (as in bumped-diffuse or bumped-specular). These shaders use a normal map to give the flat texture a 3D, or bumpy, look. These are a great way to give your models a lot of physical detail without requiring complex modeling.

Now that you are familiar with a few of the built-in shaders, it is time to look at some of the common shader properties that you will be working with. Table 3.2 describes the common shader properties.

TABLE 3.2 Common Shader Properties

Property	Description
Main Color	The Main Color property defines what color of ambient light shines on the object. This does not change the color of the object itself; it just makes the object appear different. For example, an object with a blue texture and a yellow main color will not turn yellow but green (because blue with yellow light looks green). If you want your model's color to remain unchanged, select white.
Specular Color	The Specular Color property determines what color the "shiny" parts of a specular model are. Generally speaking, this will be white unless you intend for it to appear as if another color of light is shining on the object.
Shininess	The Shininess property determines how shiny a specular object is.
Texture	The Texture block contains the texture you want to apply to your model.
Normal Map	The Normal Map block contains the normal map that will be applied to your model. A normal map can be used to apply bumpiness 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.
Offset	The Offset property defines whether a gap will exist between edges of the object and the texture.

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 this a smooth process.

TRY IT YOURSELF ▼

Applying Textures, Shaders, and Materials to Models

Let's put all of our knowledge of textures, shaders, and materials together and create a decent-looking brick wall:

1. Start a new project or scene. Note that creating a new project will close and reopen the editor.
2. Create a Textures 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). See Figure 3.7 for the cube properties.

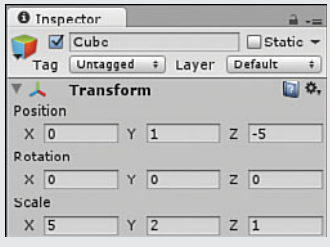

FIGURE 3.7
The properties of the cube.

5. Create a new material (right-click the Materials folder and select **Create > Material**) and name it **BrickWall**.
6. Leave the shader as Diffuse, and in the texture block click **Select**. 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 how 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**. Now the wall looks much better.
9. Add a directional light to your scene (click **GameObject > Create Other > Directional Light**). Position it at (0, 10, –10) and give it a rotation of (30, 0, 0). We will discuss lighting more in a later hour. This is just here to make your brick wall "pop."
10. You now have a textured brick wall in your scene. Figure 3.8 contains the final product.

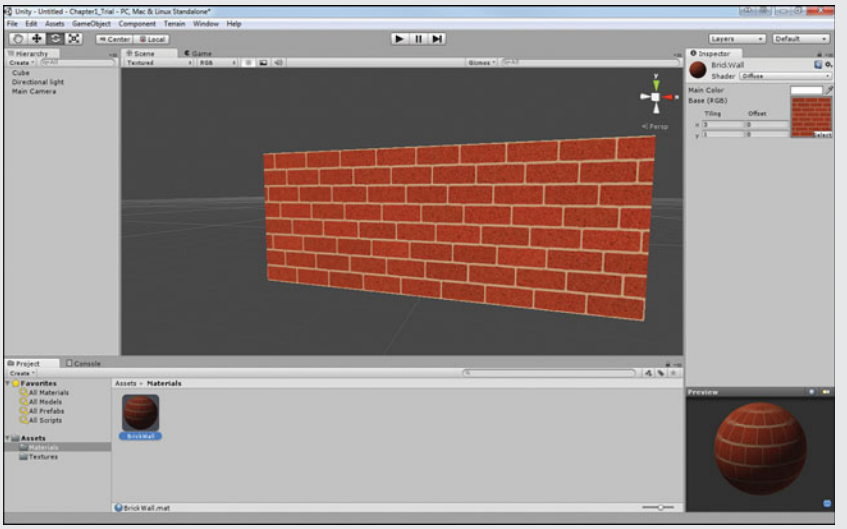

FIGURE 3.8
The final product of this Try It Yourself.

Summary

In this hour, you 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. Will I need to know how to use all the built-in shaders?

A. Not necessarily. Many shaders are very situational. Start with the shaders covered in this chapter 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, it 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 several free models.
4. Materials contain textures and shaders. Shaders dictate the properties that can be set by the material and how the material gets rendered.

Exercise

Let's 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 Hour3_Exercise and is available in the Hour 3 book files.

1. Create a new scene or project.
2. Add a Materials 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 from 1 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. Add a directional light to your project (click **GameObject > Create Other > Directional Light**) and give it a position of (0, 10, –10) with a rotation of (30, 40, 0).

6. Add four spheres to your project. Scale them each to (2, 2, 2). Spread them out by giving them positions of (2, 0, –5), (–2, 0, –5), (–2, 2, –5), and (2, 2, –5).

7. Create four new materials in the Materials folder. Name them **DiffuseBrick**, **SpecularBrick**, **BumpedBrick**, and **BumpedSpecularBrick**. Figure 3.9 contains all the properties of the four materials. Go ahead and set their values.

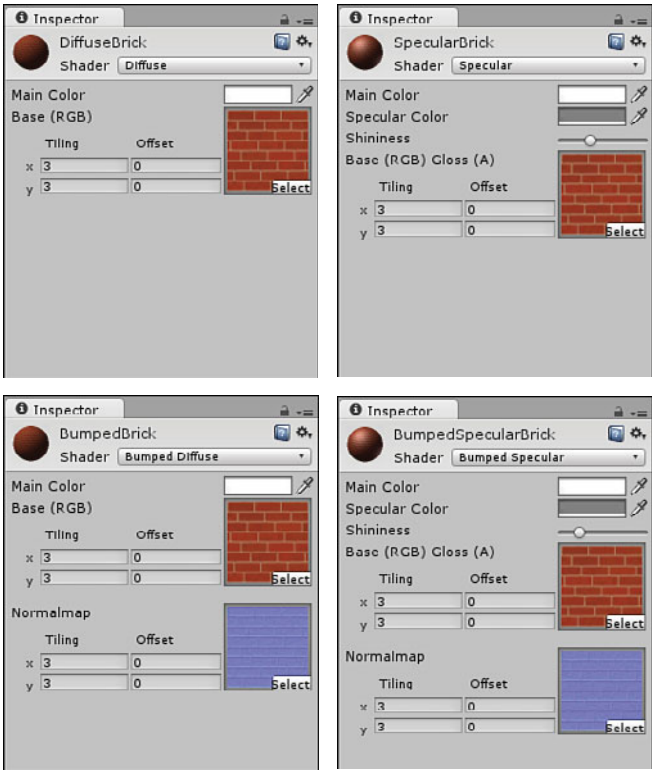

FIGURE 3.9
Material properties.

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 at different angles.

Index

Numbers

A

B

C

D

E

F

G

H

I

J-K

L

M

N

O

P

R

S

T

U

V

W-Z