Basic Math for Game Development with Unity 3D

A Beginner's Guide to Mathematical Foundations

—

Kelvin Sung
Gregory Smith

Apress®

Basic Math for Game Development with Unity 3D

A Beginner's Guide to Mathematical Foundations

Kelvin Sung
Gregory Smith

With
Figures and Illustrations: Clover Wai

Apress®

Basic Math for Game Development with Unity 3D

Kelvin Sung
Bothell, WA, USA

Gregory Smith
Everett, WA, USA

ISBN-13 (pbk): 978-1-4842-5442-4
ISBN-13 (electronic): 978-1-4842-5443-1
https://doi.org/10.1007/978-1-4842-5443-1

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Spandana Chatterjee
Development Editor: Laura Berendson
Coordinating Editor: Divya Modi

Cover designed by eStudioCalamar

Cover image designed by Pixabay

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-5442-4. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

To my wife, Clover, for always being there and supporting me; and our girls, Jean and Ruth, for completing my life.

—Kelvin Sung

To my wife and best friend, Calli, whose love and support knows no bounds.

—Gregory Smith

Table of Contents

About the Authors

Kelvin Sung is a Professor with the Computing and Software Systems Division at the University of Washington Bothell (UWB). He holds a PhD in Computer Science from the University of Illinois at Urbana-Champaign. Kelvin's background is in computer graphics, hardware, and machine architecture. He came to UWB from Alias|Wavefront (now part of Autodesk), where he played a key role in designing and implementing the Maya Renderer, an Academy Award–winning image generation system. At UWB, funded by Microsoft Research and the National Science Foundation, Kelvin's work focuses on the intersection of video game mechanics, solutions to real-world problems, and mobile technologies. Kelvin has coauthored four books, one in computer graphics and the others in 2D game engines, including with Apress.

Gregory Smith is a graduate student in the Computer Science and Software Engineering degree program at the University of Washington Bothell. He holds an undergraduate degree in Computer Science from Northwest Nazarene University in 2018. Greg is interested in machine learning, AI, and video game design. As his senior capstone project, Greg worked on the FireMAP project at Northwest Nazarene University. Here he applied machine learning algorithms to analyze drone images in order to map burned areas left by wildfires to achieve faster and safer recovery efforts. He was also a software engineer tester at a local company while pursuing his undergraduate degree. Currently, Greg is working on a project that focuses on allowing users of virtual and augmented realities to communicate and interact with each other within these mixed reality environments.

About the Technical Reviewer

Peter Shirley is a Distinguished Research Scientist at NVIDIA. He is formally a cofounder of two software companies and was a Professor or Researcher at Indiana University, Cornell University, and the University of Utah. He holds a BS in Physics from Reed College in 1985 and a PhD in Computer Science from the University of Illinois in 1991. He is the coauthor of several books on computer graphics and a variety of technical articles. His professional interests include interactive and HDR imaging, computational photography, realistic rendering, statistical computing, visualization, and immersive environments.

Acknowledgments

This book and the projects it relates to stem from the results of the authors' attempts to understand how to engage learners in exploring knowledge related to interactive computer graphics, introductory programming, and video games. Past funding for related projects include support from the National Science Foundation for the projects "Essential Concepts for Building Interactive Computer Graphics Applications" (Award Number, CCLI-EMD, NSF, DUE-0442420) and "Game-Themed CS1/2: Empowering the Faculty" (Award Number DUE-1140410). Projects supported by Microsoft Research and Microsoft Research Connections include "XNA Based Game-Themed Programming Assignments for CS1/2" (Award Number 15871) and "A Traditional Game-Themed CS1 Class" (Award Number 16531). All of these past projects have laid the foundation for our perspectives and presentation of the materials in this book. We would also like to thank NSF officers Suzanne Westbrook, Jane Prey, Valerie Bar, and Paul Tymann for their invaluable discussions and encouragements, as well as Donald Brinkman and Kent Foster as they continue to be our best advocate and supporters at Microsoft. Lastly, we remember and continue to miss Steve Cunningham, John Nordlinger, and Lee Dirks for their early recognition of our vision and ideas.

A thank you must also go out to our students, whose honest, even when brutal, feedbacks and suggestions from CSS385: Introduction to Game Development, CSS451: 3D Computer Graphics, and CSS551: Advanced 3D Computer Graphics at the University of Washington Bothell inspired us to explore presenting these materials based on an accessible game engine. They have tested, retested, contributed to, and assisted in the formation and organization of the contents of this book. The second author of this book is a recent alumnus of CSS551.

It must also be mentioned that the teaching brown bag hosted by Yusuf Pisan offered the opportunity for the discussions with Yusuf, Johnny Lin, Lesley Kalmin, and Mike Stiber on the topics of linear algebra applications which sparked the initial idea for this book. A sincere thank you goes to Yusuf for his enthusiasm and energy in organizing us, and, of course, for the delicious-looking Tim Tam; one day, I will try them.

The second author was partially supported as a research assistant and a generous grant (Award Number 09-8323/1902KS) from the Division of Computing and Software Systems, University of Washington Bothell.

Lastly, a thank you must go to Peter Shirley, our technical editor, whose frank and precise comments made this a much easier to understand book.

The vehicle models used are free assets, UAA - City Props - Vehicles, downloaded from the Unity Asset Store under the Unity-EULA. The cone shape that represents the arrow heads for the axis frames and vectors in all examples is created based on the utilities developed and shared by Wolfram Kresse available at `https://wiki.unity3d.com/index.php/CreateCone`. The cosine function plot from Figure 5-5 is based on a screenshot taken from `www.desmos.com/calculator/nqfu5lxaij`.

Introduction

Welcome to *Basic Math for Game Development with Unity 3D*. Because you have picked up this book, you are probably interested in finding out more about the mathematics involved in game development or, maybe, in the details of fascinating applications like Unity. This can be the perfect book to begin with your exploration.

This book uses interactive examples in Unity to present each mathematical concept discussed, taking you on a hands-on journey of learning. The coverage of each topic always follows a pattern. First, the concept and its relevancy in video game functionality are described. Second, the mathematics, with a focus on applicability in game development and interactive computer graphics, are derived. Finally, an implementation of the concept and derived mathematics are demonstrated as an example in Unity.

Through interacting with these examples, you will have the opportunity to explore the implications and limitations of each concept. Additionally, you can examine the effects of manipulating the various related parameters. Lastly, and very importantly, you can study the accompanied source code and understand the details of the implementations.

In Chapter 2, you will begin by reviewing simple number intervals in the Cartesian Coordinate System. Chapters 3 and 4 let you examine and learn about vectors and the rules of their operations to formally relate positions in 3D space. Chapters 5 and 6 study the vector dot and cross products to relate vectors and the space that defines them. Through this book, you will learn the mathematical and implementation details of bounding boxes, bounding spheres, motion controls, ray castings, the projection of points to lines to compute intersections between fast traveling objects, and the insights into 2D planes to create shadows, compute reflections, and much more!

Who Should Read This Book

This book is targeted toward video game enthusiasts and hobbyists who have some background in basic object-oriented programming. For example, if you are a student who has taken an introductory programming course or are a self-taught programming enthusiast, you will be able to follow the concepts and code presented in this book with little trouble. If you do not have any programming background in general, it is suggested that you first become comfortable with the C# programming language before tackling the content provided in this book.

Besides a basic understanding of object-oriented programming, you will also need to be familiar with the Cartesian Coordinate System, basic algebra, and knowledge in trigonometry. Experience with and working knowledge of Unity are not required.

Code Samples

Every chapter in this book includes examples that let you interactively experiment with and learn the new materials. You can download the source code for all the projects from the following page: www.apress.com/9781484254424.

CHAPTER 1

Introduction and Learning Environment

After completing this chapter, you will be able to

- Know the details of what this book is about
- Understand the style that this book uses to present concepts
- Install Unity and an Integrated Development Environment (IDE) for developing programming code
- Access the accompanying source code and run the example projects
- Understand the Unity terminology used throughout this book
- Begin to appreciate the intricate details of math for game development

Introduction

When you think of math in a video game, you may picture health bars, attack stats, experience points, and other game mechanics. You may not consider the underlying math that enables the in-game physics world,

K. Sung and G. Smith, *Basic Math for Game Development with Unity 3D*,
https://doi.org/10.1007/978-1-4842-5443-1_1

such as calculating gravity, movements, or enemy chasing behaviors. Additionally, you may not consider physical interaction in a mathematical manner, such as collisions between different objects and the reflections of these objects after they collide. These underlying mathematical computations are critical to implementing a successful video game. When creating a game, whether you intend on using a game engine or you intend on performing the computations yourself, understanding the details and knowing how the underlying mathematics work and when to use them to create what you want, where you want, is vital.

Traditionally math is taught without any application contexts. Typically, theories are developed based on abstract symbols, formulas are derived to support these theories, and then numbers are used to verify the formulas. You are tested on whether you can generate the correct solution based on how the formulas are applied. It is believed that learning math in this manner has the benefit of granting the learner the ability to understand the concepts being taught at the pure abstraction level. Then, once understood, the application of these concepts to different disciplinary contexts becomes straightforward. For many learners, this assumption is certainly true. However, for other types of learners, it can be difficult to appreciate the intricate details in the abstract without concrete examples or applications to build off. This fact is recognized by educators and often story problems are introduced after a basic understanding is established to help learners gain insights and appreciate the formulas. This learning approach is taken on and exploited in the context of linear algebra and video games.

This book takes you on the journey of learning linear algebra, a branch of mathematics that is the foundation of interactive graphical applications, like video games. While the underlying theories can be abstract and complicated, the application of these theories in graphical object interactions is relatively straightforward. For this reason, this book approaches linear algebra topics in a concrete manner, based around game-like examples that you can interact with. Through this book, you will learn a flavor of linear algebra that is directly applicable to video games and interactive computer graphics as a whole.

Every math concept presented in this book is accompanied with concrete examples that you can interact with and are relevant to video game development. It is the intent of this book that you will learn and know how to apply the concepts in solving the problems you are likely to encounter during game development. A direct consequence of this focused approach is that readers may find it challenging to apply the knowledge gained throughout this book to other disciplines, like machine learning or computer vision. For example, the dot product, which will be covered in Chapter 5, can be used to calculate intersection positions, and it can also be used in machine learning algorithms as a data reduction tool; however, this book will only focus on the video game applications of the dot product. If you are looking for general knowledge in linear algebra, you should consider a more traditional textbook. Such a book is likely to cover concepts at levels that are suitable for applications for multiple problem spaces. If you are interested in solving problems specific to interacting graphical objects, especially for game development, then this is the perfect book for you.

After the introduction to the game engine and terminologies in this chapter, Chapter 2 reviews the Cartesian Coordinate System and number intervals leading to the exploration of one of the most widely used tools in game development—bounding boxes. Chapter 3 continues bounding volume exploration by examining bounding spheres while also beginning the investigation of relationships between positions. Chapter 4 introduces vectors to formalize the relationships between positions in 3D space and applies vector concepts in controlling and manipulating object motions under external effects like wind or current flow. Chapter 5 presents the vector dot products to relate vectors, represents line segments based on vectors, and demonstrates the application of these concepts in computing distances between objects and motion paths when approximating potential collisions. Chapter 6 discusses the vector cross product, examines the space that defines vectors, defines vector plane equation, and illustrates the application of these concepts in computing intersections and reflections of moving objects and 2D planes. Finally, Chapter 7 summarizes all of the concepts presented in an aggregated example.

Choice of Unity Engine

Unity is the choice of platform for presenting the mathematical concepts covered in this book for three reasons. First, Unity provides elaborate utilities and efficient support for its user to implement and visualize solutions based on mathematical formulas. Its Application Programming Interface (API) implements the basic and many advanced linear algebra functionalities, while the Entity-Component-System (ECS) game object architecture allows straightforward user scripting. These qualities give Unity a close pairing of math concepts to your programming code, assisting in the visualization of the mathematical solution that you are trying to understand. This close pairing cannot be understated and is the backbone of this book.

The second reason for choosing Unity is that, being a game engine, the system allows for a high degree of intractability with the solution as well as the ability to visualize that solution. For example, in addition to being able to examine the results of a ray and 2D plane intersection computation in real time, you will also be able to manipulate the ray and the 2D plane to observe the effects on the intersection. The ability to interact, manipulate, and examine the application of mathematical concepts in real time will give you a greater understanding and appreciation for that concept. Finally, Unity is chosen because there is no better way to learn math concepts for video games than through a popular game engine!

While this book is meant for readers who may be interested in building a video game in Unity, the focus of this book is on the math concepts and their implementations and not on how to use Unity. This book teaches the basic mathematical concepts that are relevant to video game development using Unity as a teaching instrument. This book does not teach how to use the math provided by Unity in building video games. You should focus on understanding the math above the Unity-specific functionality. For example, a position in 3D space in Unity is located at `transform.localPosition`; you should focus on working with that position and not be concerned about the `Unity.Transform` class. Ultimately, you should be able to take what you have learned in this book and apply to developing games in any game engine.

Note Unity Technologies is the name of the company; the game engine is most often referred to as Unity, though it is sometimes called Unity 3D. For simplicity, this book refers to the entire game engine system as Unity.

Setting Up Your Development Environment

There are two main applications that you will work with when using Unity. The first is the game engine editor, which will be referred to as Unity or Unity Editor throughout this book. The Unity Editor can be thought of as the graphical interface to the Unity game engine. The second application you will need is a script editing Integrated Development Environment (IDE). Microsoft's Visual Studio Community 2017 is the IDE of choice for developing the C# script examples in this book. This software will be referred to as the Script Editor, or the IDE, throughout the rest of this book.

To begin your download and installation of Unity and Visual Studio Community 2017, go to `https://store.unity.com/download?ref=personal`, accept the terms, and then download Unity Hub.

Note If you ever find yourself stuck at a certain point in this book, whether on installing Unity or just using it, there is a plethora of tutorials online, many of which were referenced in the development of this book and will be listed at the end of this chapter.

Notes on Installing Unity

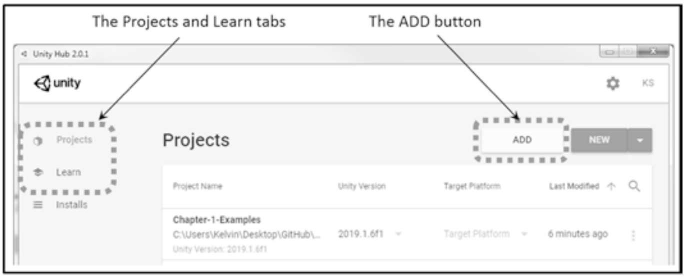
This book is based on Unity in its most basic form. Unless you know what to specify when installing features or desire extra features, it is suggested you follow the default settings. Please begin downloading, installing, and launching the Unity Hub if you haven't already. When Unity Hub is up and running, navigate to the **Installs** tab on the left side, and select the **ADD** button in the top right. From here, you will be prompted with a long list of different Unity versions. The version that this book uses is 2019.1.6f1. If you do not see this version in the selected list, you can go to this link `https://unity3d.com/get-unity/download/archive` and find it there to download. It should be noted that while this book is based on Unity 2019.1.6f1, any version at or newer than this version should suffice but is not guaranteed.

After selecting your Unity version, you will be prompted with options to install extra features. As mentioned previously, this textbook only requires the default options. These options, if you are running on Windows 10, should only be the suggested IDE, "Microsoft Visual Studio Community 2017," and "Documentation." If you already have Visual Studio 2017 installed, then "Documentation" will be the only pre-checked option. Once you have selected all the features you want, begin the install process and then move onto the next section to begin familiarizing yourself with the source code used throughout this book.

Unity Editor Environment

It should be noted, again, that in this book Unity is used as a tool for learning math concepts for game development and not as a game building editor. This means many Unity-specific and game building-related information that do not pertain to the concept at hand will simply be skipped. For example, this book does not discuss how to create or save Scenes or how to build a final executable game. If these are subjects of interests, you should consider research through the many online tutorials or for example refer to the Learn tab of the Unity Hub. It should also be noted that all examples throughout this book will be run and interacted with through the editor and not as a game. This will become clearer as the first example is discussed.

Now that you have Unity and the IDE ready to go, you can refer to the GitHub repository located at `www.apress.com/9781484254424`. After downloading the repository, open Unity Hub and add the `Chapter-1-Examples` project. Directions on how to do this can be seen in Figure 1-1.



Project Name	Unity Version	Target Platform	Last Modified
Chapter-1-Examples C:\Users\Kelvin\Desktop\GitHub\... Unity Version: 2019.1.6f1	2019.1.6f1	Target Platform	6 minutes ago

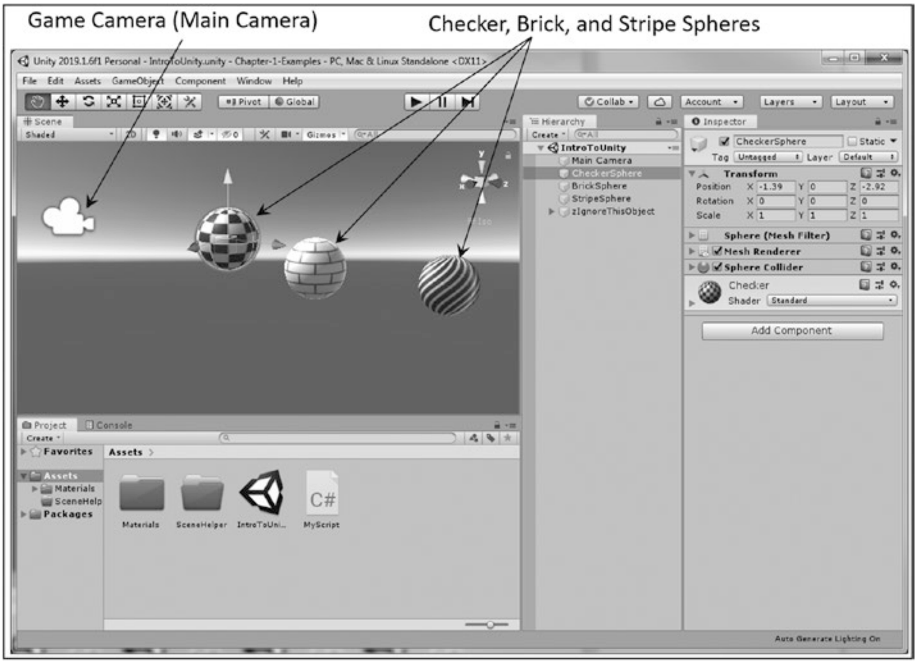
Figure 1-1. *Opening `Chapter-1-Examples` (the Intro to Unity Project) from Unity Hub*

As Figure 1-1 shows, to add a project, navigate to the Projects tab and then select the ADD button. From here, navigate to where you downloaded the source code to this book. You will notice that the file structure is organized according to chapters. The first example you should open using the ADD button is `Chapter-1-Examples`. Note that after a project is added, you need to click the newly added project to launch it.

Figure 1-1 also establishes where the Learn tab is located. Here you can view and select Unity- and community-sponsored tutorials from beginning topics such as “Play & Edit Mode” to more complicated ones like “2D Roguelike.” At the end of this chapter, there are some additional suggestions as to which tutorials to follow if you are new to Unity or just need a refresher.

Opening the Intro to Unity Project

To open a project from Unity Hub, simply click it. Once you open `Chapter-1-Examples`, you should be confronted with a window similar to the screenshot in Figure 1-2.


Game Camera (Main Camera)
Checker, Brick, and Stripe Spheres
Unity 2019.1.6f1 Personal - IntroToUnity.unity - Chapter-1-Examples - PC, Mac & Linux Standalone <DX11>
File Edit Assets GameObject Component Window Help
Pivot Global
Collab Account Layers Layout
Scene Shaded 2D Gizmos
Hierarchy Create
IntroToUnity
Main Camera
CheckerSphere
BrickSphere
StripeSphere
zIgnoreThisObject
Inspector
Project Console Create
Favorites
Assets
Materials
SceneHelp
Packages
Materials SceneHelper IntroToUni... MyScript
C#
Auto Generate Lighting On

CheckerSphere ☐ Static
Tag Untagged Layer Default
Transform
Position X -1.39 Y 0 Z -2.92
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Sphere (Mesh Filter)
☑ Mesh Renderer
☑ Sphere Collider
Checker
Shader Standard
Add Component

***Figure 1-2.** Running the Intro To Unity scene in the Chapter-1-Examples project*

Figure 1-2 shows a very simple scene. There is the game view camera, also known as the `MainCamera`, which shows what the player would see when the game is running, and three different spheres. Each sphere is named after the design pattern placed upon it: `CheckerSphere`, `BrickSphere`, and `StripeSphere`. The purpose of this example is to familiarize you with how examples are organized and to establish terminologies that will be used throughout the book.

Working with the Unity Editor

Figure 1-2 is an example of what the Unity Editor looks like and is one of the two editors you will be working in. The other editor, the Script Editor, or IDE, will be discussed later. Figure 1-3 will break down the Unity Editor.

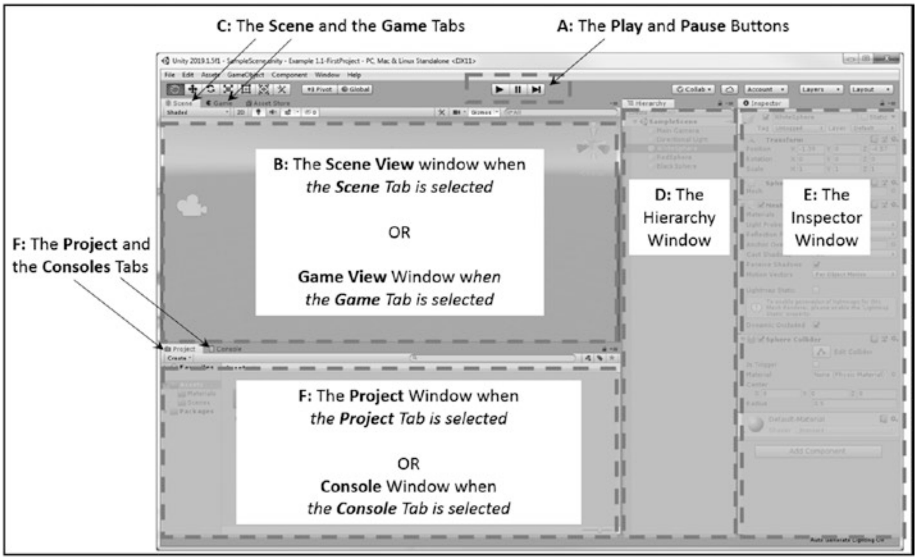

C: The Scene and the Game Tabs
A: The Play and Pause Buttons
B: The Scene View window when the Scene Tab is selected
OR
Game View Window when the Game Tab is selected
D: The Hierarchy Window
E: The Inspector Window
F: The Project and the Consoles Tabs
F: The Project Window when the Project Tab is selected
OR
Console Window when the Console Tab is selected

Figure 1-3. *The Unity Editor Environment*

Figure 1-3 overlaps the editor in Figure 1-2 with labels identifying the different windows presented by the Unity Editor and establishes the terminologies that will be used from here on:

- **A: The Play and Pause buttons**: In the top-center area, you can see the Play and Pause buttons. These buttons control the running (or playing) of the game. Feel free to click the Play button, give the system a few seconds to load, and then observe the movements of the spheres in the scene. If you click the Play button again, the game will stop running. You will learn more about and work with these buttons later.
- **B: The Scene View window**: The main 3D window in the top-left region of the Unity Editor is the main area for performing interactive editing. In Figure 1-2, this window is displaying the Scene View of the game.
- **C: The Scene and the Game View tabs**: Above the Editor Window (B), you can spot the Scene and Game tabs. If you select the Game tab, then Unity will switch to the Game View which is what the `MainCamera` from Figure 1-2 sees. An example of the Scene View next to the Game View can be seen in Figure 1-4.

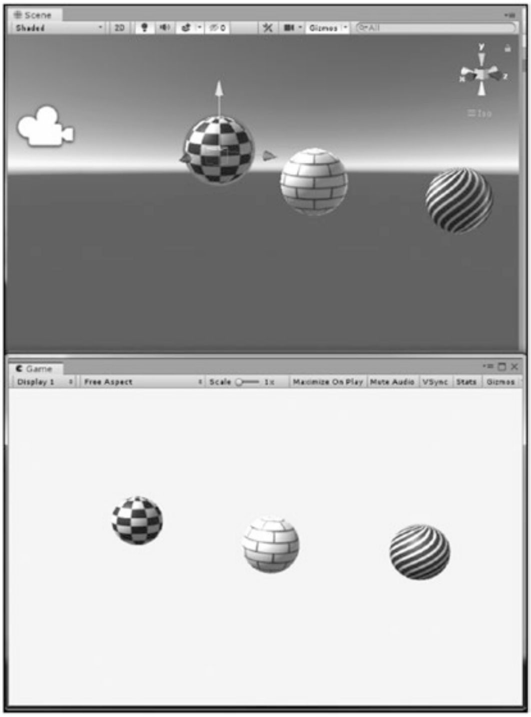


Figure 1-4. *The Scene View (top) and the Game View (bottom)*

Note Please pay attention to the differences between the Scene and Game Views. The Scene View is meant for the game designer to set up a game scene, while the Game View is what a player of the game would observe while playing the game. While both views can be invaluable tools for examining the intricate details of the mathematical concepts, you will be working exclusively with the Scene View.

Note To help distinguish between the Scene and the Game Views, as depicted in Figure 1-4, in all the examples for this book the Scene View has a skybox like background, while the Game View window has a constant, light blue backdrop. Once again, you will be working exclusively with the Scene View, the view with the skybox-like background.

EXERCISE

Working with the Scene View Window

Left-click and drag the Scene View tab to see that you can configure and place the Scene View window at different configuration locations throughout the Unity Editor or even outside as an independent window. This is the case for most of the Unity tabs, including the Game View window. Figure 1-4 shows the Scene View and Game View windows as two separate windows that can be examined simultaneously.

Figure 1-5 is a close-up view of the Hierarchy Window, which is labeled as D in Figure 1-3.

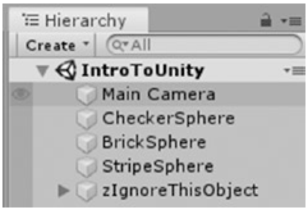


Figure 1-5. *The Hierarchy Window*

- **D: The Hierarchy window**: In the Unity Editor, this window (Figure 1-5) is typically anchored to the right of the Scene View but left of the Inspector Window (E). The Hierarchy Window contains every object and its parental relationship to other objects in the scene. Just like the Scene View and Game View, the Hierarchy Window can be moved and placed wherever you desire. You should observe the different objects within the Hierarchy Window. There is the `MainCamera`, which, as mentioned previously, is the camera of the Game View; the `CheckerSphere`, which is the checkered sphere; as well as the `BrickSphere` and `StripeSphere`, which also correspond to their object's descriptions. Finally, there is the `zIgnoreThisObject` object; this last object supports the setup of the game environment for the learning of math concepts specific to each example. You will never need to interact with this object, and therefore this book will ignore this object as its details can be distracting. You are, of course, more than welcome to examine and explore this object, and any others, at your leisure.

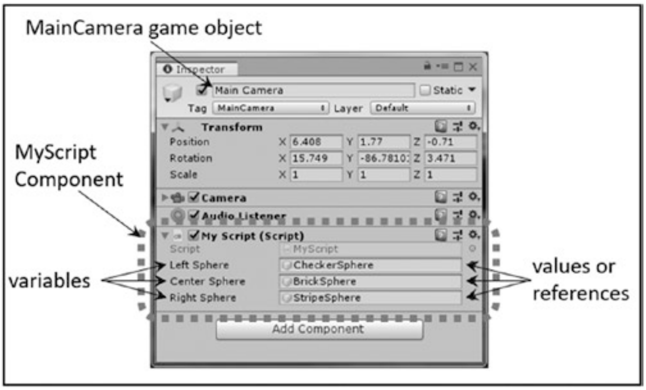
Note Try clicking on the different objects in the Hierarchy Window and observe how the Scene View highlights the object you have selected while the Game View does not. This simple feature underscores how the Scene View is meant for scene edits while the Game View is not.

EXERCISE

Observe Differences Between the Scene View and Game View

Select different spheres in the Hierarchy Window and switch between the Scene and Game Views to observe the differences between these two views. You should notice that the selected sphere is highlighted in the Scene View and not in the Game View. It is essential to differentiate between these two views when you manipulate the scene in examining concepts.

Figure 1-6 is a close-up view of the Inspector Window, which is labeled as E in Figure 1-3.



☑ Main Camera □ Static
Tag MainCamera Layer Default
Position X 6.408 Y 1.77 Z -0.71
Rotation X 15.749 Y -86.7810: Z 3.471
Scale X 1 Y 1 Z 1
☑ My Script (Script)
Script MyScript
Left Sphere CheckerSphere
Center Sphere BrickSphere
Right Sphere StripeSphere

***Figure 1-6.** Inspector Window with the `MainCamera` selected in the Hierarchy Window*

- **E: The Inspector Window**: The Inspector Window (Figure 1-6) displays the details of the selected object in the Hierarchy Window for the user to inspect and manipulate. The Inspector Window is typically to the right of the Hierarchy Window. Just like all other windows described, it can be placed wherever you want. The selected object being displayed in Figure 1-6 is the `MainCamera`. Notice that there are multiple components attached to this object including `Transform`, `Camera`, `Audio Listener`, and `MyScript`. Figure 1-6 shows that you can expand and compress each of the components to examine or hide their details. In this case, the `Transform` and `MyScript` components are expanded. The `MyScript` component is the custom script developed for this book. Note that on the left side of the `MyScript` component are the names of the public variables defined in the script: `Left Sphere`, `Center Sphere`, and `Right Sphere`. Directly across from these variable names, you can see their values or the objects that the corresponding variables reference: `CheckerSphere`, `BrickSphere`, and `StripeSphere`. These aspects of the `MyScript` component will be explained in more detail in the next section.

- **F: The Project and the Console windows and tabs**: The Project Window displays the file structure of your project. This is where scripts, prefabs, materials, and everything else that will be loaded into your game are located. The Console Window is where Unity will output debug messages, warnings, and errors, all of which can be very helpful in debugging your code if something goes wrong. The Project Tab and Console Tab allow you to switch between these two windows just like the Game View tab and Scene View tab do. These windows can also be moved around and placed wherever you decide.