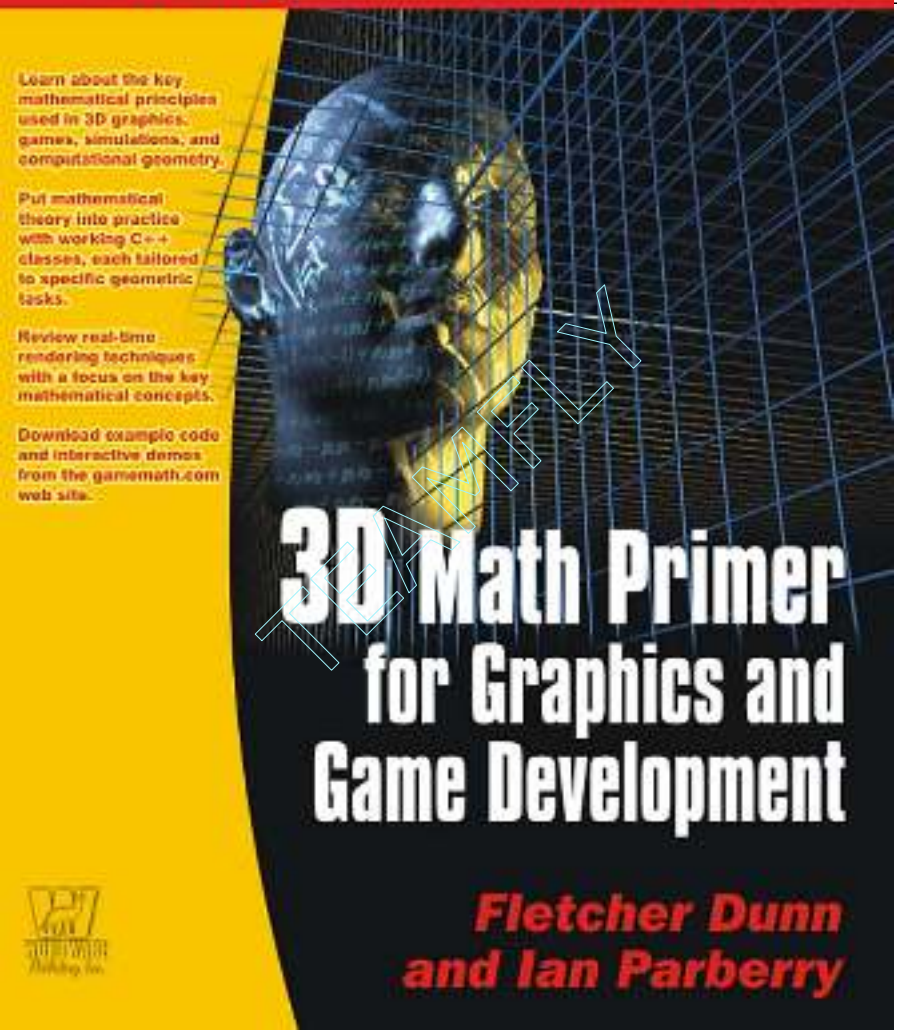
WORDWARE GAME MATH LIBRARY



Learn about the key mathematical principles used in 3D graphics, games, simulations, and computational geometry.

Put mathematical theory into practice with working C++ classes, each tailored to specific geometric tasks.

Review real-time rendering techniques with a focus on the key mathematical concepts.

Download example code and interactive demos from the gamemath.com web site.

3D Math Primer for Graphics and Game Development

Fletcher Dunn
and Ian Parberry

3D Math Primer for Graphics and Game Development

Fletcher Dunn
and Ian Parberry

Wordware Publishing, Inc.

Library of Congress Cataloging-in-Publication Data

Dunn, Fletcher.
3D math primer for graphics and game development / by Fletcher Dunn and Ian Parberry.
p. cm.
ISBN 1-55622-911-9
1. Computer graphics. 2. Computer games--Programming. 3. Computer science--Mathematics.
I. Parberry, Ian. II. Title.
T385 .D875 2002
006.6--dc21 2002004615
CIP

2320 Los Rios Boulevard
Plano, Texas 75074

Printed in the United States of America

ISBN 1-55622-911-9
10 9 8 7 6 5 4 3 2 1
0205

All inquiries for volume purchases of this book should be addressed to Wordware Publishing, Inc., at the above address. Telephone inquiries may be made by calling:

(972) 423-0090

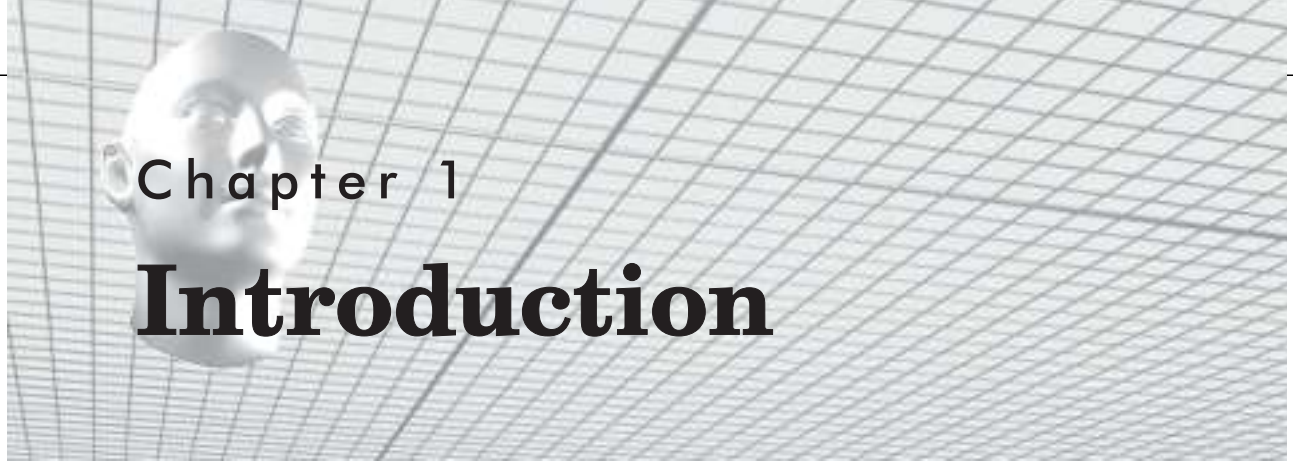
Contents

Acknowledgments

Fletcher would like to thank his high school senior English teacher. She provided constant encouragement and help with proofreading, even though she wasn't really interested in 3D math. She is also his mom. Fletcher would also like to thank his dad for calling him every day to check on the progress of the book or just to talk. Fletcher would like to thank his boss, Mark Randel, for being an excellent teacher, mentor, and role model. Thanks goes to Chris DeSimone and Mario Merino as well, who (unlike Fletcher) actually have artistic ability and helped out with some of the more tricky diagrams in 3D Studio Max. A special thanks to Allen Bogue, Jeff Mills, Jeff Wilkerson, Matt Bogue, Todd Poynter, and Nathan Peugh for their opinions and feedback.

Ian would like to thank his wife for not threatening to cause him physical harm when he took multiple weekends away from his family duties to work on this book. He would like to thank his children for not whining *too* loudly. Also, thanks to the students of his Advanced Game Programming class at the University of North Texas in Spring 2002 for commenting on parts of this book in class. Oh, and thanks to Keith Smiley for the dead sheep.

Both authors would like to extend a very special thanks to Jim and Wes at Wordware for being very patient and providing *multiple* extensions when it was inconvenient for them to do so.



Chapter 1

Introduction

1.1 What is 3D Math?

This book is about 3D math, the study of the mathematics behind the geometry of a 3D world. 3D math is related to computational geometry, which deals with solving geometric problems algorithmically. 3D math and computational geometry have applications in a wide variety of fields that use computers to model or reason about the world in 3D, such as graphics, games, simulation, robotics, virtual reality, and cinematography.

This book covers theory and practice in C++. The "theory" part is an explanation of the relationship between math and geometry in 3D. It also serves as a handy reference for techniques and equations. The "practice" part illustrates how these concepts can be applied in code. The programming language used is C++, but in principle, the theoretical techniques from this book can be applied in any programming language.

This book *is not* just about computer graphics, simulation, or even computational geometry. However, if you plan to study those subjects, you will definitely need the information in this book.

1.2 Why You Should Read This Book

If you want to learn about 3D math in order to program games or graphics, then this book is for you. There are many books out there that promise to teach you how to make a game or put cool pictures up on the screen, so why should you read *this* particular book? This book offers several unique advantages over other books about games or graphics programming:

- **A unique topic**. This book fills a gap that has been left by other books on graphics, linear algebra, simulation, and programming. It is an *introductory* book, meaning we have focused our efforts on providing thorough coverage on fundamental 3D concepts — topics that are normally glossed over in a few quick pages or relegated to an appendix in other publications (because, after all, you already know all this stuff). Our book is definitely the book you should read *first*, before buying that "Write a 3D Video Game in 21 Days" book. This book is not only an introductory book, it is also a reference book — a "toolbox" of equations and techniques that you can browse through on a first reading and then revisit when the need for a specific tool arises.

- **A unique approach**. We take a three-pronged approach to the subject matter: *math*, *geometry*, and *code*. The *math* part is the equations and numbers. This is where most books stop. Of course, the math is important, but to make it powerful, you have to have good intuition about how the math connects with the *geometry*. We will show you not just one but *multiple* ways to relate the numbers with the geometry on a variety of subjects, such as orientation in 3D, matrix multiplication, and quaternions. After the intuition comes the implementation; the *code* part is the practical part. We show real usable code that makes programming 3D math as easy as possible.
- **Unique authors**. Our combined experience brings together academic authority with in-the-trenches practical advice. Fletcher Dunn has six years of professional game programming experience and several titles under his belt on a variety of gaming platforms. He is currently employed as the principal programmer at Terminal Reality and is the lead programmer on *BloodRayne*. Dr. Ian Parberry has 18 years of experience in research and teaching in academia. This is his sixth book, his third on game programming. He is currently a tenured full professor in the Department of Computer Sciences at the University of North Texas. He is nationally known as one of the pioneers of game programming in higher education and has been teaching game programming to undergraduates at the University of North Texas since 1993.
- **Unique pictures**. You cannot learn about a subject like 3D by just reading text or looking at equations. You need pictures, and this book has plenty of them. Flipping through, you will notice that in many sections there is one on almost every page. In other words, we don't just *tell* you something about 3D math, we *show* you. You'll also notice that pictures often appear beside equations or code. Again, this is a result of our unique approach that combines mathematical theory, geometric intuition, and practical implementation.
- **Unique code**. Unlike the code in some other books, the classes in this book are not designed to provide every possible operation you could ever want. They *are* designed to perform specific functions very well and to be easy to understand and difficult to misuse. Because of their simple and focused semantics, you can write a line of code and have it work the *first* time, without twiddling minus signs, swapping sines and cosines, transposing matrices, or otherwise employing "random engineering" until it looks right. Many other books exhibit a common class design flaw of providing *every* possible operation when only a few are actually useful.
- **A unique writing style**. Our style is informal and entertaining, but formal and precise when clarity is important. Our goal is not to amuse you with unrelated anecdotes, but to engage you with interesting examples.
- **A unique web page**. This book does not come with a CD. CDs are expensive and cannot be updated once they are released. Instead, we have created a companion web page, `gamemath.com`. There you will be able to experience interactive demos of some of the concepts that are the hardest to grasp from text and diagrams. You can also download the code (including any bug fixes!) and other useful utilities, find the answers to the exercises, and check out links to other sites concerning 3D math, graphics, and programming.

1.3 What You Should Know Before Reading This Book

The *theory* part of this book assumes a prior knowledge of basic algebra and geometry, such as:

- Manipulating algebraic expressions
- Algebraic laws, such as the associative and distributive laws
- Functions and variables
- Basic 2D Euclidian geometry

In addition, some prior exposure to trigonometry is useful, but not required. A brief review of some key mathematical concepts is included in Appendix A.

For the *practice* part, you need to understand some basics of programming in C++:

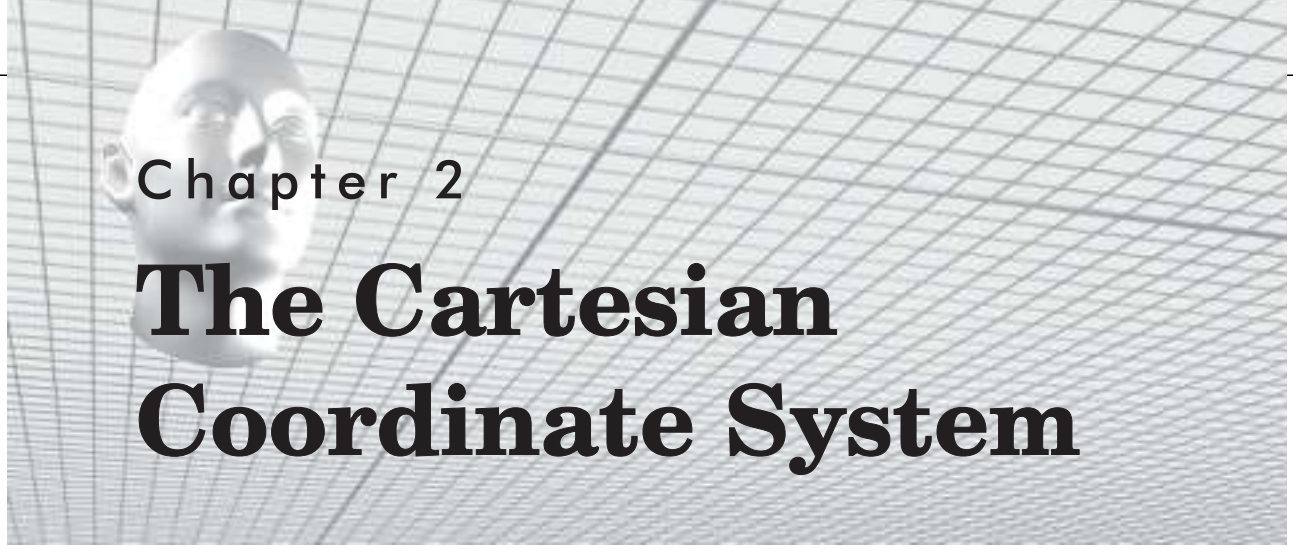
- Program flow control constructs
- Functions and parameters
- Object-oriented programming and class design

No specific compiler or target platform is assumed. No "advanced" C++ language features are used. The few language features that you may be unfamiliar with, such as operator overloading and reference arguments, will be explained as they are needed.

1.4 Overview

- **Chapter 1** is the introduction, which you have almost finished reading. Hopefully, it has explained for whom this book is written and why we think you should read the rest of it.
- **Chapter 2** explains the Cartesian coordinate system in 2D and 3D and discusses how the Cartesian coordinate system is used to locate points in space.
- **Chapter 3** discusses examples of coordinate spaces and how they are nested in a hierarchy.
- **Chapter 4** introduces vectors and explains the geometric and mathematical interpretations of vectors.
- **Chapter 5** discusses mathematical operations on vectors and explains the geometric interpretation of each operation.
- **Chapter 6** provides a usable C++ 3D vector class.
- **Chapter 7** introduces matrices from a mathematical and geometric perspective and shows how matrices can be used to perform linear transformations.
- **Chapter 8** discusses different types of linear transformations and their corresponding matrices in detail.
- **Chapter 9** covers a few more interesting and useful properties of matrices.
- **Chapter 10** discusses different techniques for representing orientation and angular displacement in 3D.

- **Chapter 11** provides C++ classes for performing the math from Chapters 7 to 10.
- **Chapter 12** introduces a number of geometric primitives and discusses how to represent and manipulate them mathematically.
- **Chapter 13** presents an assortment of useful tests that can be performed on geometric primitives.
- **Chapter 14** discusses how to store and manipulate triangle meshes and presents a C++ class designed to hold triangle meshes.
- **Chapter 15** is a survey of computer graphics with special emphasis on key mathematical points.
- **Chapter 16** discusses a number of techniques for visibility determination, an important issue in computer graphics.
- **Chapter 17** reminds you to visit our web page and gives some suggestions for further reading.



Chapter 2

The Cartesian Coordinate System

This chapter describes the basic concepts of 3D math. It is divided into three main sections.

- Section 2.1 is about 1D mathematics, the mathematics of counting and measuring. The main concepts introduced are:
  - The math concepts of natural numbers, integers, rational numbers, and real numbers.
  - The relationship between the naturals, integers, rationals and reals on one hand and the programming language concepts of **short**, **int**, **float**, and **double** on the other hand.
  - The First Law of Computer Graphics.
- Section 2.2 introduces 2D Cartesian mathematics, the mathematics of flat surfaces. The main concepts introduced are:
  - The 2D Cartesian plane
  - The origin
  - The *x*- and *y*-axes
  - Orienting the axes in 2D
  - Locating a point in 2D space using Cartesian (*x*,*y*) coordinates
- Section 2.3 extends 2D Cartesian math into 3D. The main concepts introduced are:
  - The *z*-axis
  - The *xy*, *xz*, and *yz* planes
  - Locating a point in 3D space using Cartesian (*x*,*y*,*z*) coordinates
  - Left- and right-handed coordinate systems

3D math is all about measuring locations, distances, and angles precisely and mathematically in 3D space. The most frequently used framework to perform such measurements is called the *Cartesian coordinate system*. Cartesian mathematics was invented by, and named after, a brilliant French philosopher, physicist, physiologist, and mathematician named René Descartes who lived

from 1596 to 1650. Descartes is not just famous for inventing Cartesian mathematics, which at the time was a stunning unification of algebra and geometry. He is also well known for taking a pretty good stab at answering the question "How do I know something is true?" This question has kept generations of philosophers happily employed and does not necessarily involve dead sheep (which will disturbingly be a central feature of the next section), unless you really want it to. Descartes rejected the answers proposed by the ancient Greeks, which are *ethos* (roughly, "because I told you so"), *pathos* ("because it would be nice"), and *logos* ("because it makes sense"), and set about figuring it out for himself with a pencil and paper.

2.1 1D Mathematics

You're reading this book because you want to know about 3D mathematics, so you're probably wondering why we're bothering to talk about 1D math. Well, there are a couple of issues about number systems and counting that we would like to clear up before we get to 3D.

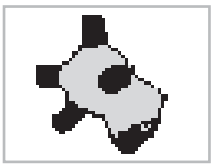


Figure 2.1: One dead sheep

Natural numbers, often called *counting numbers*, were invented millennia ago, probably to keep track of dead sheep. The concept of "one sheep" came easily (see Figure 2.1), then "two sheep" and "three sheep," but people very quickly became convinced that this was too much work. They gave up counting at some point and invariably began using "many sheep." Different cultures gave up at different points, depending on their threshold of boredom. Eventually, civilization expanded to the point where we could afford to have people sitting around thinking about numbers instead of doing more survival-oriented tasks, such as killing sheep and eating them. These savvy thinkers immortalized the concept of zero (no sheep), and while they didn't get around to naming *all* of the natural numbers, they figured out various systems whereby we *could* name them if we really wanted to, using digits such as "1", "2", etc. (or if you were Roman, "M", "X", "I," etc.). Mathematics was born.

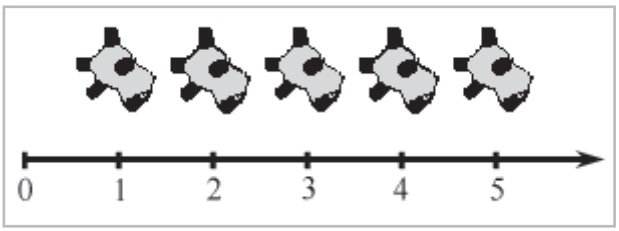


Figure 2.2: A number line for the natural numbers

The habit of lining sheep up in a row so that they can easily be counted leads to the concept of a *number line*, that is, a line with the numbers marked off at regular intervals, as in Figure 2.2. This line can, in principle, go on for as long as we wish, but to avoid boredom we have stopped at five sheep and put on an arrowhead to let you know that the line can continue. Clear thinkers can visualize it going off to infinity, but historical purveyors of dead sheep probably gave this concept little thought, outside of their dreams and fevered imaginings.

At some point in history it was probably realized that you can sometimes, if you are a particularly fast talker, sell a sheep that you don't actually own, thus, simultaneously inventing the important concepts of *debt* and *negative numbers*. Having sold this putative sheep, you would in fact own "negative one" sheep. This would lead to the discovery of *integers*, which consist of the natural numbers and their negative counterparts. The corresponding number line for integers is shown in Figure 2.3.

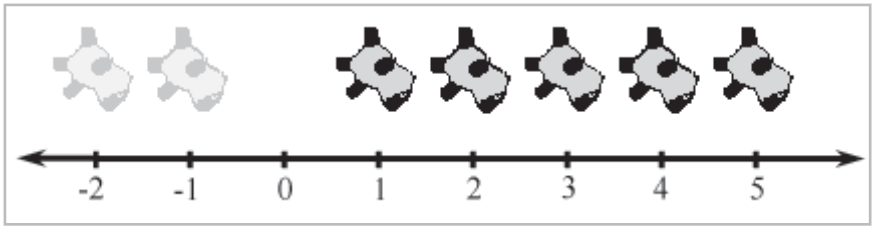


Figure 2.3: A number line for integers (note the ghost sheep for negative numbers)

The concept of poverty probably predated that of debt, leading to a growing number of people who could afford to purchase only half a dead sheep, or perhaps only a quarter. This lead to a burgeoning use of fractional numbers, consisting of one integer divided by another, such as 2/3 or 111/27. Mathematicians called these *rational numbers*, and they fit in the number line in the obvious places between the integers. At some point, people became lazy and invented decimal notation, like writing "3.1415" instead of the longer and more tedious 31415/10000.

After a while, it was noticed that some numbers that appear to turn up in everyday life are not expressible as rational numbers. The classic example is the ratio of the circumference of a circle to its diameter, usually denoted as (pronounced "pi"). These are the so-called *real numbers*, which include rational numbers and numbers such as that would, if expressed in decimal form, require an infinite number of decimal places. The mathematics of real numbers is regarded by many to be the most important area of mathematics, and since it is the basis for most forms of engineering, it can be credited with creating much of modern civilization. The cool thing about real numbers is that while rational numbers are countable (that is, placed into one-to-one correspondence with the natural numbers), real numbers are uncountable. The study of natural numbers and integers is called *discrete mathematics*, and the study of real numbers is called *continuous mathematics*.

The truth is, however, that real numbers are nothing more than a polite fiction. They are a relatively harmless delusion, as any reputable physicist will tell you. The universe seems to be not only discrete, but also finite. If there are a finite amount of discrete things in the universe, as currently appears to be the case, then it follows that we can only count to a certain fixed number. Thereafter, we run out of things to count — not only do we run out of dead sheep, but toasters, mechanics, and telephone sanitizers also. It follows that we can describe the universe using only

discrete mathematics, and requiring the use of only a finite subset of the natural numbers at that. (Large, yes, but finite.) Somewhere there may be an alien civilization with a level of technology exceeding ours that has never heard of continuous mathematics, the Fundamental Theorem of Calculus, or even the concept of infinity; even if we persist, they will firmly but politely insist on having no truck with , being perfectly happy to build toasters, bridges, skyscrapers, mass transit, and starships using 3.14159 (or perhaps 3.14159265358979323846264338327 95, if they are fastidious) instead.

So why do we use continuous mathematics? It is a useful tool that allows us to do engineering, but the real world is, despite the cognitive dissonance involved in using the term "real," discrete. How does that affect you, the designer of a 3D computer-generated virtual reality? The computer is by its very nature discrete and finite, and you are more likely to run into the consequences of the discreteness and finiteness during its creation that you are likely to in the real world. C++ gives you a variety of different number forms that you can use for counting or measuring in your virtual world. These are the **short**, the **int**, the **float** and the **double**, which can be described as follows (assuming the current PC technology). The **short** is a 16-bit integer that can store 65,536 different values, which means that "many sheep" for a 16-bit computer is 65,537. This sounds like a lot of sheep, but it isn't adequate for measuring distances inside any reasonable kind of virtual reality that take people more than a few minutes to explore. The **int** is a 32-bit integer that can store up to 4,294,967,296 different values, which is probably enough for your purposes. The **float** is a 32-bit value that can store a subset of the rationals — 4,294,967,296 of them, the details not being important here. The **double** is similar, though using 64 bits instead of 32. We will return to this discussion in Section 6.3.1.

The bottom line in choosing to count and measure in your virtual world using **int**s, **float**s, or **double**s is not, as some misguided people would have it, a matter of choosing between discrete **short**s and **int**s versus continuous **float**s and **double**s. It is more a matter of precision. They are all discrete in the end. Older books on computer graphics will advise you to use integers because floating-point hardware is slower than integer hardware, but this is no longer the case. So which should you choose? At this point, it is probably best to introduce you to the First Law of Computer Graphics and leave you to think about it:

The First Law of Computer Graphics: If it looks right, it *is* right.

We will be doing a large amount of trigonometry in this book. Trigonometry involves real numbers, such as , and real-valued functions, such as sine and cosine (which we'll get to later). Real numbers are a convenient fiction, so we will continue to use them. How do you know this is true? You know because, Descartes notwithstanding, we told you so, because it would be nice, and because it makes sense.

2.2 2D Cartesian Mathematics

You have probably used 2D Cartesian coordinate systems even if you have never heard the term *Cartesian* before. *Cartesian* is mostly just a fancy word for rectangular. If you have ever looked at the floor plans of a house, used a street map, seen a football game, or played chess, you have been exposed to 2D Cartesian coordinate space.

2.2.1 An Example: The Hypothetical City of Cartesia

Let's imagine a fictional city named Cartesia. When the Cartesia city planners were laying out the streets, they were very particular, as illustrated in the map of Cartesia in Figure 2.4.

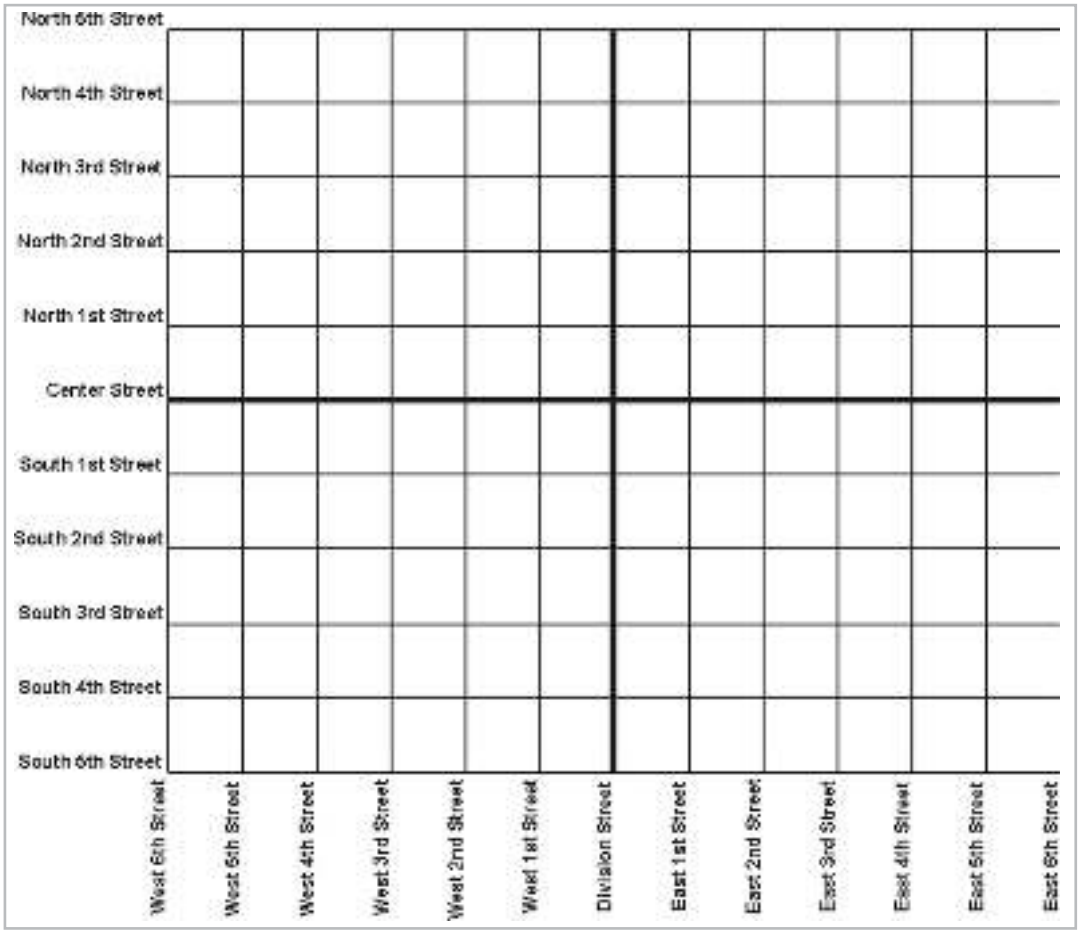


Figure 2.4: Map of the hypothetical city of Cartesia

As you can see from the map, Center Street runs east-west through the middle of town. All other east-west streets (parallel to Center Street) are named based on whether they are north or south of Center Street and how far away they are from Center Street. Examples of streets which run east-west are North 3rd Street and South 15th Street.

The other streets in Cartesia run north-south. Division Street runs north-south through the middle of town. All other north-south streets (parallel to Division Street) are named based on whether they are east or west of Division street, and how far away they are from Division Street. So we have streets such as East 5th Street and West 22nd Street.

The naming convention used by the city planners of Cartesia may not be creative, but it certainly is practical. Even without looking at the map, it is easy to find the doughnut shop at North 4th and West 2nd. It's also easy to determine how far you will have to drive when traveling from one place to another. For example, to go from that doughtnut shop at North 4th and West 2nd to the police station at South 3rd and Division, you would travel seven blocks south and two blocks east.

2.2.2 Arbitrary 2D Coordinate Spaces

Before Cartesia was built, there was nothing but a large flat area of land. The city planners arbitrarily decided where the center of town would be, which direction to make the roads run, how far apart to space the roads, etc. Much like the Cartesia city planners laid down the city streets, we can establish a 2D Cartesian coordinate system anywhere we want — on a piece of paper, a chessboard, a chalkboard, a slab of concrete, or a football field.

Figure 2.5 shows a diagram of a 2D Cartesian coordinate system.

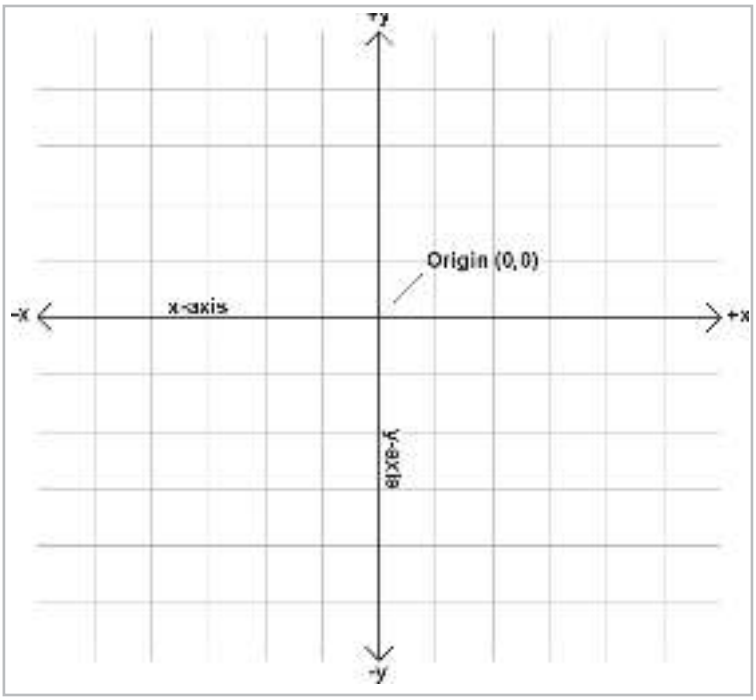


Figure 2.5: A 2D Cartesian coordinate space

As illustrated in Figure 2.5, a 2D Cartesian coordinate space is defined by two pieces of information:

- Every 2D Cartesian coordinate space has a special location, called the *origin*, which is the "center" of the coordinate system. The origin is analogous to the center of the city in Cartesia.
- Every 2D Cartesian coordinate space has two straight lines that pass through the origin. Each line is known as an *axis* and extends infinitely in two opposite directions. The two axes are perpendicular to each other. (Actually, they don't *have* to be, but most of the coordinate systems we will look at will have perpendicular axes.) The two axes are analogous to Center and Division streets in Cartesia. The grid lines in the diagram are analogous to the other streets in Cartesia.

At this point, it is important to highlight a few significant differences between Cartesia and an abstract mathematical 2D space:

- The city of Cartesia has official city limits. Land outside of the city limits is not considered part of Cartesia. A 2D coordinate space, however, extends infinitely. Even though we usually only concern ourselves with a small area within the plane defined by the coordinate space, this plane, in theory, is boundless. In addition, the roads in Cartesia only go a certain distance (perhaps to the city limits), and then they stop. Our axes and grid lines, on the other hand, each extend potentially infinitely in two directions.
- In Cartesia, the roads have thickness. Lines in an abstract coordinate space have location and (possibly infinite) length, but no real thickness.
- In Cartesia, you can only drive on the roads. In an abstract coordinate space, *every* point in the plane of the coordinate space is part of the coordinate space, not just the area on the "roads." The grid lines are only drawn for reference.

In Figure 2.5, the horizontal axis is called the x-axis, with positive x pointing to the right. The vertical axis is the y-axis, with positive y pointing up. This is the customary orientation for the axes in a diagram. Note that "horizontal" and "vertical" are terms that are inappropriate for many 2D spaces that arise in practice. For example, imagine the coordinate space on top of a desk — both axes are "horizontal," and neither axis is really "vertical."

The city planners of Cartesia could have made Center Street run north-south instead of east-west. Or they could have placed it at a completely arbitrary angle. (For example, Long Island, New York is reminiscent of Cartesia, where for convenience the streets numbered "1st Street," "2nd Street," etc., run across the island and the avenues numbered "1st Avenue," "2nd Avenue," etc., run along its long axis. The geographic orientation of the long axis of the island is an arbitrary freak of nature.) In the same way, we are free to place our axes in any way that is convenient to us. We must also decide for each axis which direction we consider to be positive. For example, when working with images on a computer screen, it is customary to use the coordinate system shown in Figure 2.6. Notice that the origin is in the upper left-hand corner, $+x$ points to the right, and $+y$ points *down* rather than up.