CHAPTER 2

Unity UI Basics—Getting Started

As you are probably pretty anxious to get started creating something in Unity, the introduction to Unity's UI (user interface) will be kept rather brief. You will explore and investigate it more fully as the need arises, but for now, you can get a basic feel for the layout options and concepts. If you'd like to delve further into the UI before moving on, the Unity help files include a very good section on getting started.

Installing Unity and Starting Up

Download Unity from `http://unity3d.com/unity/download/` and install it. The Unity icon will appear on your desktop when the installation is complete (Figure 2-1). When you run it for the first time, you'll have to register the product by following the prompts. Registration can be done quickly online from any machine, even if the machine you've installed Unity on isn't connected to the Internet. If your machine is connected, you'll be taken directly to the registration page. A 30-day trial of the Pro version is available from Unity3D. As you will be introduced to a few of the Pro features in the last few chapters, you may wish to try the Pro version at that time.

Figure 2-1. *The Unity icon*

To start Unity, you can click the desktop icon or choose it from the Windows Start menu ➤ Programs ➤ Unity and, on the Mac, Applications ➤ Unity.

Loading or Creating a New Project or Scene

To work with Unity, you must always start with a project.

Tip The folder you create or select for your project *is* the project. When you open a project through the Project Wizard, you select the folder.

When you create a new project, a directory or folder is created, with sub-folders containing the required files. Be sure to note where the project is installed or use the Browse button to select a different location. You will be asked to choose which asset packages you want to include with your project. Earlier versions of Unity had generic Pro and Standard packages as the choices. Now, however, the packages have been split into component parts, so you can reduce clutter for specific game types. In the book, you will use most of the packages, so you can get a feel for what is available.

1. Open Unity.

If you have downloaded the demo project, Unity will open it to start. If there is no project, the Project Wizard dialog opens, prompting you to select a project.

2. If Unity has opened an existing project, from File, select Open New Project.
3. Select the Create New Project tab, as shown in Figure 2-2.

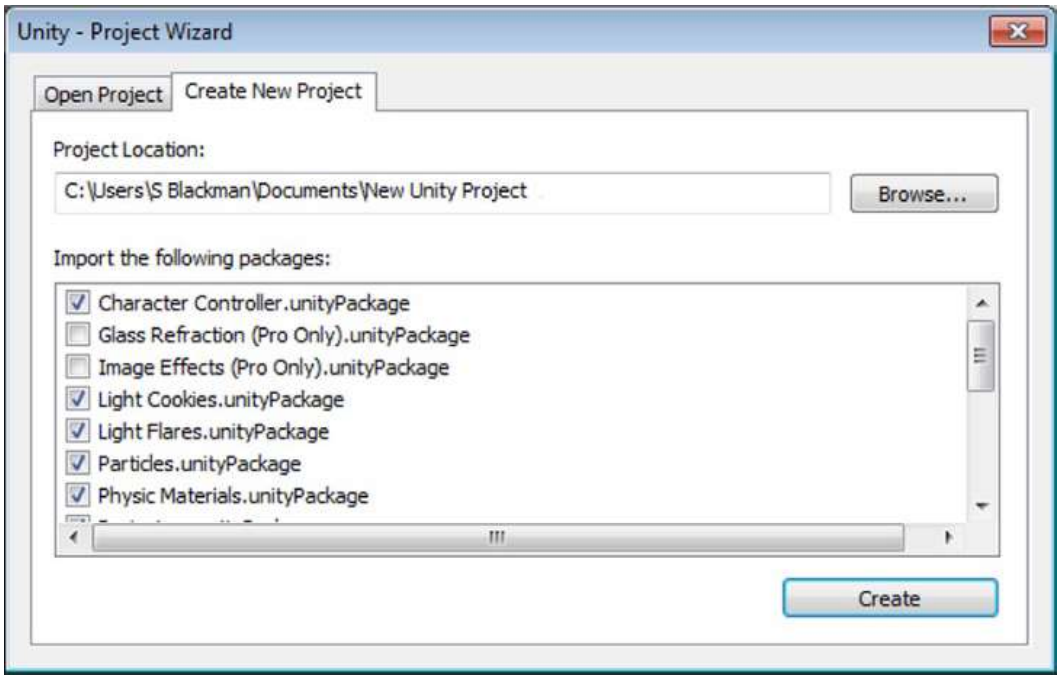

Figure 2-2. *Project Wizard Packages*

■ **Tip** If Unity is already open, choose New Project from the File menu.

---

4. Click Browse and navigate to where you'll keep your Unity projects, then create a New Folder for the book project.
5. Name the folder **BookProject**
6. Select Character Controller, Light Cookies, Light Flares, Particles, Physics Materials, Projectors, Scripts, Skyboxes, Terrains, and Water(Basic).

■ **Note** This book will make optional use of several Pro features, but the project will not rely on them to achieve a finished result. Should you decide to upgrade to Unity Pro or try the 30-day trial midway through the project, you can import the Pro packages easily, through the Assets menu, under Import Package.

7. Click Create.

Unity closes and reopens to show your new project—BookProject.

Unity Pro ships with the "dark" UI, as shown in Figure 2-3. As many readers of this book will be using the free version, screenshots will reflect the original, lighter version of the UI.

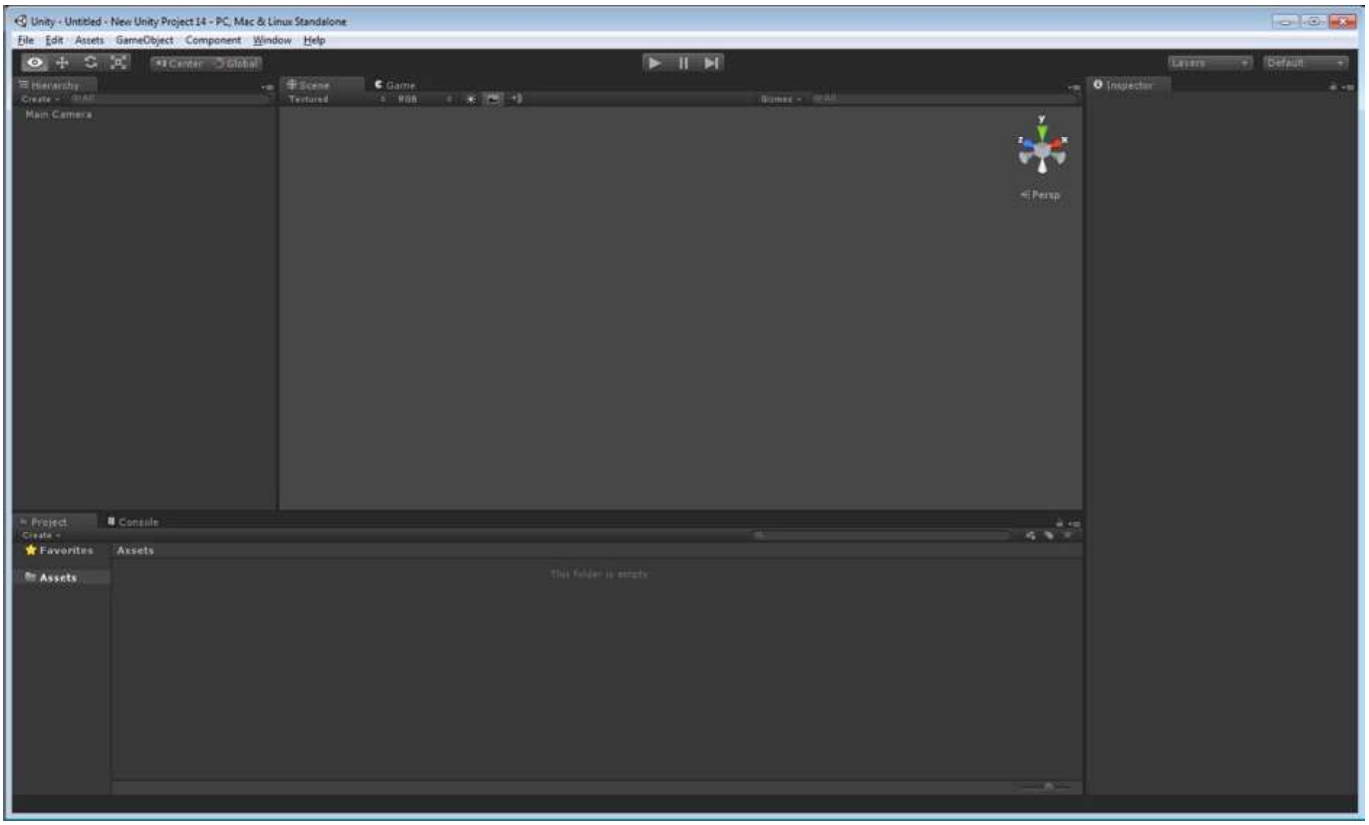

Figure 2-3. *The default Unity Pro "Dark" Skin*

If you have Pro, you may choose between the two skins through the Preferences dialogfrom the Edit menu, as shown in Figure 2-4.

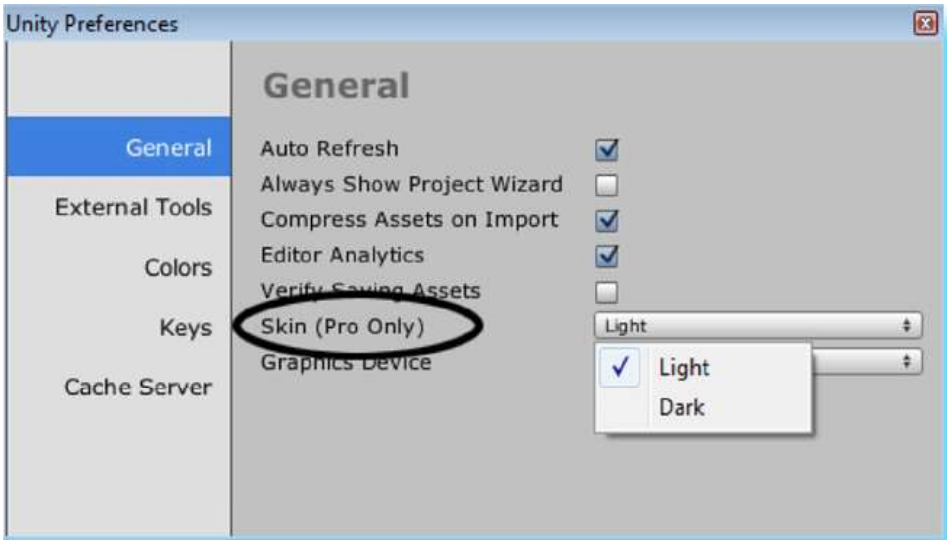

Figure 2-4. *The Unity Preferences dialog*

In your new scene, you will see a camera in the Hierarchy view and several items in the Project view (see Figure 2-5). When you create a new project, a folder called Assets is created in your machine's file system; its contents are displayed in the Project view. The top section contains filters to help you locate assets quickly, and in the lower section, you will see the Assets folder and its subfolders. In the right-hand column, you can see the contents of the selected folder.

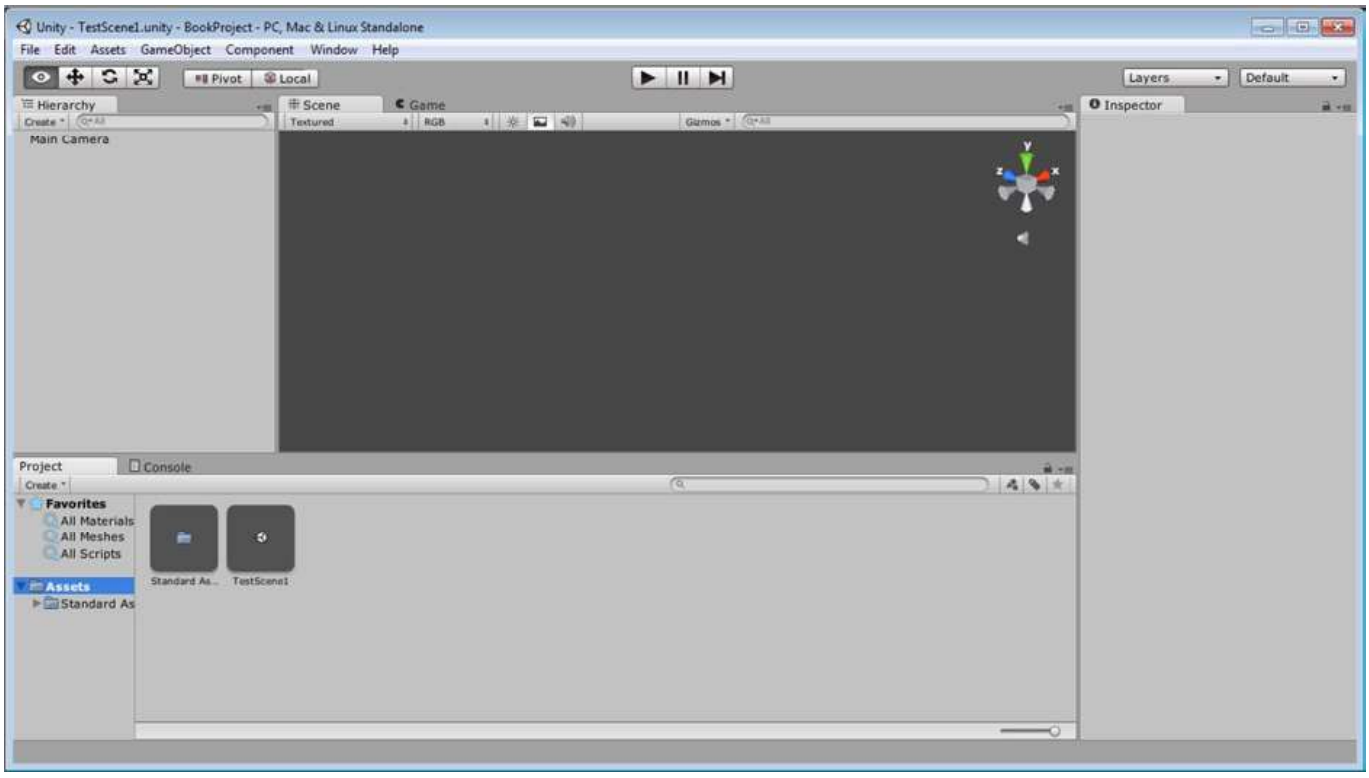

Figure 2-5. *The new project in the main editor window*

■ **Important!** Once an asset has been added to your scene, *never* rearrange or organize it using the OS, i.e., Explorer (Windows) or Finder (Mac), as that will break internal paths and metadata in the scene.

Before going any further, you will save the *scene*. Think of scenes as levels. A project can contain several scenes. Some may be actual levels, while others may be menus, splash screens, and so forth.

8. From the File menu, choose Save Scene, and save the scene as **TestScene1**.

As a default, the new scene is saved in the root Assets folder, one of the folders automatically created every time you start a new project. In Unity, it appears in the Project view with the Unity game icon next to its name in the One Column Layout, or in the second column if you are using the Two Columns Layout as shown in Figure 2-5.

You will eventually give the scenes a sub-folder of their own, but for now, the scene is easily visible. When you open a project, you will often have to select a scene before anything shows up in the Hierarchy view.

The Layout

There are many ways to rearrange the layout of the Unity UI. For this book, it is assumed that you will be using the standard 2×3 layout, as shown in Figure 2-6, because it allows you to have access to the Scene view when you are in Play mode. You can change the layout by selecting from the Layout drop-down in the upper right area of the application window, as shown in Figure 2-7. If you have the screen real estate, you may even want to tear off the various views into floating windows.

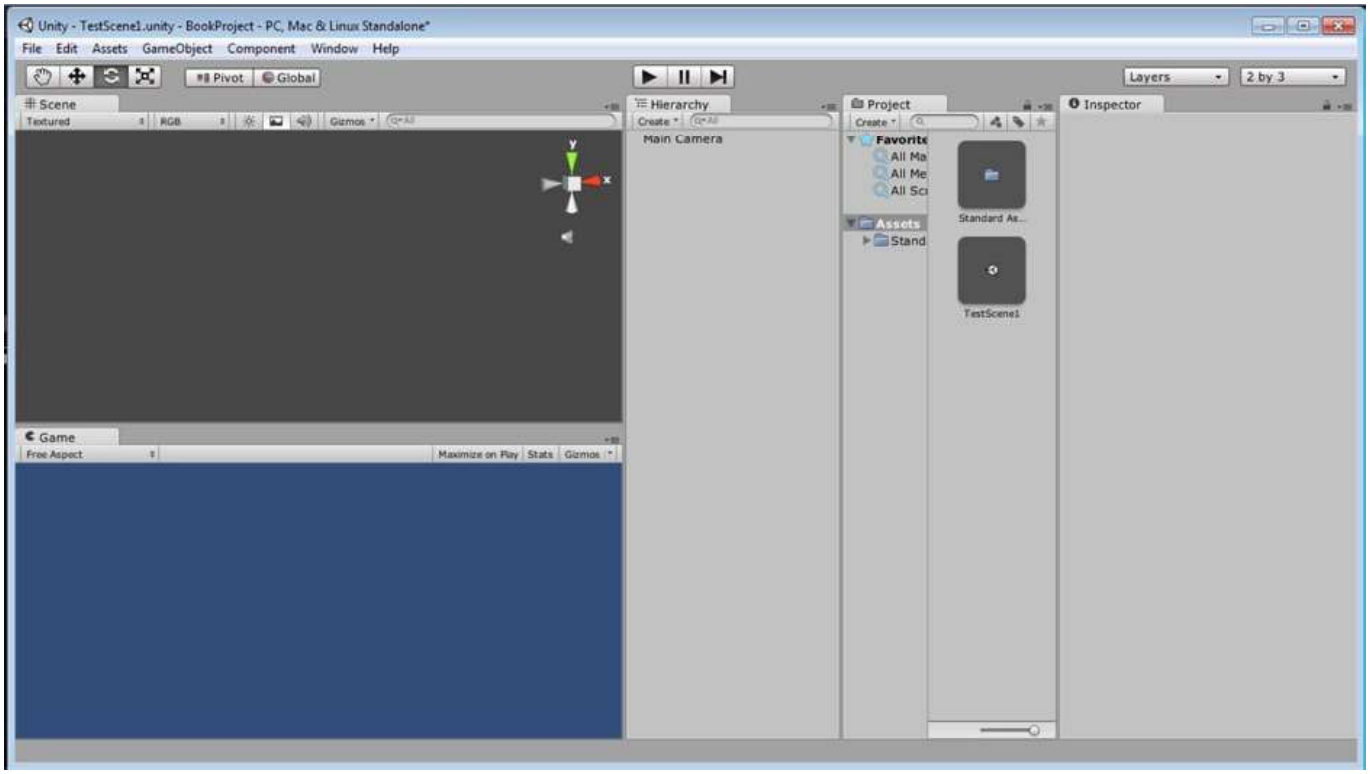

Figure 2-6. *Unity's UI—the 2×3 layout*

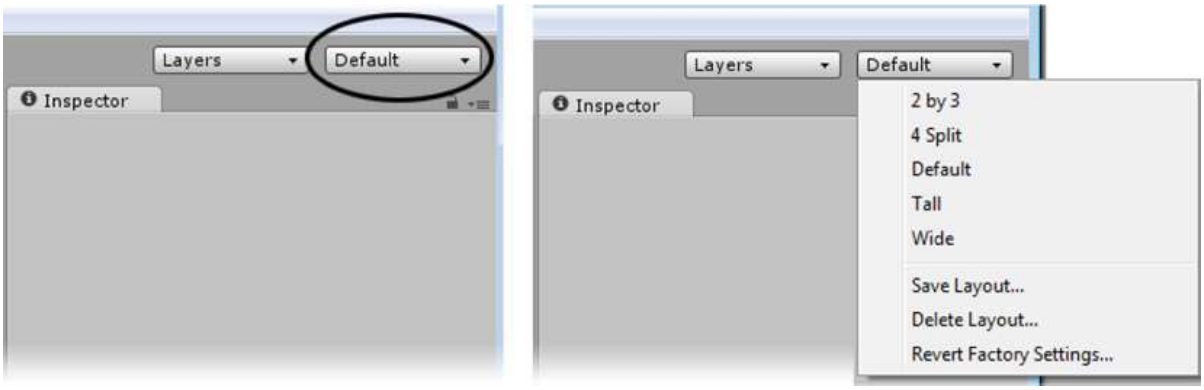

Figure 2-7. *Reconfiguring the UI using the Layout drop-down*

Unity 4.0 introduced a new split-column Project view. It's great for inspecting new assets in unfamiliar projects, but many people feel it introduces unnecessary clutter in their own projects. You are free to use whichever you prefer, but for the book, most instructions and screenshots will assume you are using the one-column Project view. You can select the Project-view layout of your choice by right-clicking over the Project tab and choosing either One Column Layout or Two Columns Layout, as in Figure 2-8.

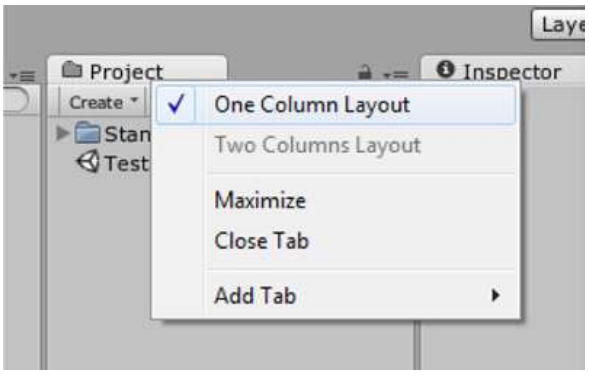

Figure 2-8. *Selecting the One Column Layout*

Scene View

The Scene view shown in Figure 2-9, is where you build the visual aspects of your scene—where you plan and execute your ideas.

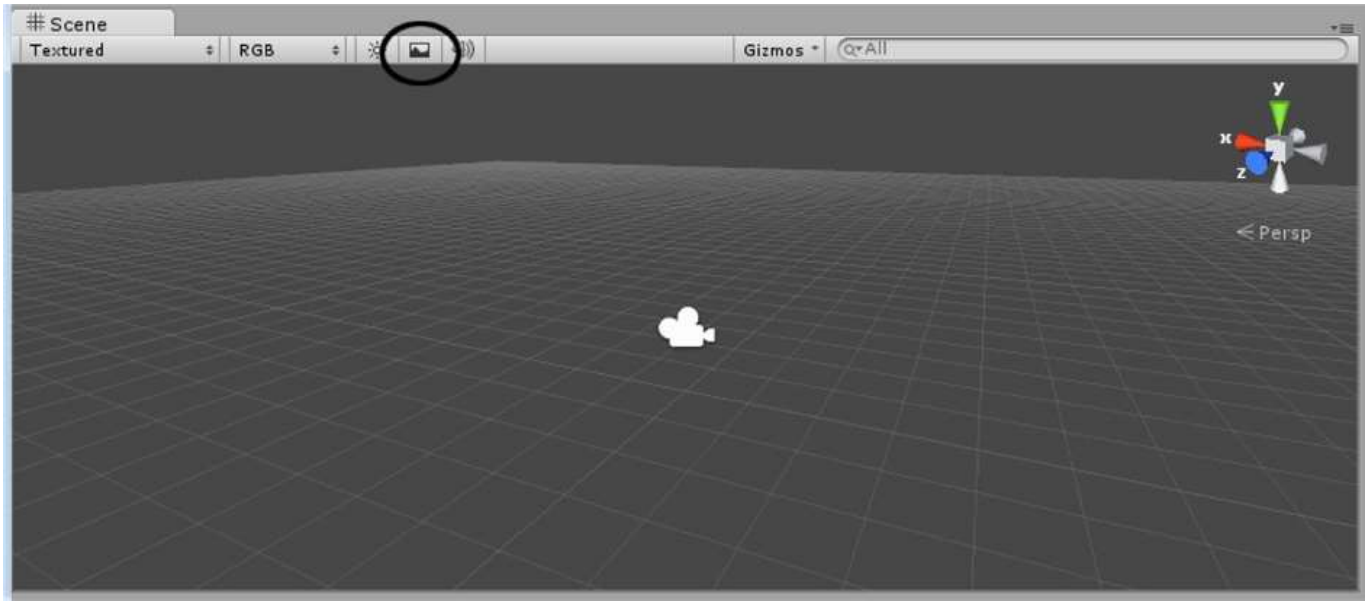

Figure 2-9. *The Scene view, showing the camera icon and the grid*

■ **Tip** To show the grid in the Scene view, toggle off the Game Overlay button (Figure 2-9).

Game Window

The Game window, shown in Figure 2-10, is where you test your game before building it in the runtime environment. You can't select objects in this view and, unlike the Scene view, it has no default lighting. You'll have to use the Main Camera to see objects in the Game window.

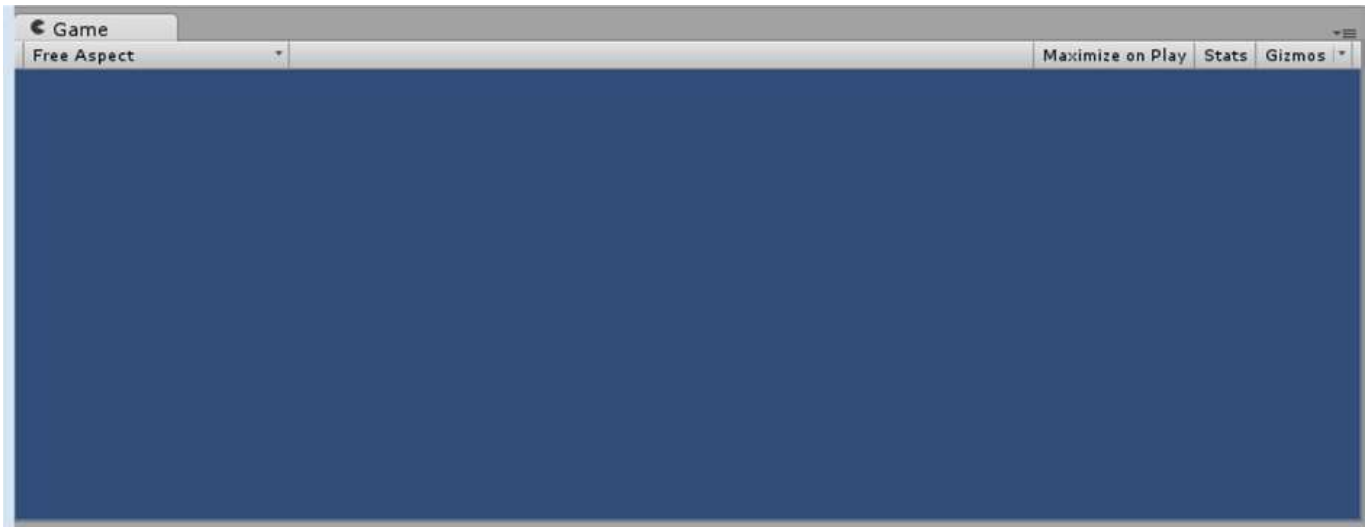

Figure 2-10. *The Game window*

Hierarchy View

The Hierarchy view (Figure 2-11) shows what's in the currently active scene. GameObjects that are dynamically added and removed from the scene during runtime will appear here when they are active in the scene.

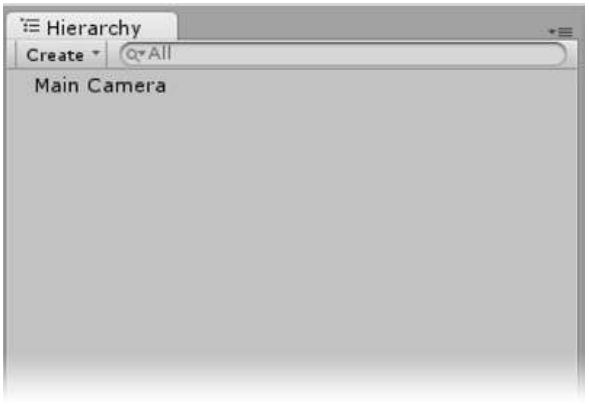

Figure 2-11. *The Hierarchy view, showing the only object in the scene, the Main Camera*

Project View

The Project view, shown in Figure 2-12, contains all the assets available to the current project, as well as all scenes or levels available for the finished game or application. It is a mirror of the Assets folder in the directory where the project resides. Removing assets from this location deletes them from the hard drive! Removing assets from the directory in Explorer removes them from the Project view and could break the scene.

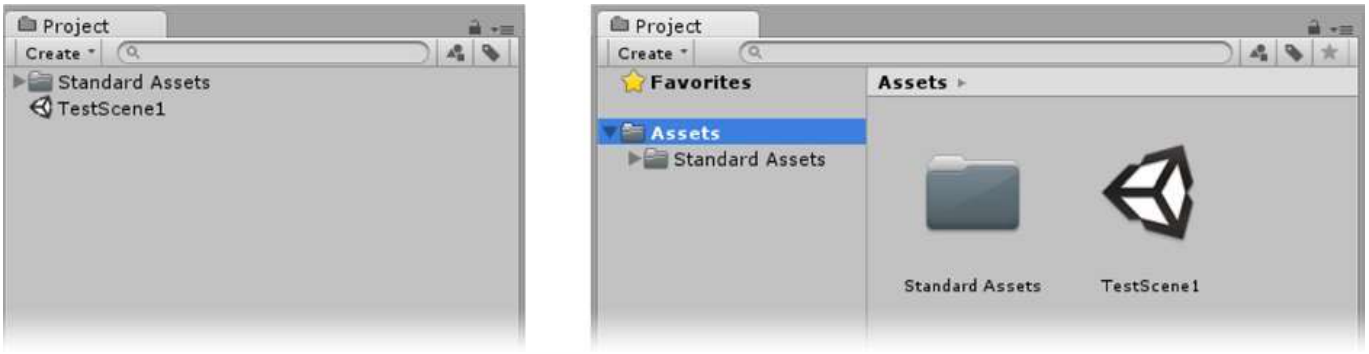

Figure 2-12. *The Project view , One Column Layout (left) vs. Two Columns Layout (right)*

Inspector

You use the Inspector to access various properties and components for objects you've selected in either the Hierarchy or Project views. You can also access other scene-related information here. Select the only object that's in the scene now, the Main Camera, from the Hierarchy view, and then take a look at the Inspector, as shown in Figure 2-13.

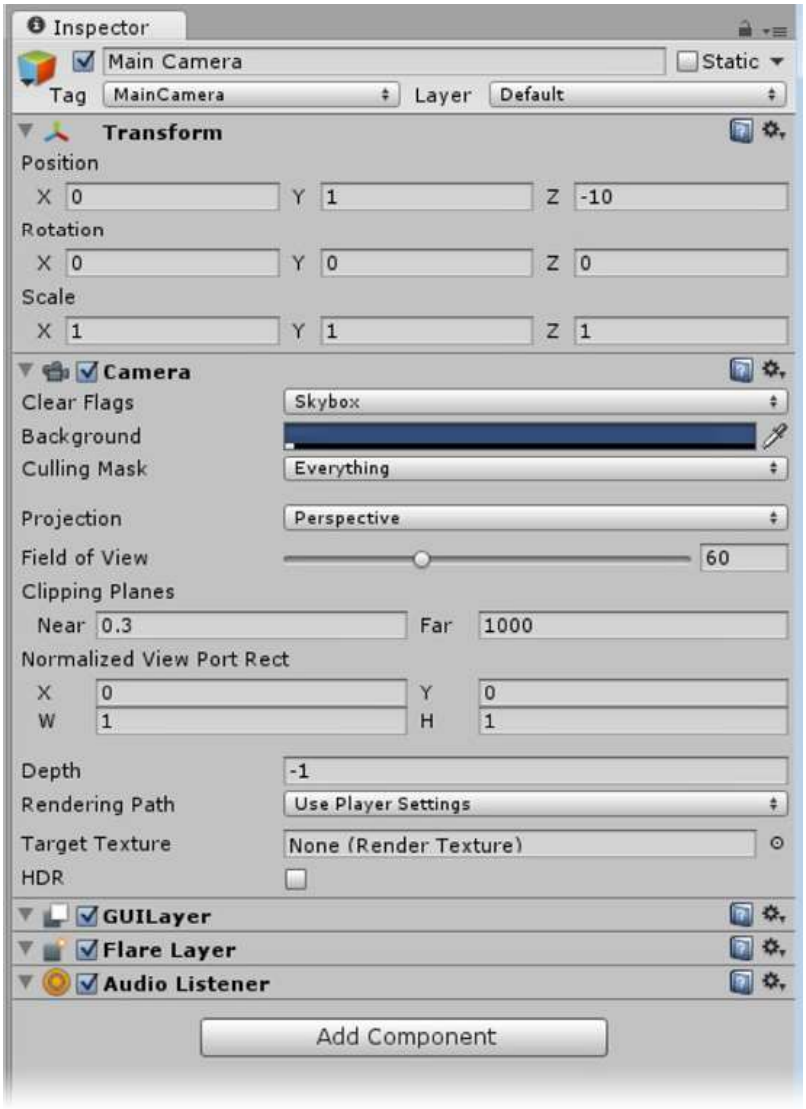

Figure 2-13. *The Inspector, with the Main Camera selected in the Hierarchy view*

Toolbar

Below the menu bar is the toolbar (see Figure 2-14), which contains five different controls.

Figure 2-14. *The toolbar*

The Transform tools, shown in Figure 2-15, provide functionality for navigating the scene and transforming objects. The button on the far left, the pan tool, can also become orbit and zoom tools for navigating and adjusting the Scene view. You can click and drag to move the view around (to pan). To orbit the current viewport center, hold the Alt key down,

while clicking and dragging. And to zoom, hold the Alt key (Windows) or the Cmd key (Mac), plus the right mouse button. There are other ways to perform these tasks, as you'll see when you add an object to the scene. But don't test the navigation until you have an object to work with.

Figure 2-15. *Scene navigation tools, pan, orbit, and zoom, (far left of each of the transform buttons)*

The remaining three buttons are for transforming objects in the scene in edit mode. The available transforms are move, rotate, and scale.

Objects can be transformed in different coordinate systems and from different pivot points. The next set of controls, to the right of the Navigation tools, shown in Figure 2-16, let you toggle between the choices.

Figure 2-16. *Pivot point and coordinate system tools*

The center controls, those shown in Figure 2-17, are the Play mode controls that allow you to see how the game will work in real time, as if you were the player.

Figure 2-17. *The Play mode controls*

To the right of the Play controls, you'll find the Layers drop-down (Figure 2-18). Layers are used in Unity for controlling which objects are rendered by which cameras or lit by which lights.

Figure 2-18. *The Layers control*

You already tried the Layout drop-down (Figure 2-19), when you set it to use the 2×3 split layout. You can rearrange and customize most of Unity's UI. The Layers drop-down lets you quickly get back to several default or predefined layouts.

Figure 2-19. *The Layout control*

Menus

Along with the usual file-handling options, there are many concepts and features that are specific to Unity. Most are found in the menus, as shown in Figure 2-20. Menus items also show the keyboard shortcuts for those who prefer them.

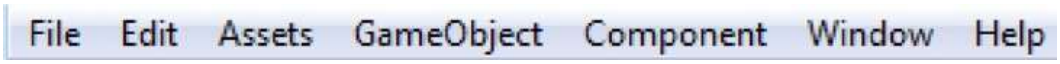

Figure 2-20. *The Unity Menu bar*

File

In the File menu, you can load, create, and save scenes and projects. It is also where you build your game as an executable or other deployment type.

Edit

The Edit menu (Figure 2-21) contains the expected Cut, Copy, Paste, Duplicate, and Delete commands, as well as several runtime commands. The actual editing of assets, however, is done in the Inspector. Of particular value here are the keyboard shortcuts shown next to the commands.

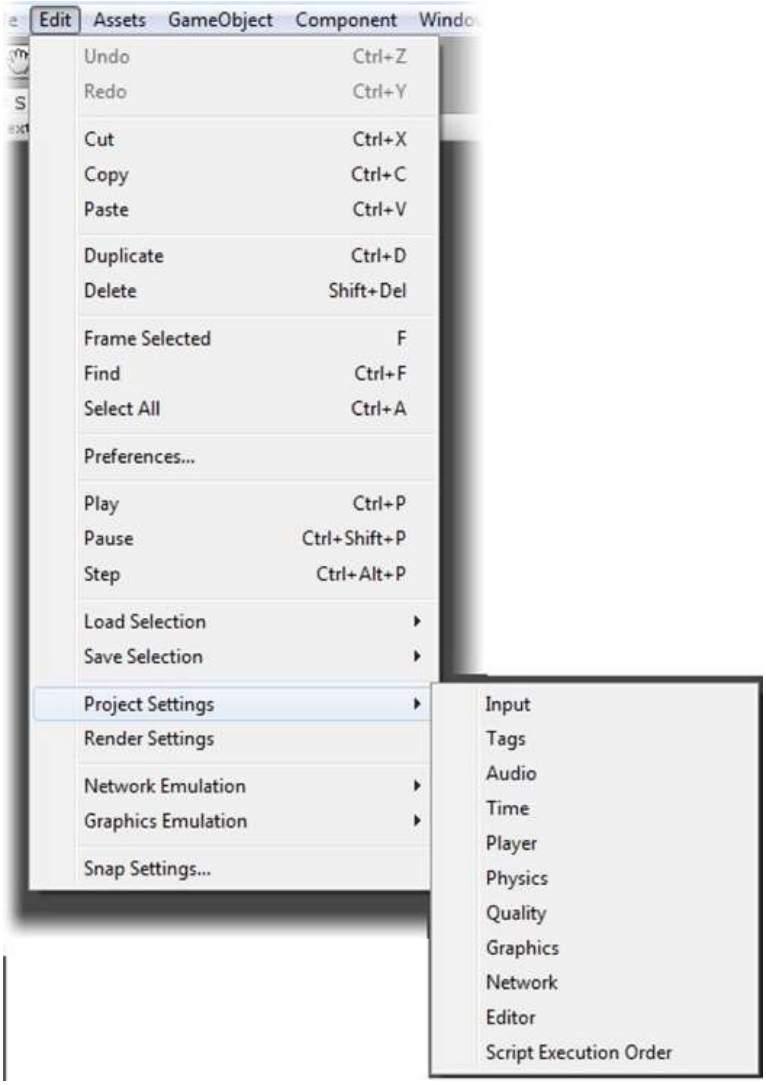

Figure 2-21. *Project Settings options accessed through the Edit menu*

There are several scene settings that are not accessed through the Hierarchy or Projects views. Besides the Render Settings, the Project Settings (Figure 2-21) are where you'll access many of your scenes' attributes. When you choose an option, the information will show up in the Inspector view.

Assets

In the Assets menu, you will find the Create submenu. This is where you can create the type of assets that generally can't be imported from DCC programs, such as non-mesh special effects, physics materials, and scripts. It is also the place to define custom fonts and organize the project with folders and prefabs. You can also reach this menu by right-clicking in the Project view.

GameObject

The GameObject menu lets you create an empty GameObject. The GameObject is the base container for objects, scripts, and other components in Unity. Like a group, it has its own transform and, so, is a logical means of organizing similar objects or meshes. At its simplest, a GameObject contains its transforms (location and orientation) and a few other properties. With the addition of various components, it can become almost anything you need. GameObjects are invaluable when used as a parent group or "folder" to keep multiple scene objects organized.

You can create various objects, such as lights, particle systems, GUI 2D objects, and Unity primitive objects from scratch with Components or find them ready-made in the Create Other submenu. Unlike many modeling programs, you cannot change the number of segments in Unity's primitive objects.

Component

The Component menu gives you access to items associated with objects, such as meshes, rendering types, sound, scripts, and colliders, as well as those that create or extend functionality with predefined behaviors or even editors. Components let you add physics characteristics to imported meshes, function-curve animation to objects, and all of the interactive intelligence that drives games and other, more serious applications via scripts. They are the building blocks that are added to the GameObject foundation to create almost everything in your scenes.

Window

As you'd guess, this menu displays the commands for managing various windows and editors in Unity, along with their keyboard shortcuts. It is also the way you must access Unity's Asset Store, a place where you can find or purchase assets of all sorts for your project.

Help

The Help menu provides access to the various Unity help manuals, as well as to the massively supported Unity Forum, Unity Answers, release notes, and instructions for reporting bugs.

Creating Simple Objects

It always helps to have something in the scene when you're going to experiment with navigation in a 3D application. Though you will import most of the assets for your games, you'll find many uses for the primitive objects available in the Unity game engine.

1. From the Create Other option in the GameObject menu, select Cube, as in Figure 2-22.

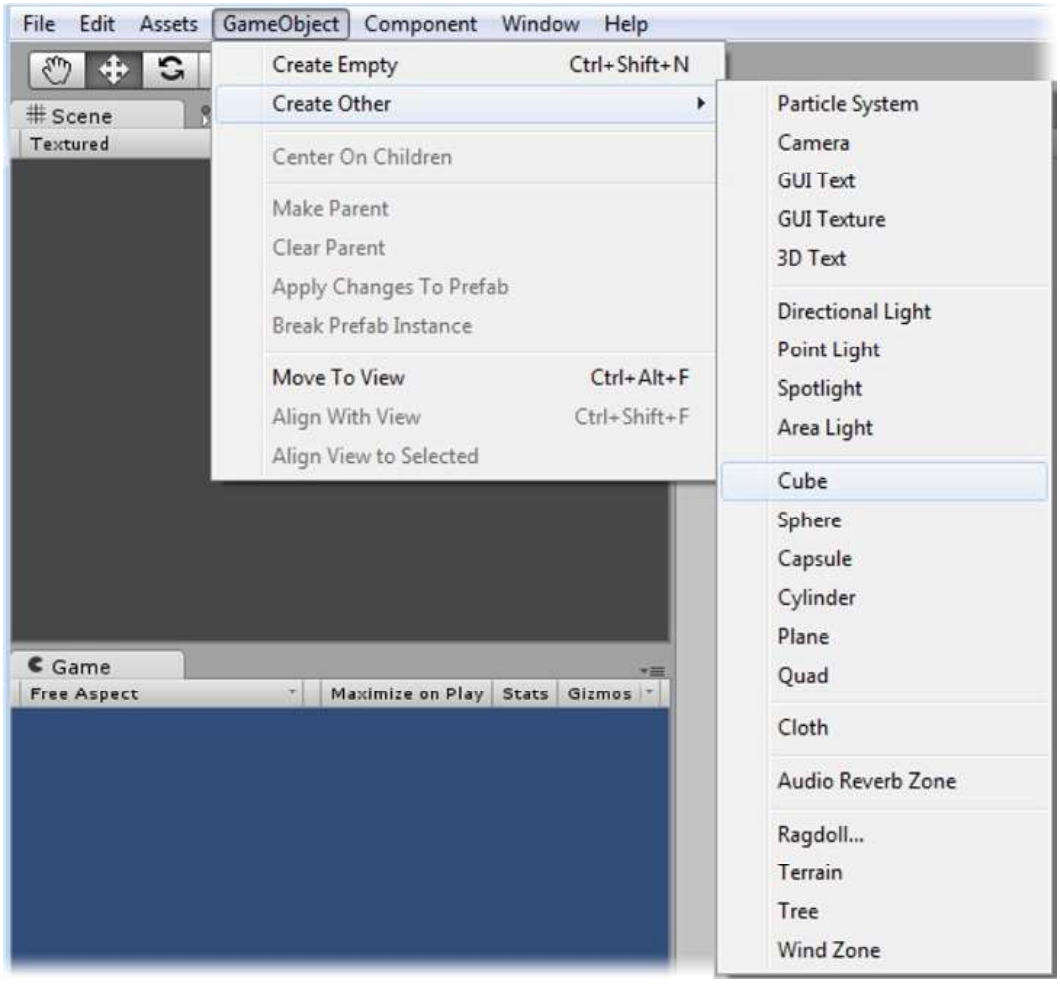

Figure 2-22. *Choosing the Cube in the GameObject ➤ Create Other menu*

A cube is created in the center of the Scene viewport. It may be little more than a speck at this point, if you've played with the viewport navigation tools. You may also see it as a small dark square in the Game window.

■ **Tip** If you have not navigated the viewport yet, the Move button will be active in the Navigation toolbar, and you will see the cube's transform gizmo.

---

2. To zoom, use the middle mouse roller or the Cmd button on the Mac. Zoom in and out on the new cube in the Scene view, as shown in Figure 2-23.

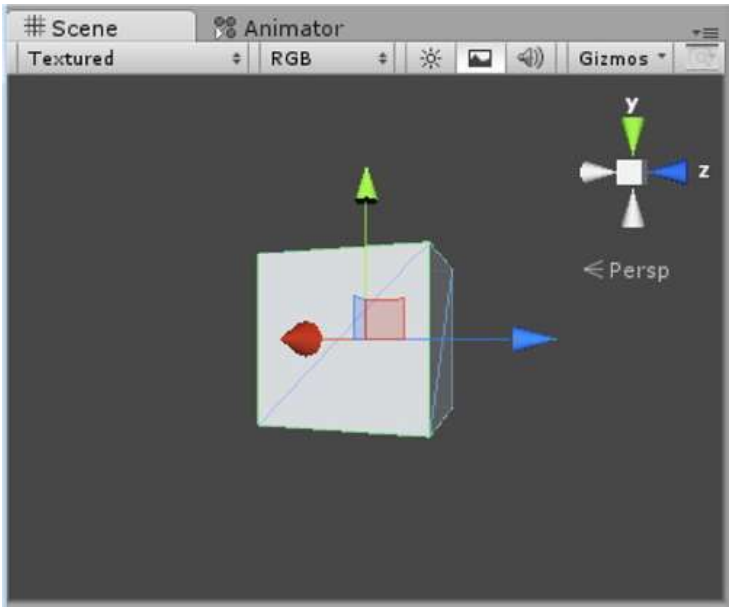

Figure 2-23. *The newly created cube*

To see the cube properly in the Game view, you have to move the camera. Rather than fussing with moving and rotating the camera, you can set the camera to match the Scene view, where you've already adjusted the preferred view of the cube.

3. Select the Main Camera in the Hierarchy view.
4. From the GameObject menu, choose Align with View.

The cube is now visible in the Game view and the camera Preview inset in the Scene view (see Figure 2-24).

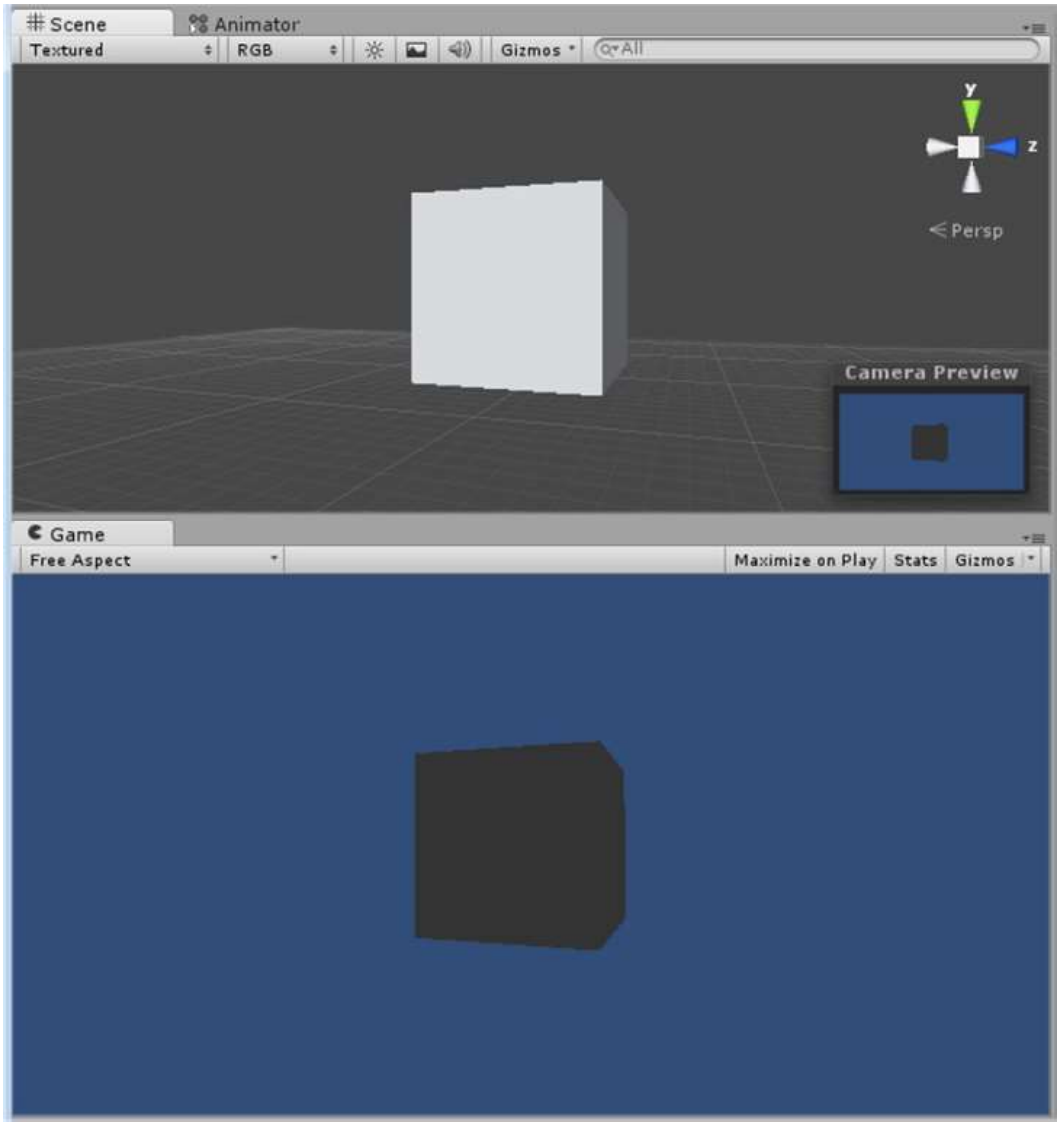

Figure 2-24. *The cube, as viewed by the camera in the Game window*

When the Light button is off, the Scene window is lit with default lighting, a single light pointing straight into the viewport. This assures that the object in focus is always well lit.

5. Toggle the built-in lighting off and the scene lighting on, by clicking the light button.

The cube is now lit only with the scene ambient light, as in the Game window (Figure 2-25).

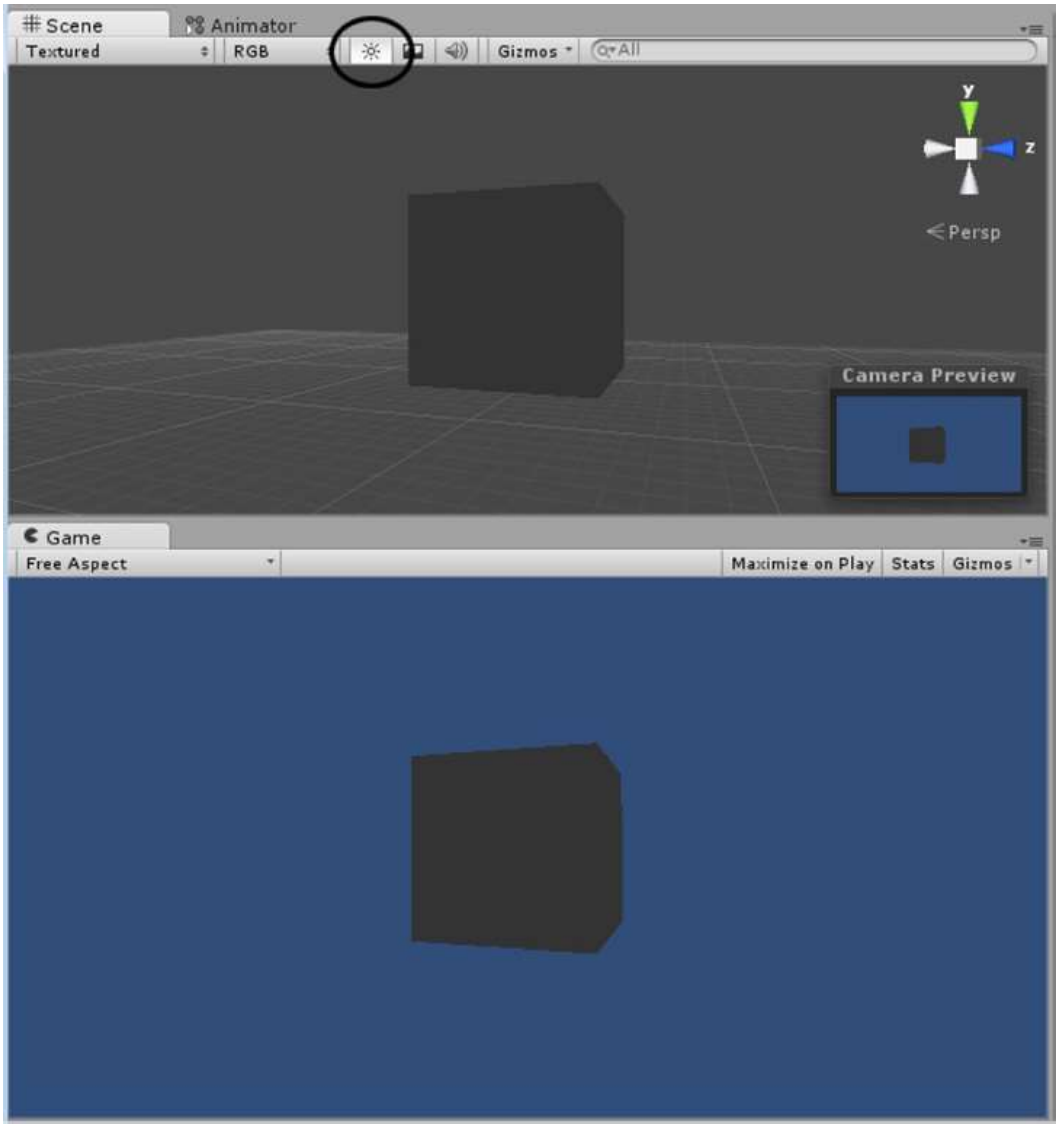

Figure 2-25. *Scene Light toggled on—there is no scene light yet*

Navigating the Scene viewport does not affect the camera. You can also zoom in and out by positioning the cursor in the viewport, holding the Alt key down, and holding the right mouse button down and moving the mouse back and forth. On a Mac, hold down the Cmd key while clicking and dragging to zoom.

Don't pan the viewport yet.

6. Toggle the scene lighting back off in the Scene view.
7. Slowly zoom out from the cube in the Scene view until a large camera icon blocks the cube.
8. Continue to zoom out, then zoom back in.

The icon changes size as you zoom in and out.

9. Click Gizmos above the Scene view and uncheck 3D to supress icon scaling (Figure 2-26).

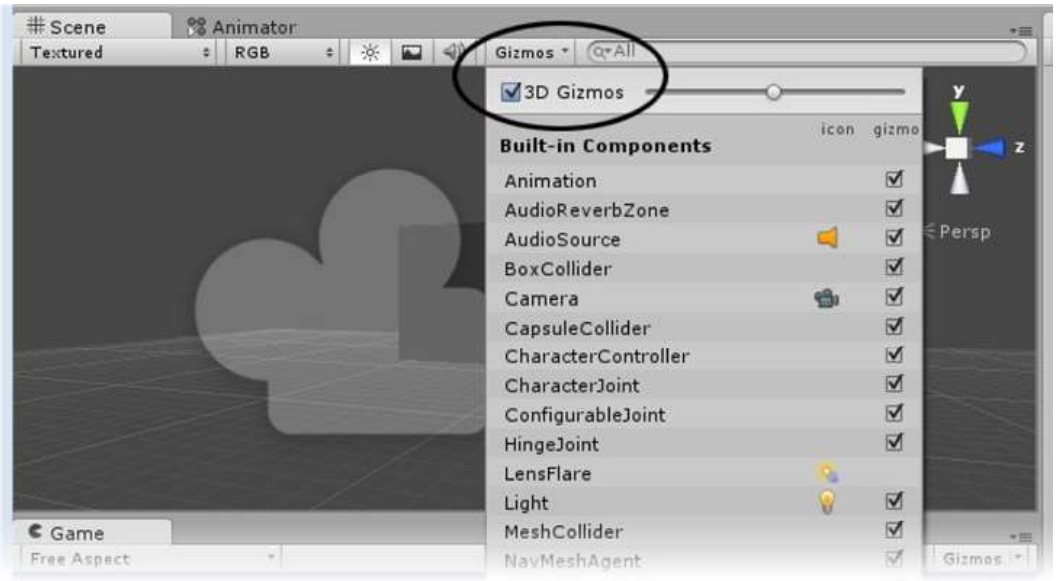

Figure 2-26. *Gizmo options for the Scene view*

10. Deselect the camera by clicking in a blank area of the Hierarchy view.

11. Toggle off the scene Light button.

12. To orbit the viewport around the cube, position the cursor in the viewport, hold down the Alt key and the left mouse button, and move the mouse around.

The view pivots around the center of the viewport's focal point. The cube itself is not rotated, as you can see by watching the cube in the Game view.

You can pan the viewport by clicking the Pan button and then holding the left mouse button down and dragging in the viewport. You can also position the cursor in the Scene window, hold the middle mouse roller down, and move the mouse.

13. Pan the viewport so the cube is no longer centered.

14. Use the Alt key again and orbit, to see how the view still orbits its current center point, *not the cube* (Figure 2-27).

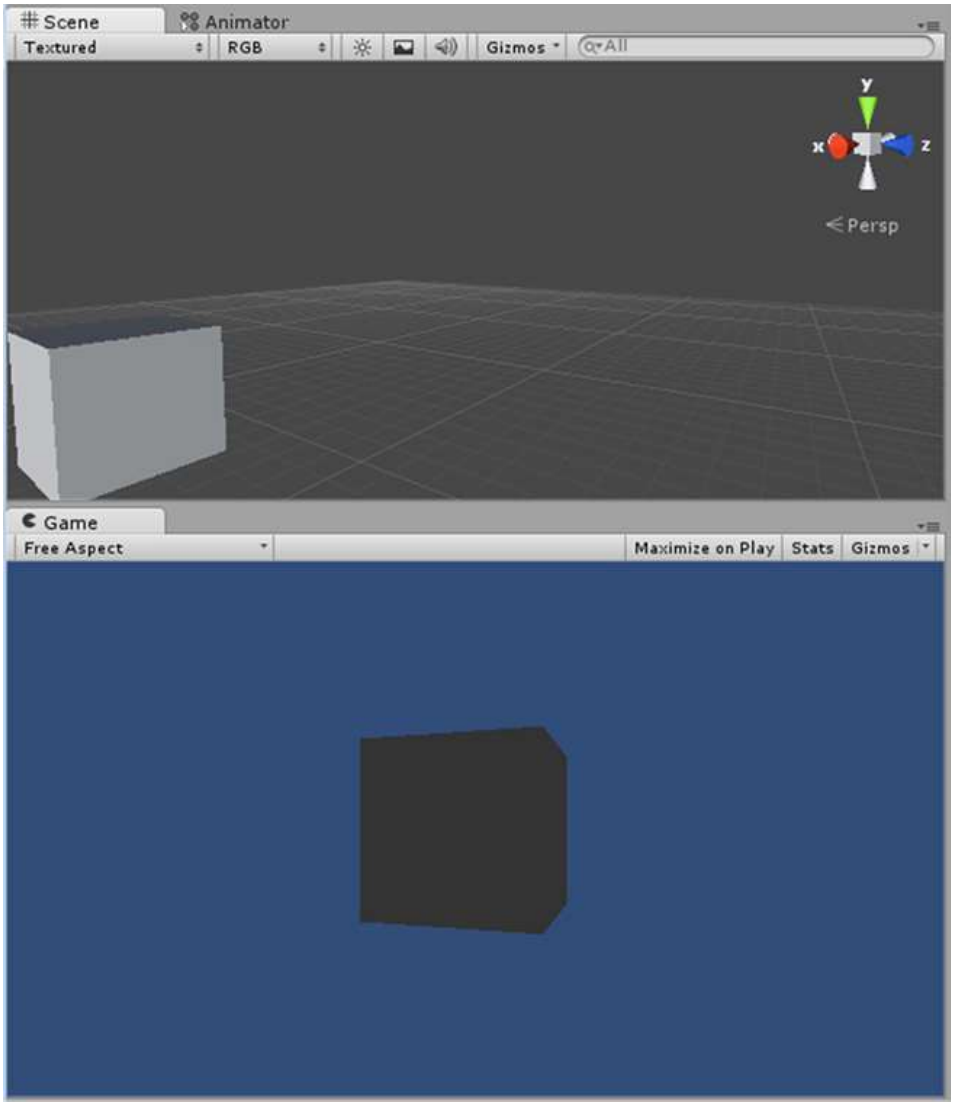

Figure 2-27. *The cube is no longer focused in the Scene view. Note that adjusting the view in the Scene view does not affect the camera's view, as seen in the Game window*

Selecting and Focusing

One of the most important navigation concepts in Unity is that of quickly zooming in to specific objects or areas.

To focus the viewport back to the cube, or to "find" the cube, do the following:

1. Make sure the cube is selected in the Hierarchy view.
2. Put the cursor *over the Scene window*.
3. Press the F key on your keyboard.

The view shifts to the center of the cube.

Also note that the edges of the selected object show in pale green. Using the View mode drop-down, shown in Figure 2-28, you can view scene objects in several other modes.

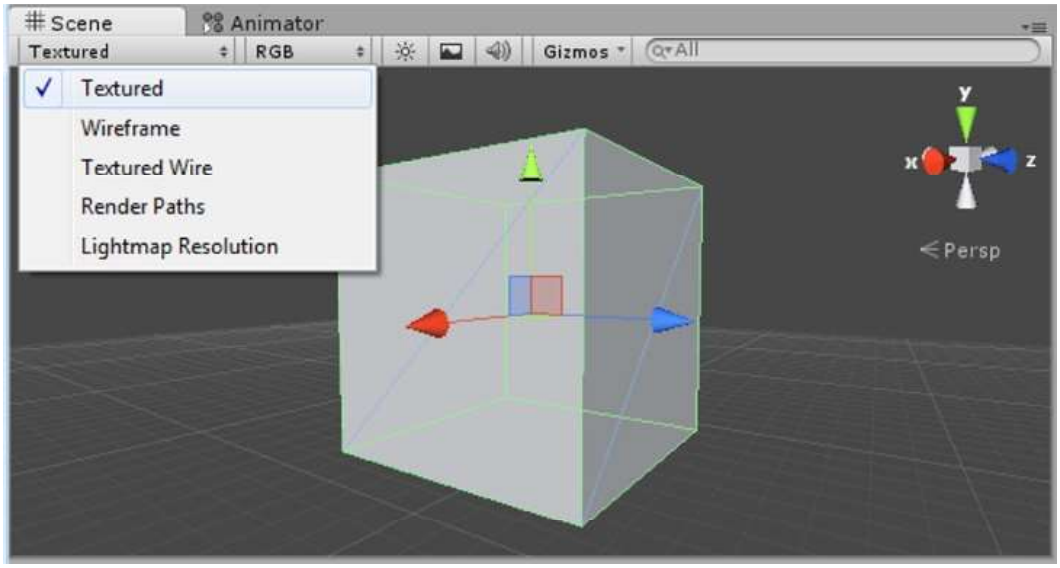

Figure 2-28. *The View mode drop-down*

4. Try the Wireframe mode.

The object shows only the object edges.

5. Try the Texured Wire.

The Textured Wire mode shows the solid object with the edges, but unlike the selected object in Texture mode, all of the scene objects will also be shown textured and edged.

6. Return the view mode to Textured.

Transforming Objects

In 3D space, you use what is called the Cartesian coordinate system. If you come from Autodesk products, you will be used to Z representing "up" in the world. This derived from traditional drafting, where the paper coordinates on the drafting table were X and Y, so Z became up. In Unity, Y is up, harking back to the early days of 2D monitor space, where X is horizontal and Y is vertical. With the advent of 3D space on computers, Z was assigned as back or forward from the monitor. Z depth is an important concept in real-time applications, as it is used to determine the order objects are drawn into the scene.

No matter which way is up, the directions are color-coded, with red, green, blue (RGB) corresponding to X, Y, Z. Objects imported from applications using Z up will retain their original orientation, regardless of coordinate system.

Objects can be transformed (moved, rotated, scaled) around the scene with the rest of the tools on the upper left toolbar (Figure 2-29).

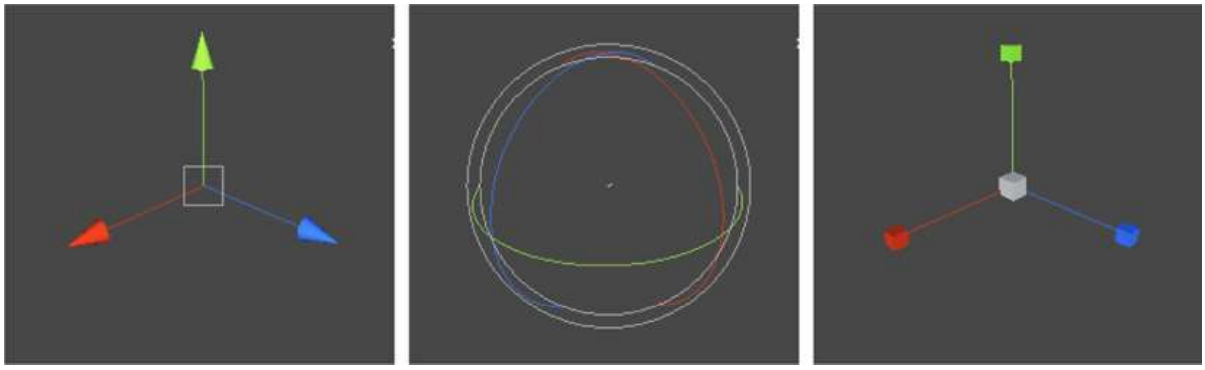

Figure 2-29. *Translate(Move), Rotate, and Scale Tool gizmos*

1. Select the cube from the Hierarchy view or by picking it in the Scene view.
2. Select the Move button.

A Transform axis appears for the cube.

3. Click and drag on any of the three axes.

The active direction turns yellow, and the movement is confined to that arrow's direction. The object also moves in the Game window, as Figure 2-30 shows.

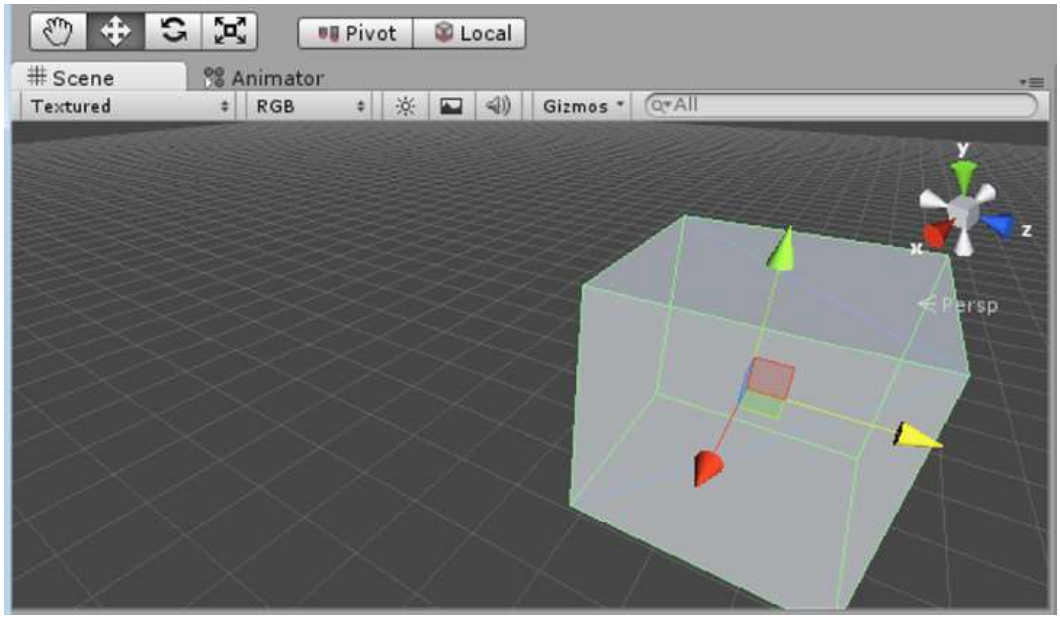

Figure 2-30. *The translated (moved) cube. Note that the camera has not moved, but the cube has, as is reflected in the Game window*

4. Next choose the Rotate icon.

The gizmo consists of circles, once again color-coded.

5. Click and drag the red, green, and blue circles to see what happens.

The gray outer circle will rotate the object to the current 2D screen coordinates, no matter which way you orbit the view.

In the Inspector, you can set the object's transforms manually (see Figure 2-31).

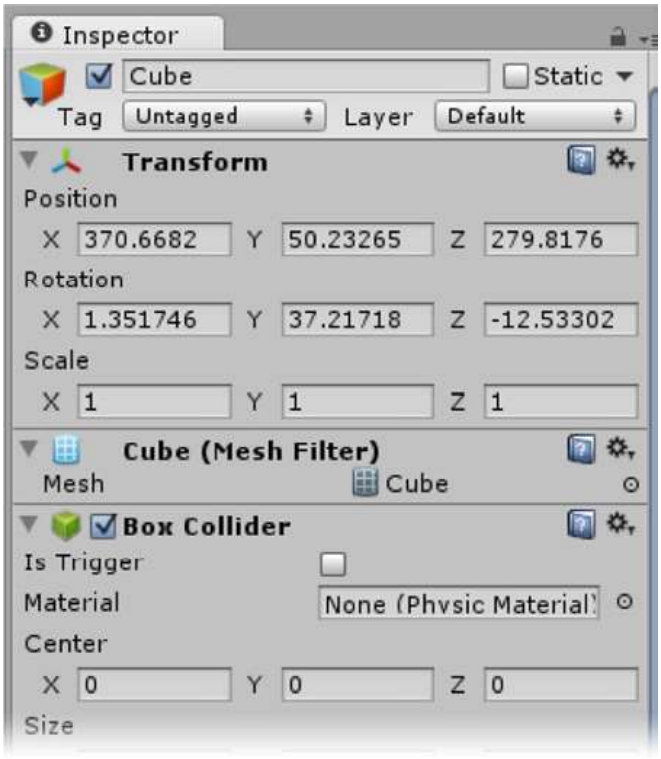

Figure 2-31. *Performing transforms on the cube, as shown in the Inspector*

6. In the Transform section for the cube, set the Y Position to **0**, the X Position to **20**, and the Z Position to **10**.
7. Use the F key to focus the viewport to the cube's new position, but remember to position the cursor in the Scene window first.

■ **Tip** In Unity, a single unit is generally considered to be 1 meter. As with most 3D apps, though, scale is really quite arbitrary. You will, however, find that elements such as lights and physics typically have default settings that will require quite a bit of tweaking if you stray too far from the norm.

The transforms are all floating point numbers; that is, they can have fractions, as indicated by the decimal points.

You can also adjust the values by positioning the cursor over the axis label, pressing the left mouse button, and dragging.

8. Position the mouse over the Y Scale label and drag.

The cube is scaled on its Local axis, and the values change accordingly in the Inspector.

The small center gray rectangle allows you to adjust more than one axis at the same time. With the Scale gizmo, it will cause the object to be uniformly scaled on all three axes.

9. Select the Scale tool.
10. Test the uniform scale by clicking and dragging within the small gray cube at the center of the Scale gizmo.
11. Set the X, Y, and Z Rotations and Positions to **0**.
12. Set the Scale X, Y, and Z to **1**.
13. Use the F key to focus the viewport again.

Snaps

Besides manually moving objects in the Scene view or adjusting their transforms in the Inspector, you can snap objects by incremental amounts, or to each other, by vertex.

1. From the bottom of the Edit menu, open the Snap settings dialog, shown in Figure 2-32.

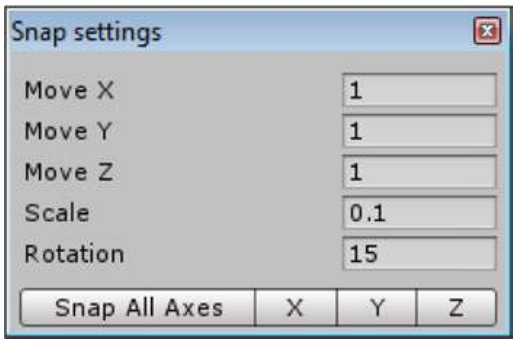

Figure 2-32. *The Snap settings dialog*

Note that Move X, Move Y, and Move Z are all set to 1, and the Rotation snap is set to 15 degrees. To use the increment snap, hold down the Ctrl (Windows) or Cmd (Mac) key while transforming the object.

2. Select the cube.
3. Make sure the Transform tool is active.
4. Hold down the Ctrl or Cmd key and move the cube slowly in the Scene window.

Ever so subtly, the cube moves in one-unit increments, as you can see by watching the Inspector.

5. Zoom out a bit, using either Alt + the right mouse button or the mouse roller, if you have one.
6. In the Snap settings dialog, change the Move amounts from 1 to 5.
7. With the Ctrl or Cmd key still down, move the cube again.
8. Set the Move amounts back to **1**.

This time, the snap is more noticeable.

Rotation snaps can be very useful when changing an object's orientation. Rotation snaps work the same way as the position snaps.

1. Select the cube.
2. Make sure the Rotation tool is active.
3. Hold down the Ctrl or Cmd key and rotate the cube slowly in the Scene window.

The cube snaps in 15-degree increments.

4. Close the Snap settings dialog.

Vertex Snaps

As well as the increment snaps, you can also snap objects using their vertices as registration points. While this may not be terribly useful in an organic environment, it can be invaluable for snapping objects such as platforms, walls, roads, and tracks together.

1. Activate the Move tool.
2. Select the cube.
3. Use Ctrl+D to duplicate the cube.
4. Select one of the cubes and move it away from the other.
5. Hold down the V key (for Vertex) and slowly move the cursor over the repositioned cube.

The transform gizmo snaps to the cube's vertices as you move the cursor, as shown in Figure 2-33.

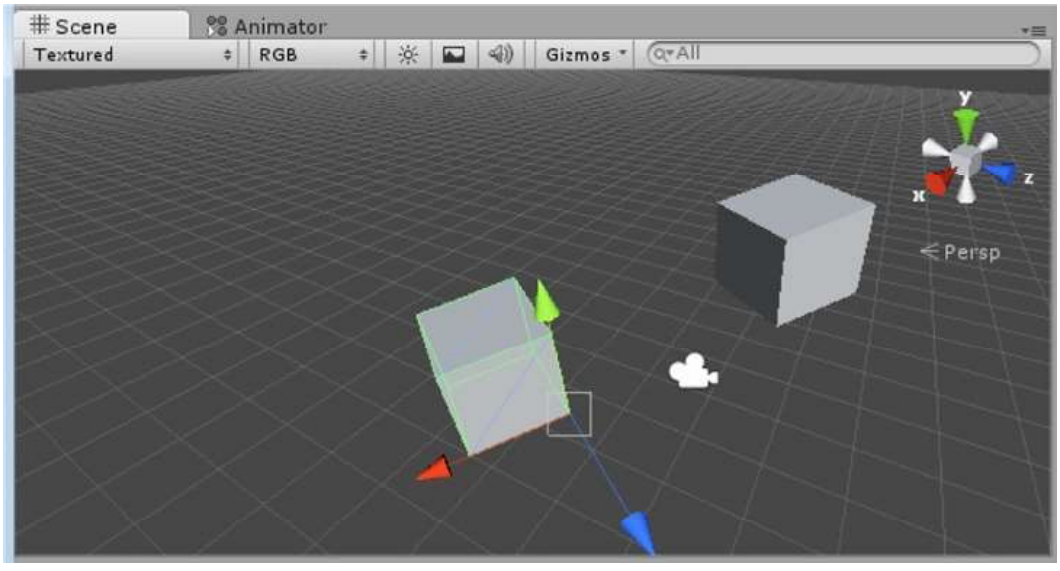

Figure 2-33. *The transform gizmo, snapped to one of the cube's vertices*

6. Continuing to hold the V key down, press and hold the left mouse button and move the cube toward the other cube, until it snaps to the desired vertex on the other cube.
7. Delete the extra cube by using the Delete key on the keyboard or right-clicking the cube in the Hierarchy view and selecting Delete from the menu.

For more information on positioning objects with the snaps, see "Positioning GameObjects" in the manual.

Scene Gizmo

So far, you have adjusted the viewport and transformed the object in a Perspective viewport. While this is useful when you know the exact location, or don't need to be exact at all, it can be challenging and slow to continually rotate the view and adjust the object with more accuracy. The Scene Gizmo icon, shown in Figure 2-34, lets you switch between Perspective and Iso views by clicking on the view label. You can also quickly change the viewing direction by clicking on the gizmo's cones. The cones point in the viewing direction. Orthographic views, Front, Back, Top, Bottom, when using Iso projection, allow you to judge position and scalar relationships without the distortion of perspective.

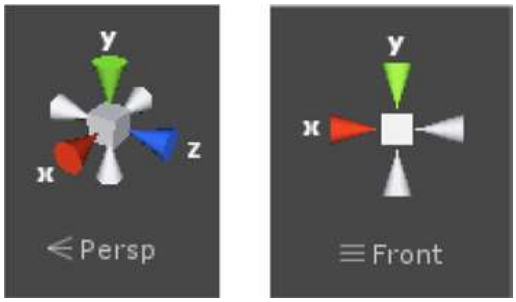

Figure 2-34. *Scene Gizmo, showing a perspective view (left) and an iso Front view (right)*

1. Click the Y cone on the Scene Gizmo icon.

You are now in a Top viewport. At this point, the screen-space transform gizmo makes more sense.

2. Select the cube and activate the Rotate tool.
3. Rotate the cube about 20 or 30 degrees with the outer circle, until it is no longer in an orthographic orientation.
4. Activate the Move tool.
5. Toggle the coordinate system from Global to Local (see Figure 2-35).

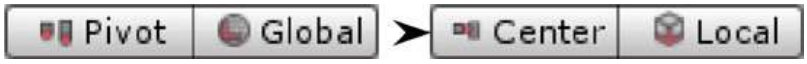

Figure 2-35. *Global to Local Coordinate system and object pivot to object center toggles*

The local coordinate system allows you to transform the object relative to its own local coordinates, as opposed to the fixed scene or World coordinates. You may also wish to use the object Center instead of the object's creation Pivot.

6. Now, move the box in its local X direction.
7. Set the cube's rotations back to **0** in the Inspector.

You may have noticed a small gray square in the middle of the Move gizmo. It allows two-axis movement and is very useful in orthographic-iso views.

Non-Snap Alignment

To align an object with another, you can once again use the options in the GameObject menu.

1. Click the gray center box of the Scene Gizmo to get back to a Perspective viewport.
2. Select the cube and use the F key to focus or find it.
3. From the GameObject menu, choose Create Other ➤ Create a Sphere.
4. Activate the Scale tool.
5. Scale the sphere using the center yellow cube on the gizmo to uniformly scale it, so it shows through the cube (see Figure 2-36).

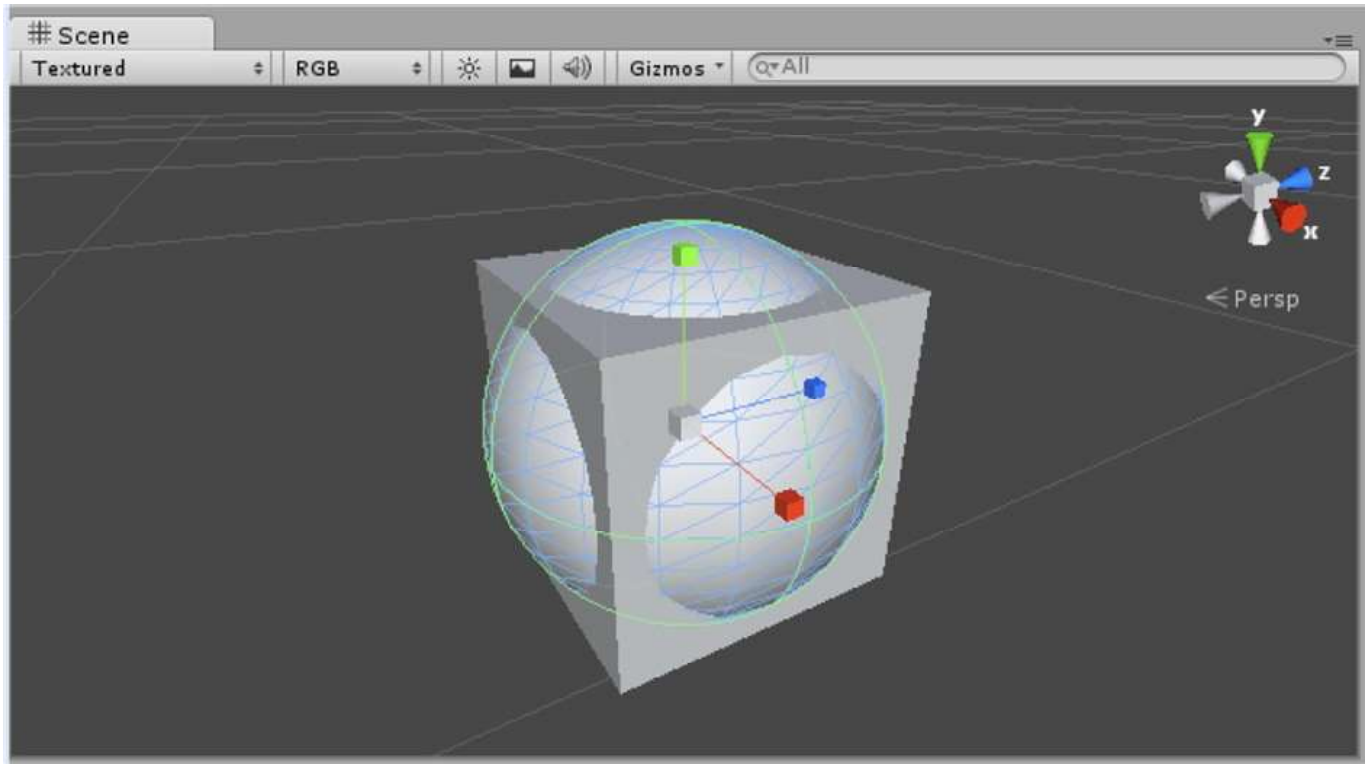

Figure 2-36. *The sphere is created in the same position as the cube*

The sphere is created in the same spot as the cube.
To align existing objects, you can use the third alignment option from the GameObject menu.

1. Select the Move tool.
2. Move the sphere away from the cube.
3. Select the cube and focus the view to it.
4. Select the sphere.
5. From the GameObjects menu, select Move to View.

The sphere is positioned at the cube once again, because the cube was focused to the center of the view.

6. Delete the sphere by selecting it in the Hierarchy view, then pressing the Delete key, or by selecting Delete from the Edit menu, or by pressing Shift+Delete.

Lights

In the Game window, the cube is dark, illumined only by ambient scene lighting, but in the Scene window, it is lit. When you pressed the Lighting button at the top of the window, it toggled the built-in lighting off, so you could see the scene lighting. Because there are no lights in the viewport yet, it went dark, just as in the Game window.

1. Focus the Scene view on the cube again and zoom out a bit.
2. From the GameObject menu, choose Create Other ➤ Create a Directional Light.

Directional lights emit parallel rays, no matter where they are in the scene.

3. Move the light up and out of the way of the cube.
4. Rotate it in the Scene window until the cube is lit nicely in the Game window, as shown in Figure 2-37.

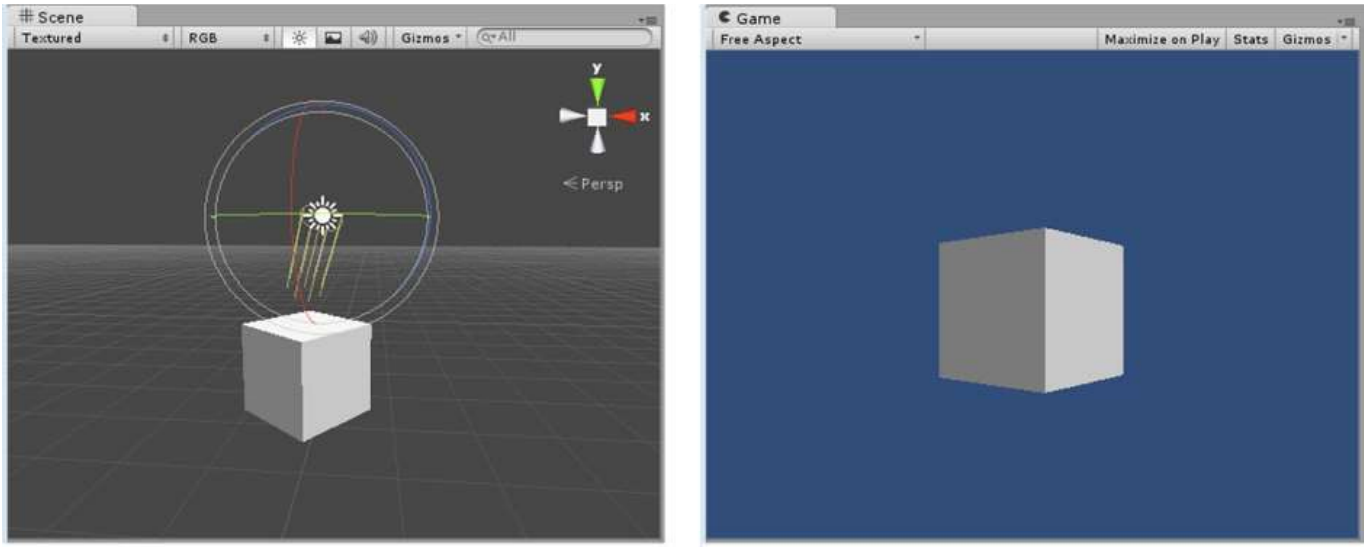

Figure 2-37. *The Directional light oriented to light the cube*

5. Using Alt + the left mouse button, drag/orbit the viewport so that you are facing the unlit side of the cube, as shown in Figure 2-38, left.

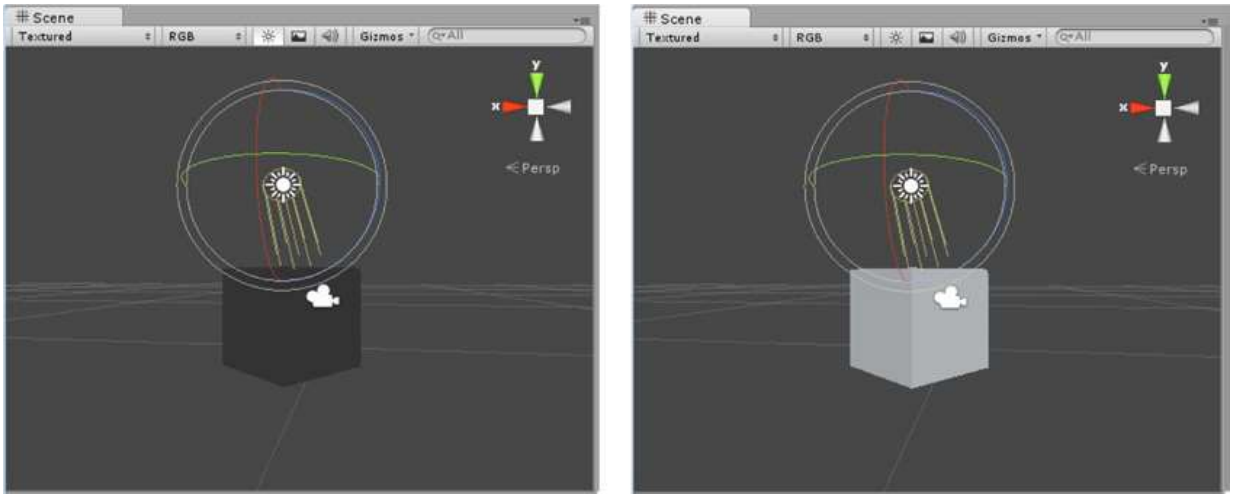

Figure 2-38. *The unlit side of the cube (left) and the Light button toggled off (right)*

The details disappear.

6. Toggle the scene lighting back off with the light icon (Figure 2-38, right).

The default lighting shines directly into the viewport, ensuring objects are easy to see, regardless of viewpoint in the Scene window.

As you create your scene, you will probably toggle the Scene lighting off and on regularly.

Now is a good time to take a peek at the light in the Inspector (see Figure 2-39). The light object is simply a gameObject with a Light component.

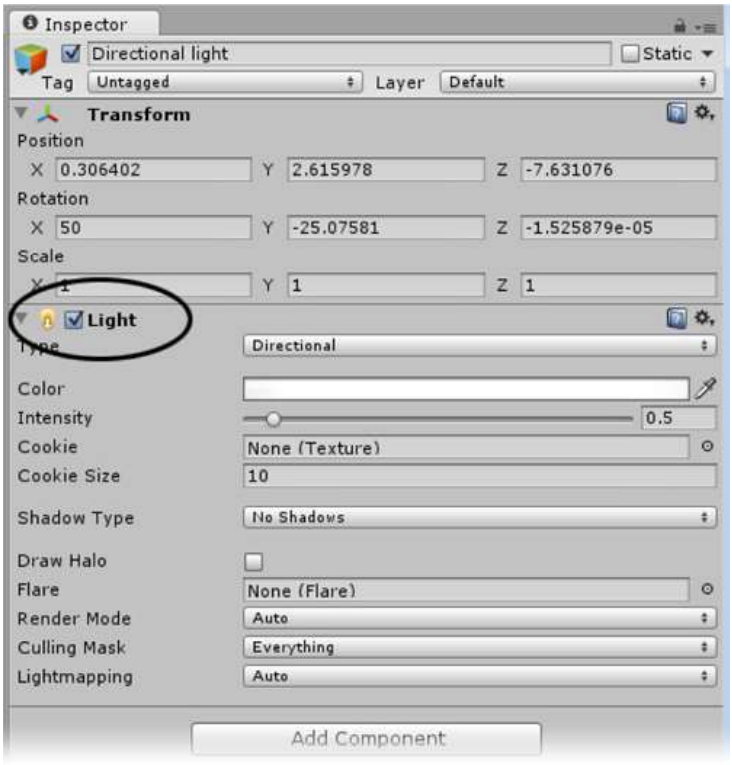

Figure 2-39. *Light component selected in the Inspector*

7. With the light selected, uncheck then check the box next to the Light component's label (Figure 2-39).

The light turns off then on.

8. Click on the color swatch next to the Color label.

The floating Color dialog opens, as shown in Figure 2-40.

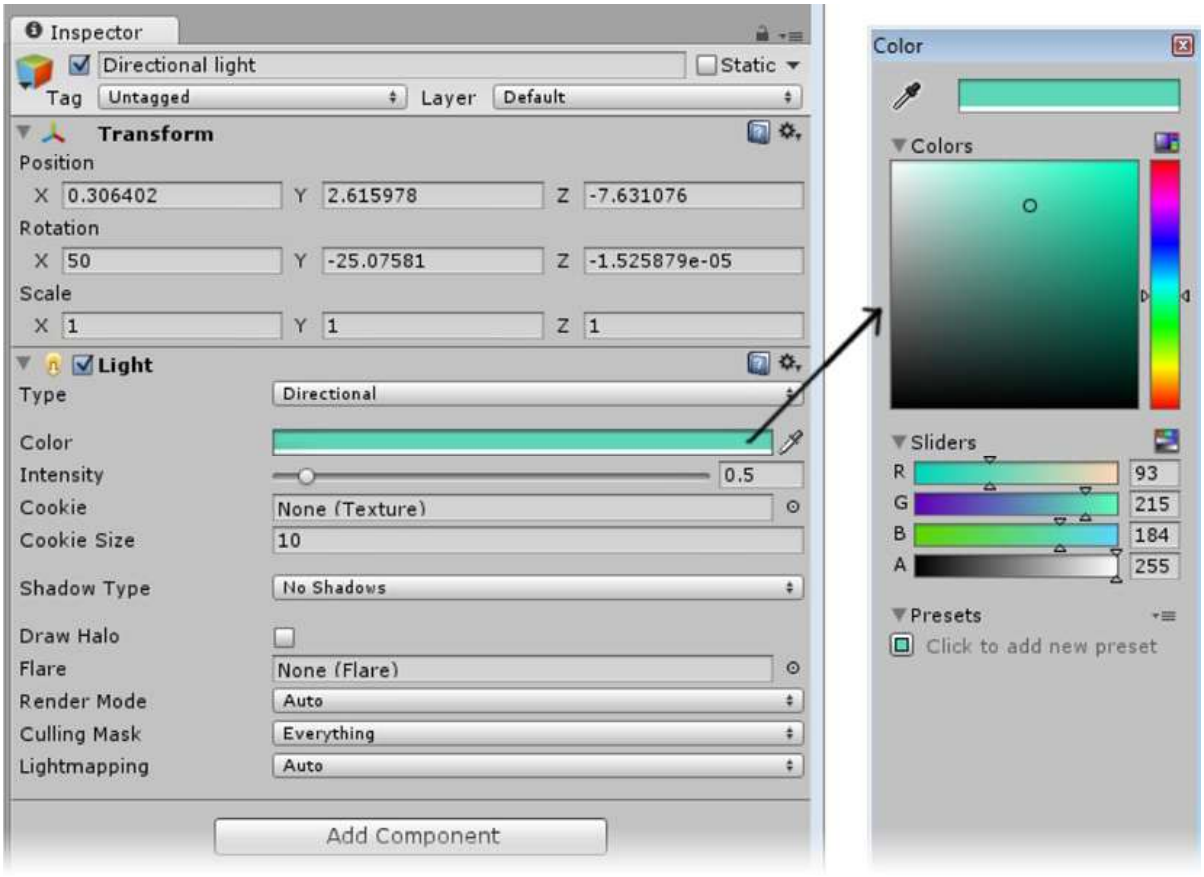

Figure 2-40. *The Color dialog, accesed through the Light's Color parameter*

9. Click and drag in the Color's image/color swatch and watch the light's color on the cube change.
10. Choose something subtle.
11. Close the dialog.

Next, you will add a new material to the cube to replace the default.

1. In the Inspector, select the Cube.
2. In the Mesh Renderer section, click to open the Materials array list (see Figure 2-41).

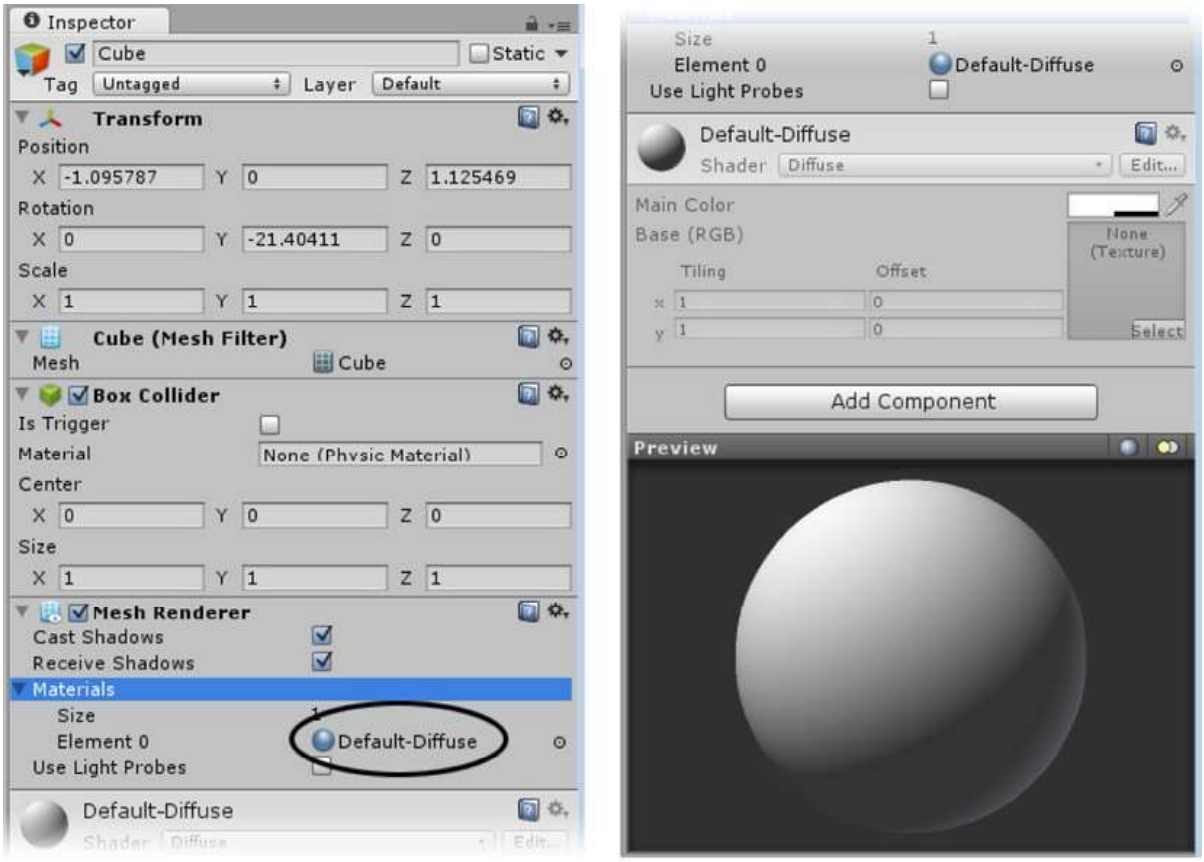

Figure 2-41. *The cube's default material*

It has a single material, Element 0, named Default-Diffuse. The material is shown at the bottom of the Inspector.

Most of the ready-made materials from the standard assets package are specialty materials, so you will be creating your own. You will begin by creating a new folder to keep them organized.

3. From the Assets menu, choose Create ➤ Folder (Figure 2-42).

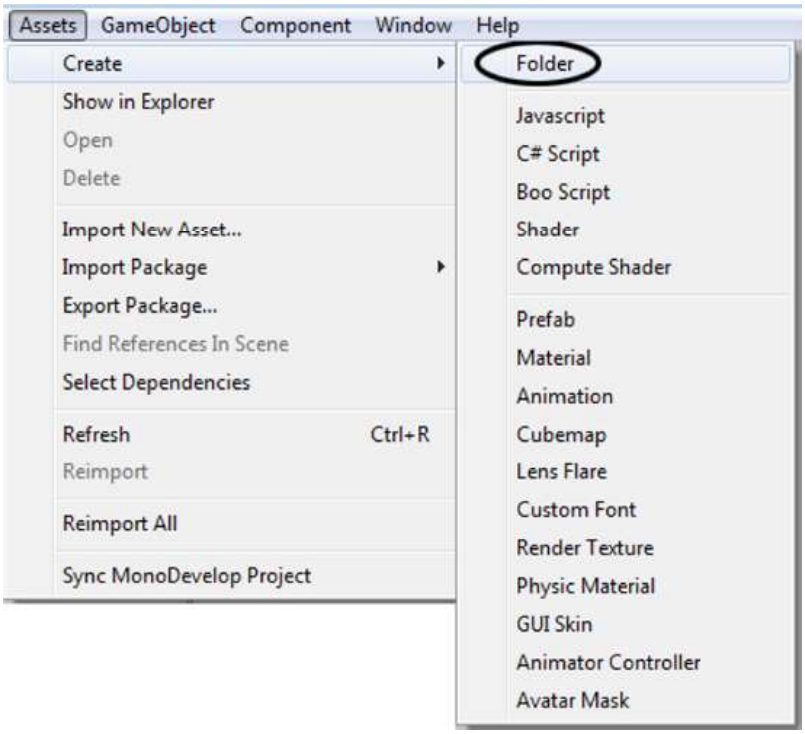

Figure 2-42. *Creating a new folder*

4. Rename the folder **My Materials**.

■ **Tip** As you create new assets for a game, it becomes important to stay organized and keep the Project view uncluttered. You will be creating several folders for that purpose throughout the book.

You can access the same menu directly in the Projects view.

5. Right-click the new My Materials folder and, from the same Create submenu, choose Material (Figure 2-43).

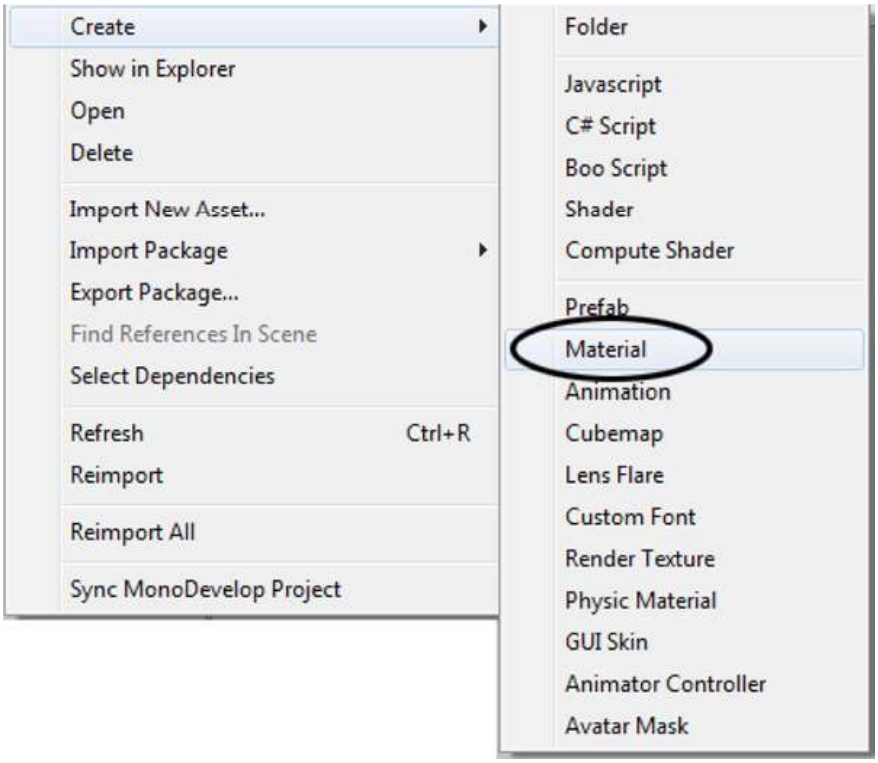

Figure 2-43. *Creating a new material*

A New Material entry appears in the Inspector.

6. Name the new material **TestBark**.
7. Drag and drop the material from the folder onto the cube in the Scene window.
8. In the Texture thumbnail window (not the Preview window), pick the Select button (Figure 2-44) and, with the preview slider all the way to the left in the Asset Selector (Figure 2-45), choose Palm Bark Diffuse.

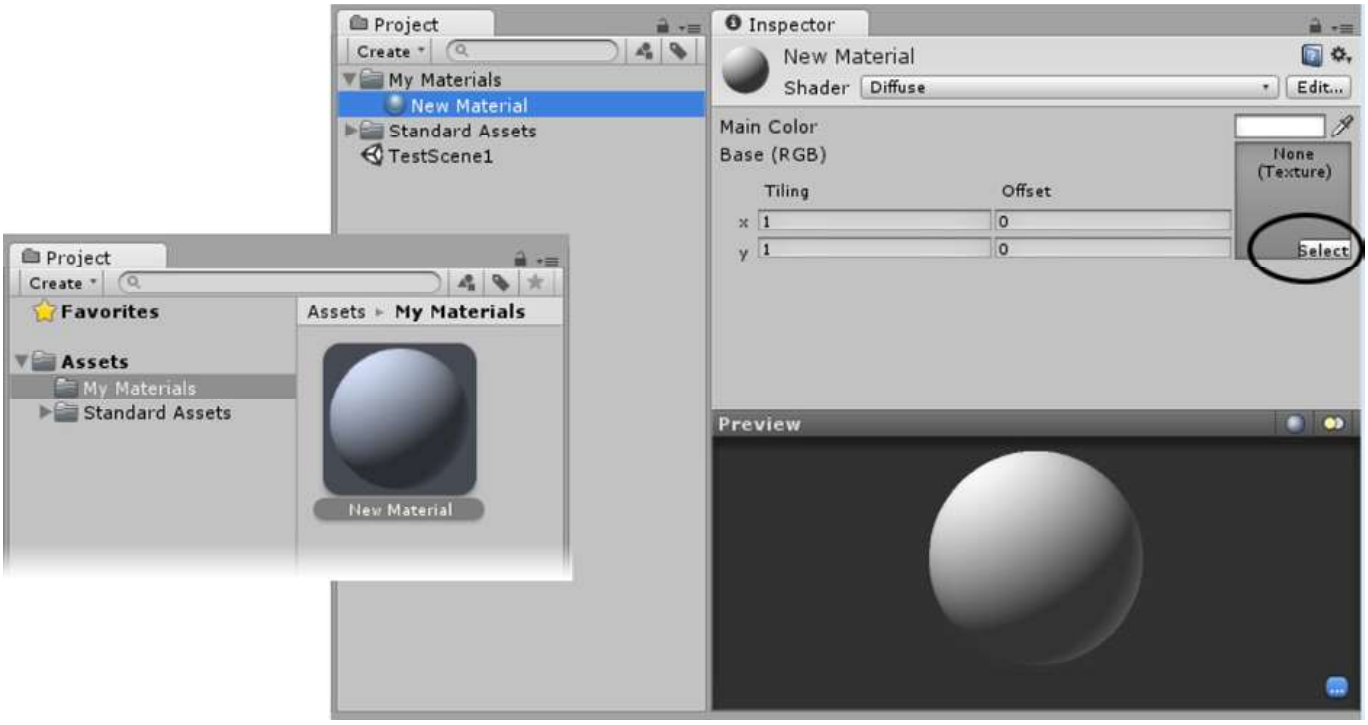

Figure 2-44. *The new material in the Project view and in the Inspector. The inset shows what the Project view looks like when using the the two-column layout*

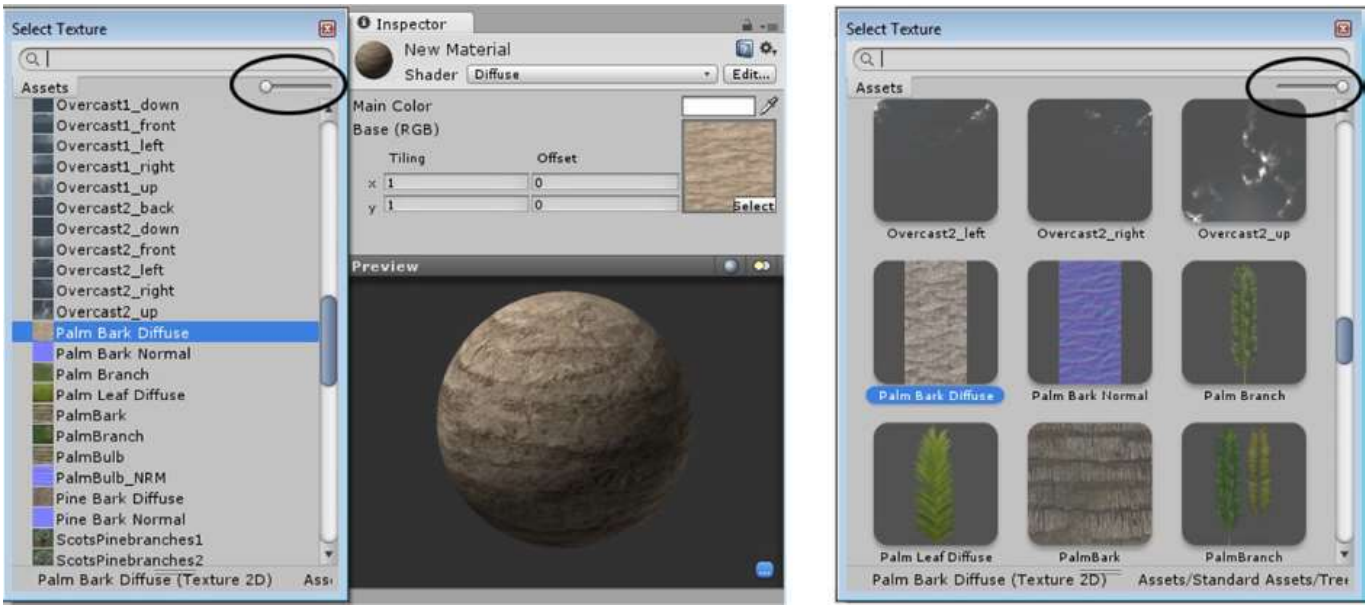

Figure 2-45. *Selecting Palm Bark Diffuse texture in the Select Texture dialog(left). The browser with small icons (left) and with large icons (right)*

■ **Tip** The Asset Selector is one way to load images, animation clips, and sound clips into components, shaders, or other parameters in your scene. It has the advantage of filtering for only the correct asset type, but the disadvantage is that it shows *all* available assets of that type. Later in the project, you will also drag assets directly from the Project view into the target parameters.

The cube now has a more interesting texture.
The default shader is a simple Diffuse shader.

9. Click the down arrow and select the Specular shader.
10. Try adjusting the Shininess slider.

Note the texture is retained as you experiment with different shaders.
Now that your simple test scene has a few objects in it, you should save it.

11. From the File menu, select Save Scene, then repeat to Save Project.

■ **Tip** While it is not always necessary to save the project as well as the scene, it is a safe habit to get into. You won't be reminded to save often, but you should do so regularly to prevent data loss in case of mishaps. Changes in the Hierarchy, whether directly or in scene object parameters, require a scene save. Changes in project settings or asset parameters in the Inspector generally require a project save.

3D Objects

This book is generally aimed toward artists and designers looking to become familiar with the Unity engine and game-engine concepts in general. However, as a review for those already familiar with 3D, as well as a short primer for those with a 2D art background, you will look at 3D from a slightly different viewpoint, to see how several concepts relate to real-time and game-associated issues.

Most people these days are quite familiar with the end-product of 3D, through film or games. Many have watched several "Making of" documentaries and already possess a basic knowledge of many of the concepts. As you have already experimented with transforms and viewport navigation in the previous sections, this time, you will look at the objects themselves.

Meshes

While outwardly they appear no different from the primitive objects you've been using as test subjects so far, meshes are stored quite differently. Instead of being stored as a list of parameters used to create a 3D object, meshes are stored as a collection of points in space. As such, they can take on an endless variety of shapes but tend to take up a lot of hard drive space. In runtime memory, both types of objects take up the same amount of space, so storage, and download time, become more important issues when deciding if it is easier to create simple meshes in Unity or import them as collapsed meshes.

Sub-Objects of a Mesh

Unity does not provide easy access to the sub-objects of a mesh, but you *can* affect them through scripting, so it is worth looking into what makes a mesh.

- **Vertex:** A vertex is the smallest sub-object, a point in space. Since vertices are little more than locations, they are not drawn. They can, however, contain information about color, opacity, and lighting, for example.
- **Edges:** Edges are the straight lines between vertices. It is worth noting that in most 3D applications, the rendered result does not contain true curved lines or surfaces; rather, they are approximations of curves created by a large number of smaller straight lines.
- **Faces:** The face, also referred to as a triangle (see Figure 2-46), is the smallest renderable part of a mesh. It is defined by three vertices, the edges that connect them, and the surface between them. A face also has what is called a normal, to tell the engine on which side to render the face. Unless you're using a shader that specifically indicates that a face should be rendered on both sides, a face is drawn only on one side.

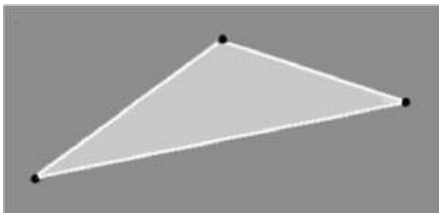

Figure 2-46. *Anatomy of a face*

Faces are often treated differently in DCC applications. In 3ds Max, for example, you can work with backfaces "turned on," but at render time and export time, they are not calculated. In Maya, the default actually creates backfaces that render and export. This increases the amount of information contained in a mesh, as well as taking extra time to cull the hidden faces at render time—a step to be avoided in real-time engines. A good practice is to get in the habit of turning off backfaces when you are modeling assets for your game.

The face normal is not to be confused with the vertex normal. Vertex normals are used to calculate the light a face will receive. A vector is traced from the light to the vertex normal, and the angle of incidence, the angle between the two, is used to calculate the amount of light received at that vertex. The smaller the angle of incidence, the higher the percentage of light it receives. A vertex with a light directly over its normal would have an angle of incidence of 0 degrees and receive the full amount of light.

Occasionally, you'll see faces with either the face normal flipped or the vertex normals flipped, giving very odd results, as shown in Figure 2-47. CAD data used for meshes often has no face normal information at all. As it tends always to be drawn double-sided, when normals are created for it, they are often mixed up.

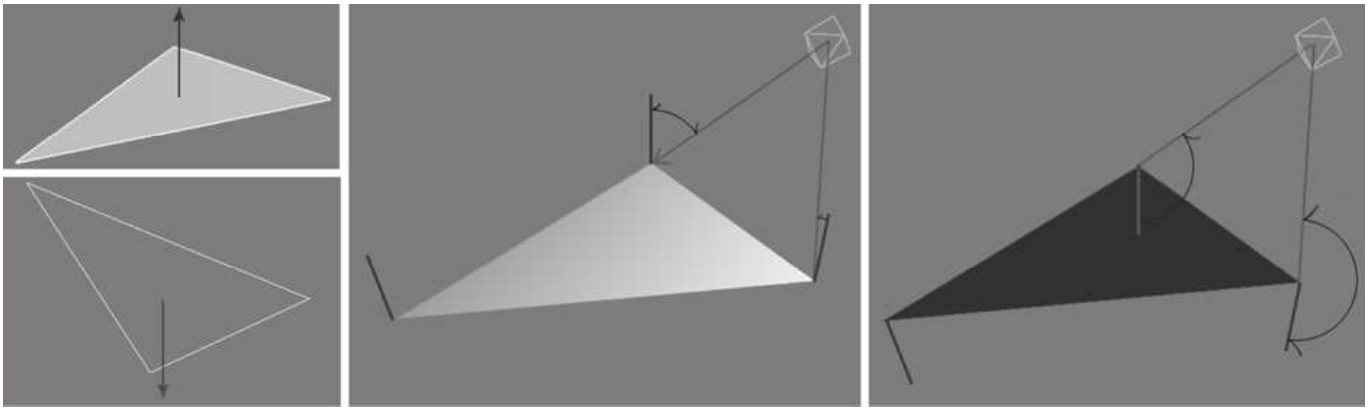

Figure 2-47. *Faces and their face normals and vertex normals. Left: The face normal determines which side the face is drawn on. Think of a perpendicular line that indicates the "up" side (top). The face normal pointing down (the face is not drawn [bottom]). Center and right: The vertex normals and angles of incidence. Note the large angle of incidence on the face, with its vertex normals pointing away from the light. The smaller the angle of incidence, the higher the percentage of light received at that vertex. The amount is blended across the face from vertex to vertex. Face normal and vertex normals pointing up, small angles of incidence (left). Face normal pointing up, but vertex normals pointing down (right)*

Mapping

Whenever an object uses a texture, or image, as part of its material, it needs mapping coordinates, to tell the renderer how to apply the image to the object's faces. Mapping can be as straightforward as a simple planar projection or as complicated as multiple UV unwraps.

Mapping coordinates are referred to as U, V, and W, where U represents one side of the map and V represents the other edge, as in Figure 2-48. W is the axis of a face normal and is of use when rotating the map about that axis.

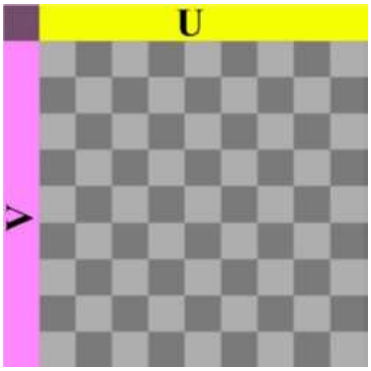

Figure 2-48. *U, V, W coordinates*

The four most common mapping types are planar, box, cylindrical, and spherical, as shown in Figure 2-49.

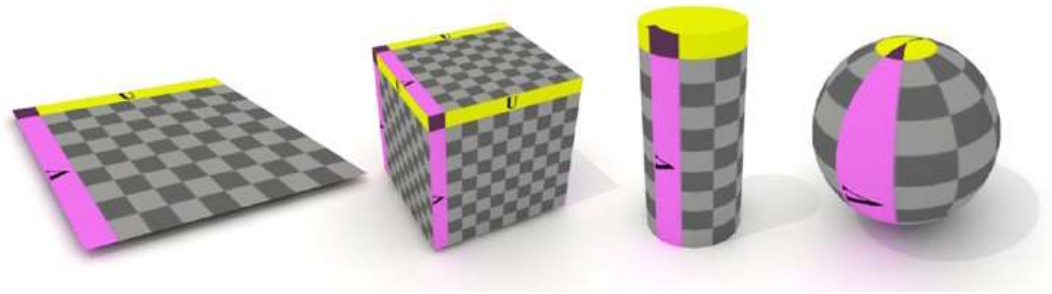

Figure 2-49. *The four most common mapping types*

- **Planar:** The image is projected from the chosen direction, through the mesh.
- **Box:** Planar projections of the image are put on each side of the object in a cube fashion. Some applications project only three sides, leaving the "back" side inside out.
- **Cylindrical:** The image is wrapped into a cylinder and projected inward and outward onto the mesh. This mapping type requires a map that tiles at the edges, so the seam won't be noticeable, or that the position of the seam is never seen. The caps are often mapped using planar projection.
- **Spherical:** The image is wrapped into a cylinder and then gathered at top and bottom. This also usually requires seamless tiling.

An object can have a combination of mapping types, as with the sign shown in Figure 2-50. The posts are cylindrical; the sign is box-mapped; and the finials are spherical mapping type.

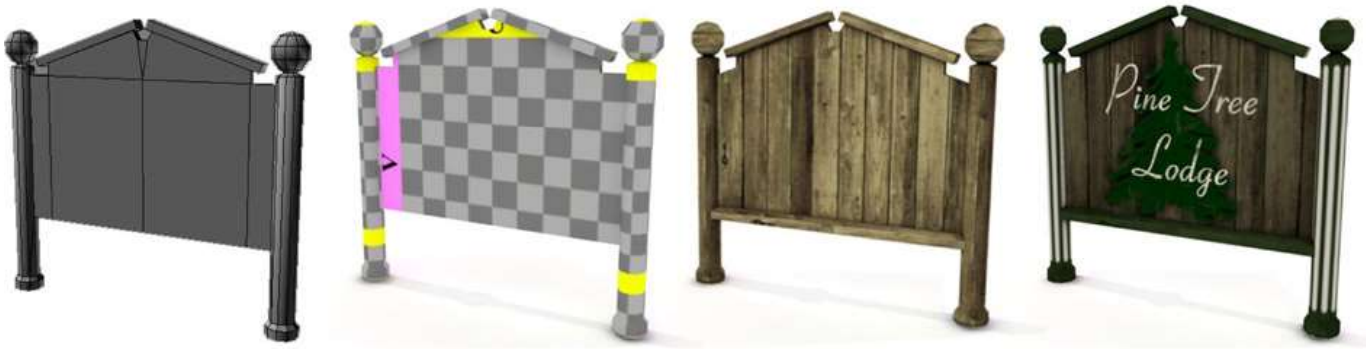

Figure 2-50. *Compound mapping on an object. The sign's geometry (far left); the sign with box, cylindrical, and spherical mapping (center left); the mapped sign with a single texture (center right); the mapped sign with multiple materials (far right)*

When a single texture is painted for an object, as opposed to using separate materials for its sub-objects, it is considered "unwrapped." The mapping is arranged (packed) to fit within a specific aspect ratio, usually 1:1, a square, and the pieces are often hand-painted to achieve a more customized texture. The tricky part about unwrapping is the compromise between less distortion but hard to paint (lots of small pieces) and more distortion but easier to paint (large pieces), as shown in Figure 2-51.

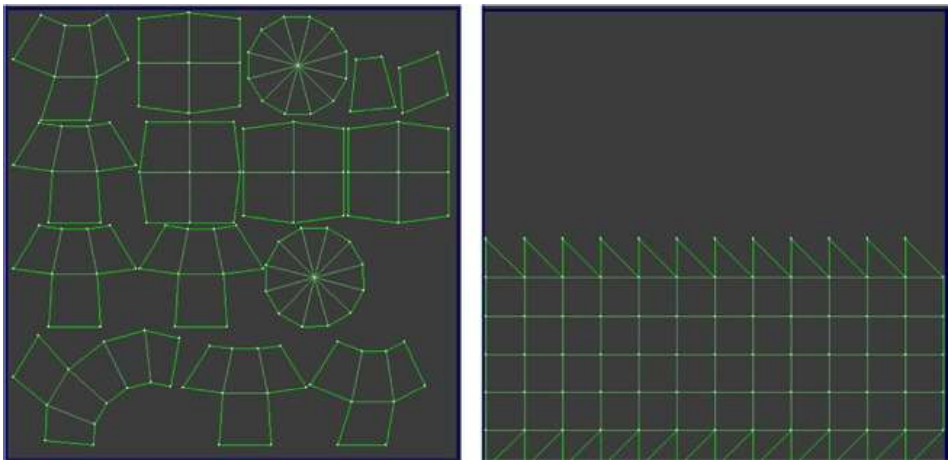

Figure 2-51. *Unwrap compromise with a sphere. Little distortion (left) and ease of painting (right)*

Using a single texture image instead of several is another trade-off. The downside is that you generally forfeit resolution in favor of custom detail. The upside is that for one-off objects, a single material is far easier to manage in the game engine. Additionally, in Unity, separate objects are internally combined, or *batched,* when they use the same material and are below a certain vertex count, for more efficient handling.

When the object is structural rather than organic, you can save the hand-painting step by rendering the conventional mapping and materials into the unwrap map, as shown in Figure 2-52. This can be touched up or altered in your painting program or used as is.

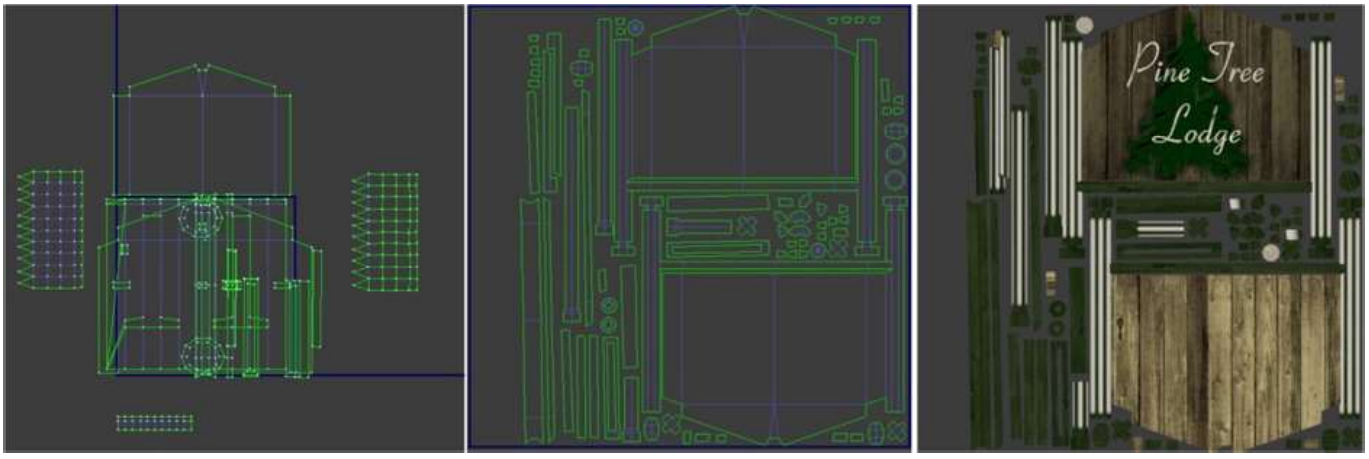

Figure 2-52. *The sign, original mapping, mapping unwrapped, packed and rendered out using the pre-existing materials and original mapping rendered to the new unwrap channel—materials "baked" together*

Typically, you also include lighting information in static objects. This is known as *baked lighting*. It can be actual scene lighting or serve as grunge or *ambient occlusion* (the darkness in nooks and crannies), as shown in Figure 2-53.

Figure 2-53. *The sign with lighting baked in (left) and fully self-illuminated, no lighting (right)*

One problem with lightmaps is that you can't have any pieces overlapping. This means that parts of the mesh that are identical or mirrored now have to be separated. The pieces all must be normalized, or shrunk to fit within the map, causing a loss of resolution.

One solution to this problem is to use two sets of UV maps—one for the diffuse texture and one for the lightmap (see Figure 2-54). Although this does take a bit more time to render, it allows for much higher resolution of the diffuse textures, as they can be tiled and/or overlapped. The lightmap doesn't require the same amount of detail and is added in at render time. The lightmap is "modulated" with the texture map, so that white is ignored and black is blended in. The Photoshop layer-type Multiply is an example of this functionality. Typically, buildings and other structures with large areas of the same physical material are handled this way, where a large lightmap will service an entire structure, and a single detailed texture map is tiled across all. It is also quite useful for terrains, where parts of the baked shadow map are swapped out at runtime for real-time shadows, and also for games and configurators that let the player customize textures in the game or application.

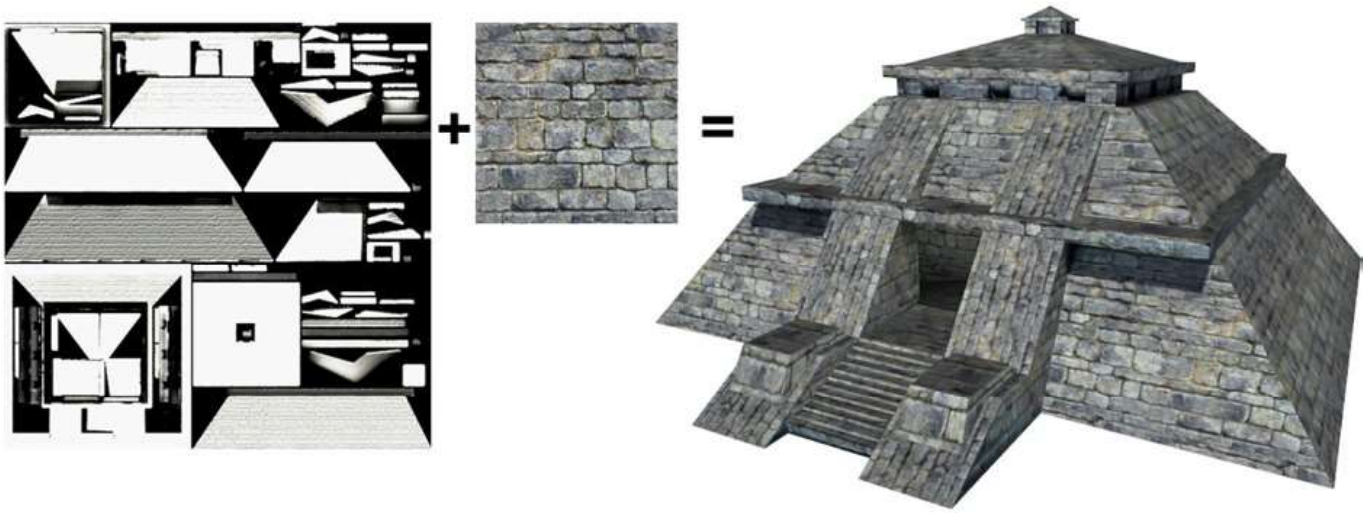

Figure 2-54. *A structure using separate diffuse and lightmaps and the resulting combination of the tiled diffuse modulated with the normalized lightmap*

Materials

Materials are generally different in real-time engines than in their DCC counterparts. At best, simple materials with a diffuse texture will import intact. Plan on spending time rebuilding materials with their shader counterparts, once they're in the engine. Most game engines have a particular process for best practices when dealing with imported textures.

Modern engines have shaders for many uses, so you can use your textures and masks to duplicate the more complex materials from your DCC application.

Summary

In this chapter, you got your first look at using Unity. You started by creating a new project, then took a quick look at each of the five main views: Scene view, where you edit and arrange your assets; Hierarchy view, where the assets in the current scene are listed; Project view, where all of the assets available to the project reside; the Game window, where you can see and interact as the player; and the Inspector, where you can access asset parameters, components, and scene or project settings. The most important takeaway from this section is that you should never rearrange the contents of the project's Assets folder from the operating system's file manager.

Next, you took a brief look at some of the menus, to get an idea of what they contain, noting especially the Help menu, with its link to the Unity Forum and Unity Answers.

You created your first object, a simple cube, in Unity and used it as a test subject to explore viewport navigation, object finding or focusing, and object transforms (move, rotate, and scale). You were able to create a directional light and also match a camera to the Scene view, so you could view your test object in the Game window. You found that selected objects displayed their parameters and other information in the Inspector and that you could modify their transforms more accurately by changing their values in the Inspector. And, finally, you learned how to create a new folder in the Project view and a new material asset to go inside it.

Having gotten a start (or perhaps a review) on exploring the 3D space of viewports, objects, and transforms, you delved a bit deeper into concepts uniquely 3D and looked at the sub-objects that make up a 3D mesh. You saw how light is calculated and why 3D faces have a front and back side. You then looked at the mapping of 3D objects, starting with simple mapping types and working your way up to fully unwrapped and light-mapped treatments, and the logic behind deciding which type to use for various objects. To finish, you found that most materials will probably require a bit of attention to get the best results after importing your meshes.

In the next chapter, you will look at scripting in Unity. You will start by creating very simple scripts to make your objects perform on cue, then deconstruct a couple of Unity's most useful standard asset scripts to start getting a feel for the syntax.