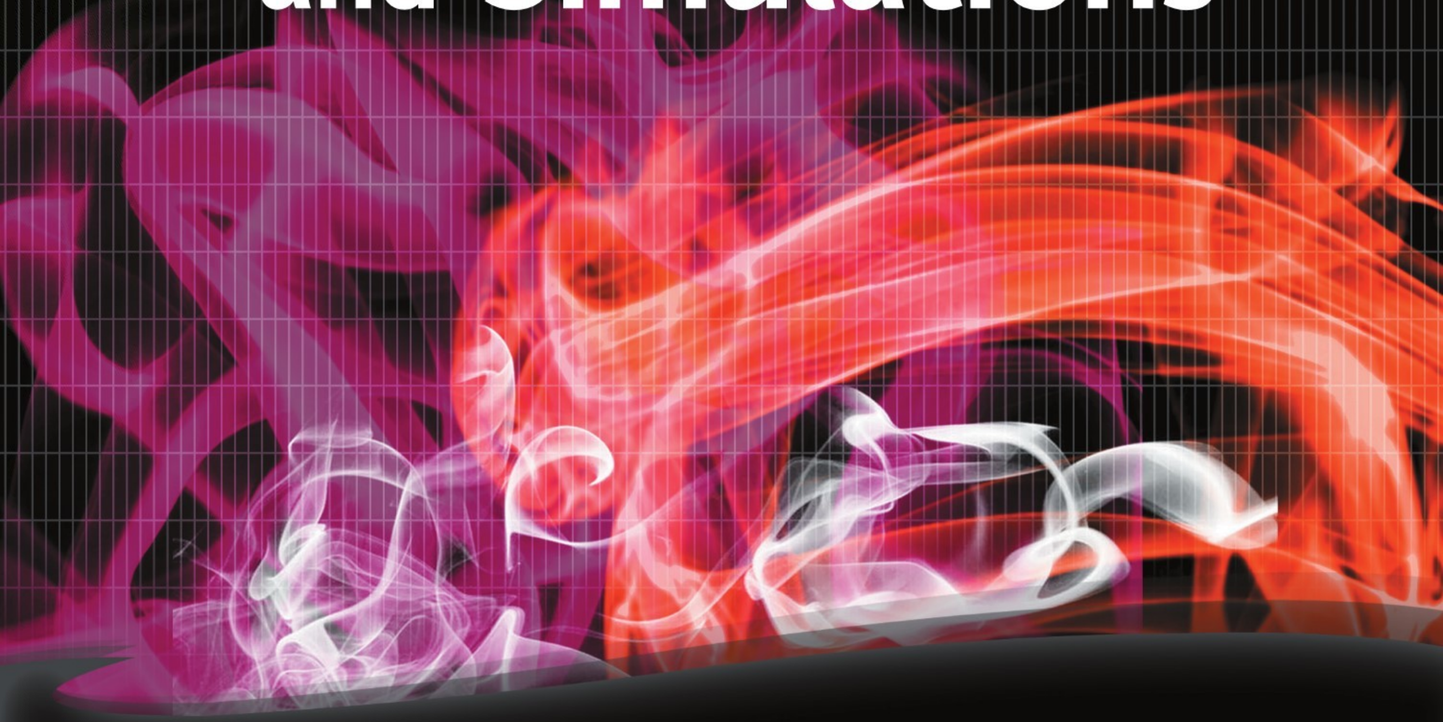
THE ESSENTIAL GUIDE TO

# Physics for Flash Games, Animation, and Simulations



DEV RAMTAL AND
ADRIAN DOBRE

friendsof™
an Apress® company

# The Essential Guide to Physics for Flash Games, Animation, and Simulations

Dev Ramtal
Adrian Dobre

friendsof ™
DESIGNER TO DESIGNER™
an Apress® company

# The Essential Guide to Physics for Flash Games, Animation, and Simulations

ISBN-13 (pbk): 978-1-4302-3674-0

ISBN-13 (electronic): 978-1-4302-3675-7

## Credits

**President and Publisher:**
Paul Manning

**Lead Editor:**
Chris Nelson

**Technical Reviewer:**
RJ Owen

**Editorial Board:**
Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Jonathan Gennick, Jonathan Hassell, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Frank Pohlmann, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh

**Coordinating Editor:**
Corbin Collins

**Copy Editor:**
Nancy Sixsmith

**Compositor:**
Apress Production (Christine Ricketts)

**Indexer:**
SPI Global

**Artist:**
SPI Global

**Cover Artist:**
Anna Ishchenko

**Cover Designer:**
Corné van Dooren

*To my mother, who made me possible; and to my wife and daughter, who supported me in making this book possible.*

*–DR*

*To my family who have been so understanding while I have been away from them working on this book.*

*–AD*

# Contents at a Glance

# Contents

# About the Authors

**Dev Ramtal** has been coding physics for more than 20 years. Alongside his academic research background in mathematical and computational physics, he has a long history of web development experience since building his first website back in 1997. Dev has been working with the Flash platform for the past six years, using it as a research and teaching tool, for general web programming, and just for some serious fun. He is also increasingly interested in game programming. Dev has a BSc and a PhD in Physics from the University of London. He lives in the UK where he works as a research scientist in academia. He can be contacted at `www.physicscodes.com`.

**Adrian Dobre** has more than 15 years of experimental and computational modeling experience in fluid dynamics. His experience in scientific research and teaching brought his attention to ActionScript and Flash, where programming combines with nice visual output. From here his dedication over more than five years for using Actionscript as a tool to model physical processes and create virtual laboratory platforms. Adrian holds a BSc in Aeronautical Engineering from University *Politehnica* in Bucharest, Romania and a PhD in Engineering Science from the University of Western Ontario, Canada. He currently lives in Bucharest, Romania, with his family.

# About the Technical Reviewer

**RJ Owen** is a senior software architect at Effective UI. RJ has been working with Flex since version 2 and has worked on many projects of varying sizes and complexities, but always favors those with great user interfaces. He is a regular blogger and has spoken at many events including 360|Flex, Adobe MAX, and O'Reilly's Web 2.0. RJ enjoys advocating for teaching developers how to design and branch into skills beyond software. He has an MBA degree in Physics and Computer Science and lives in Colorado with his family.

# About the Cover Image Artist

**Corné van Dooren** designed the front cover image for this book. After taking a brief hiatus from friends of ED to create a new design for the Foundation series, he worked at combining technological and organic forms, the results of which now appear on this and other book covers.

Corné spent his childhood drawing on everything at hand and then began exploring the infinite world of multimedia—and his journey of discovery hasn't stopped since. His mantra has always been, "The only limit to multimedia is the imagination"—a saying that keeps him constantly moving forward.

Corné works for many international clients, writes features for multimedia magazines, reviews and tests software, authors multimedia studies, and works on many other friends of ED books. You can see more of his work at and contact him through his website at `www.cornevandooren.com`.

# Acknowledgments

We are extremely grateful to all the individuals, at Apress and elsewhere, who have helped turn this book from an idea into physical reality. An enormous amount of work goes into producing a book, of which writing it is only one part. We were fortunate to have the support of such a fantastic team—we won't mention any names for fear of missing anyone.

We are also grateful to the Flash and programming community for sharing their knowledge and expertise so openly through blogs, forums, websites, and books. It is a great pleasure to be able to give something back after having gained so much from the community.

Finally, we would like to thank our respective families for their constant support and patience during the intense period of writing this book.

# Preface

We hope you will enjoy reading this book as much as we enjoyed writing it. Doing physics with Flash is a lot of fun—there is a sense of satisfaction in writing a few lines of code and then seeing how they make objects behave like they do in the real world. The ability to do this gives us great power as programmers. The aim of this book is to provide you with tools that will make you feel that sense of power. Whether you want to build convincing animations, games that behave realistically, or accurate simulators, you will find herein plenty of tools, examples and ideas that should be of help.

The inspiration for this book came from the excellent book authored by Keith Peters, *Foundation ActionScript Animation*. This was the first book we read when we were still getting acquainted with Flash and ActionScript several years ago. The book contained a good amount of physics-related topics and showed us the power of Flash in creating interactive animations with physics that were as fun to code as they were to play with. Since then we became addicted to Flash and ActionScript, using them to create numerous physics-based simulations and other applications. Along the way we learned or figured out a lot of things that seemed worth sharing with the Flash community. Keith Peters's book provided a great introduction to physics-based animation. But it seemed to us that there was also a need for a book that explored physics further and in greater depth, and that catered for more demanding applications such as accurate simulators or more complex game programming. After all, there are several "game physics" books written for other programming languages such as C++ and Java, so why not ActionScript? The present book is meant to fill this gap.

We should make it clear at the outset that this is primarily a book about physics, and secondarily about Flash. Therefore, the focus is less on producing attractive animations, and more on modeling real physics using the Flash environment. Some of the simulations in the book may not be very pretty or smooth from a visual perspective, but they do contain a lot of physics goodness! The approach we adopt attempts to make serious physics accessible. But although we don't shy away from going into technical stuff, with all the accompanying math, we hope to have done so in a way that is simple and straightforward.

Inevitably, due to space and time restrictions, there are topics that we were not able to include or that we didn't explore in detail. Nevertheless, the book covers an awful lot of ground, taking you from coding a simple bouncing ball animation in a few lines of code in the first chapter to a highly accurate simulation of the solar system in the final chapter. We hope you enjoy the journey in between!

## What this book will (and won't) teach you

*The Essential Guide to Physics for Flash Games, Animation, and Simulations* teaches you how to include physics into your programming. It does not teach you how to program. It does not teach you ActionScript. And though this book teaches you how to implement physics into your games (as well as other projects), it is not about game programming *per se*.

We assume that you have at least some programming experience with ActionScript 3.0 (AS 3.0) as well as familiarity with an authoring tool for building Flash applications in AS 3.0, such as Adobe Flash (version CS3 or above) or Adobe Flash Builder (formerly Adobe Flex Builder). This book does not teach you how to use those authoring tools. If you don't have this background we highly recommend picking up one of the many excellent introductory books on Flash and ActionScript available, such as *Foundation ActionScript 3.0 for Flash and Flex* by Darren Richardson and Paul Milbourne. We also highly recommend *Foundation ActionScript 3.0 Animation: Making Things Move!* by Keith Peters.

We do not assume any previous knowledge of physics, and only assume basic school level math knowledge. One chapter is dedicated to explaining some of the more difficult math concepts and tools that you are likely to need in the book.  All the physics concepts and knowledge you will need are explained in a self-contained way. Numerous applications are given throughout the book with full source code to illustrate the application of the principles learnt.

## Overview of this book

This book is divided into four parts.

Part I: "The Basics" (Chapters 1-4) introduces the necessary background in basic math and physics concepts upon which the rest of the book builds. For completeness, it also covers selected topics in ActionScript 3.0 that are most pertinent to physics programming.

Part II: "Particles, Forces, and Motion" (Chapters 5-10) begins by formulating the laws of physics that govern the motion of particles under any type of force (Chapter 5). The next chapters then apply those laws to make objects move under the action of a large variety of forces including gravity, friction, drag, buoyancy, wind, springs, central forces, and many more. The main focus in this section is on simulating the motion of individual particles and other simple objects.

In Part III: "Multi-Particle and Extended Systems" (Chapters 11-13), we show you how to model more complicated systems, including multiple interacting particles and extended objects. In these systems the constituent particles and objects do not simply co-exist but interact with each other, mutually influencing their motion. These interactions include collisions, short-range forces, and long-range forces. This part includes a discussion of particle systems, rigid bodies, and deformable bodies.

Part IV: "Building More Complex Simulations" (Chapters 14-16) is devoted to building more complex simulations, where accuracy and/or realism is especially important, not just visual effects. This part includes a discussion of numerical integration schemes and other numerical and technical issues such as scale modeling and 3D. Part IV ends with a chapter including some example simulation projects.

## Source code and examples

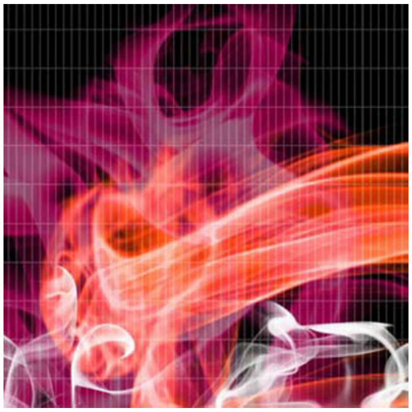
All the code for this book can be downloaded from the book's page on the Apress website: `www.apress.com`.

Source code in the book will work with Adobe Flash (version CS3 onwards; a few examples require CS4) and Adobe Flex/Flash Builder (version 3 onwards). All the code is pure ActionScript. We do not use MXML or the Flex framework.

We encourage you to modify and build upon the example codes given. While doing the technical review of this book, RJ Owen suggested it would be interesting to have a website where readers could share their codes to see what people can build with the tools given in this book. We thought it was an excellent suggestion, and have therefore set up the following page on our website where you can find and contribute more code and examples: www.physicscodes.com/as3book.

# Part I

# The Basics



Chapter 1

# Introduction to Physics Programming

Because you picked up this book, we assume that you are interested in implementing physics into your programming projects. But why would you want to do that? What can it do for you? And how difficult will it be? This chapter will provide answers to these questions.

Topics covered in this chapter include:

- **Why model physics?** This section will explain some of the reasons why you might want to add physics to your projects.
- **What is physics?** Here we lift the veil of mystery and explain in simple terms what physics is. We also tell you, in a nutshell, what you'll need to know.
- **Programming physics.** Thankfully, programming physics is not as difficult as you might imagine, once you understand some basic principles. This section explains what you'll need to do.
- **A simple example.** As a concrete example, we'll code up a simple animation involving physics, using a minimum of code.

## Why model real physics?

There are a number of reasons why you might be interested in modeling physics using Flash. Here are some of the most common reasons.

## Creating realistic animation effects

One of the main strengths of working with Flash is the ability to create animations easily. With a little ActionScript and some physics, it is also possible to make animations that look and behave like the real thing. For example, suppose you are animating a scene in which someone kicks a ball and it bounces off the ground. If you're working with Adobe Flash, you could try and fake the animation with tweens, but however hard you might try it would probably look less than realistic. With just a little bit of coding and some knowledge of simple physics you could produce a far more realistic animation. And if, like the authors, you are programmers rather than designers, you might even find it easier! We'll show you just how easy it can be in the example at the end of this chapter.

## Creating realistic games

Flash games are ever so popular on the Web. As the capabilities of the Flash Player continue to improve, better and more powerful games can be built. Hardware acceleration and native 3D support are just two of the recent developments that have the potential to improve the gaming user experience dramatically. But apart from performance and appearance, it is equally important for games to feel realistic. If you are throwing a ball, it has to fall according to the law of gravity; if you fire a torpedo underwater, it must move differently from a ball falling in air. In other words, your game needs to know physics.

How do you build physics awareness into your games? This book will show you how.

## Building simulations and models

A *computer simulation* or *computer model* is a program that attempts to imitate certain key aspects of a system. Simulations vary in completeness or accuracy, depending on purpose and resources. Let's take a flight simulator program as an example. One would expect a flight simulator designed for training pilots to be much more comprehensive and accurate than one designed for a game. Simulations are extremely common in e-learning, training and in scientific research. In the final chapter of this book, you'll build simulations, including a submarine, a basic flight simulator, and a model of the solar system. In fact, many of the coded examples throughout the book are simulations, even if generally simpler.

## Generating art from code

Generative art has gained popularity in recent years. ActionScript and Processing are popular tools used by artists to generate art from code. A lot of fun can be had with some simple physics—for example, elaborate effects and motions can be produced using particles and different kinds of forces.

We will explore the world of generative art and provide additional tools and algorithms that can be used to create original and interesting effects.

## Can’t I just use a physics engine?

You probably can. Physics engines are certainly useful for a lot of situations. But they may not necessarily be the panacea for all your problems. Here are a few reasons why the time you invest in learning how to program physics can pay off:

- You’ll have more flexibility to do what you want. Physics engines cannot do everything.
- In some cases, it may be simpler to write code from scratch rather than to use a physics engine.
- Knowing how to use a physics engine doesn’t guarantee that you’ll know what physics to use for a particular application.
- Sometimes things may not work quite as they should with a physics engine. For example, problems may arise due to numerical instability or inaccuracy in some situations.
- You’ll be able to build your own physics engine, matched to your own needs.
- It’s fun!

The bottom line, though, is that it should not be an either/or lifetime decision whether to choose between a physics engine and coding your own physics. Experiment with both, and you’ll become better at both: being able to code up your own physics will give you a deeper understanding of how physics engines work and how to use them more effectively; conversely, experimenting with physics engines should give you plenty of ideas that you can implement into your own coding.

There are now a number of Flash physics engines out there. Here is a brief list of some of them:

- **Box2DFlash**: Also known as Box2DAS3, this is an ActionScript 3.0 (AS3.0) port of Box2D, an open source 2D physics engine written by Erin Catto in C++. Box2DFlash is primarily a rigid body library, but with a host of other features. It is a popular choice among Flash game developers. Because Box2DFlash closely resembles Box2D, users can get help from the Box2D forums.
- **FOAM**: Another 2D rigid body physics engine. In the words of its creator, FOAM trades efficiency for modularity and extensibility, allowing a savvy developer to extend and repurpose FOAM to his own ends. FOAM is released under the MIT license.
- **Fisix**: The Fisix engine is an AS3.0 verlet-based physics engine aimed for use in games and relatively CPU-intensive applications. The website pledges to give good documentation and help, and states that the engine is currently free for both commercial and noncommercial applications until the next release.
- **APE**: APE stands for *Actionscript Physics Engine*, another AS3.0 open source 2D physics engine written by Alec Cove and released under the MIT license.
- **WOW-Engine**: This is an AS3.0 open source 3D physics engine written by Jerome Birembaut. It uses the Sandy library for all its 3D math computations and handles collisions and physics by extending APE, the 2D physics engine created by Alec Cove (see the previous bullet).

- **JiglibFlash**: This is another AS3.0 open source 3D physics engine, ported from the C++ physics engine Jiglib. The engine has support for Away3D, Papervision3D and Sandy3D. Documentation appears sparse, but it seems to have an active forum.

*A disclaimer: The selection and ordering of the list are essentially random and do not constitute a recommendation to use any of them in preference to any other.*

## What is physics?

*Physics* is the most fundamental of the sciences. In a broad sense, physics is the study of the natural laws that govern how things behave. More specifically, it concerns itself with space, time and matter (defined as any “stuff” that exists in space and time). One aspect of physics is to formulate general laws that govern the behavior of matter, its interactions, and its motion in space and time. Another aspect is to use these laws to make predictions of the way specific things move and interact—for example, the prediction of eclipses from the laws of gravity or how airplanes are able to fly from the laws of aerodynamics.

Physics is a vast subject, and in a book of this nature we cannot do more than scratch the surface. Fortunately, most of the physics that you will probably need to know fall within a branch known as *Mechanics*, which is one of the easiest to understand.

### Everything behaves according to the laws of physics

Without getting too philosophical, it is fair to say that the laws of physics are truly universal, as far as physicists have been able to observe. What this means is that everything *must* behave according to physics. This is different from say, the laws of biology, which pertain only to living things. A stone thrown in the air, a planet orbiting the Sun, the workings of the human body, and the operation and motion of a man-made machine must all obey the laws of physics. Moreover, many seemingly diverse phenomena are governed by the same subset of laws. For example, a falling stone and a planet orbiting the Sun both obey the laws of gravity.

### The laws can be written as math equations

The great thing is that the laws of physics can be written as mathematical equations. Okay, that may not sound too great if you don’t like math! But the point here is that for a law to be useful, it has to be made precise. And math equations are as precise as anything can be. There is no possible ambiguity in how to apply a law that is expressed mathematically, in contrast with the laws that are fought over in courtrooms! Second, this means that centuries of developments in mathematics prove to be applicable to physics, making it possible to solve many physics problems. Third, and what is of most relevance for us: math equations are readily convertible into code.

## Predicting motion

Let's get more specific. As an ActionScript programmer, you are mostly interested in how things move. Much of physics deals with how things move under the action of different types of influences. These "influences" can be from other things and from the environment. In physics we have a special name for these influences: they are called *forces*. The really good news is that the forces have simple mathematical forms. Although the motion of objects is usually complicated, the underlying mathematical laws that describe the forces are usually quite simple.

The general relationship between force and motion can be written schematically as follows:

**motion =** function{**forces**}

This is a cause-and-effect relationship. Forces cause objects to move in different ways. In practical terms, it means this: specify the forces acting on an object and put them in a mathematical equation, and then you can calculate the motion of the object. Simple isn't it?

*Motion is effect. Force is cause. The motion of an object is the result of the forces acting on it. The mathematical relationship between force and motion is known as the "Law of Motion."*

Now to be able to put this principle to use, we need to know the following:

- **Definitions**. The precise definitions of *motion* and *force*.
- **The law of motion**. In other words, the precise mathematical form of the function that relates a force to the motion it produces.
- **Force laws**. In other words, how to calculate the forces. There are equations that tell you how to calculate each type of force.

So there are two kinds of laws you need to know about: laws of motion and force laws. You will also need to know the proper concepts (known as *physical quantities*) to describe and analyze motion and forces and the relationship between them. Finally, you will need to know the mathematics for manipulating and combining these quantities.

## Programming physics

So, how do you code up physics? Do you program the motion or forces, or both? And what does it involve?

Once you know some basic physics (and some simple math), coding it is not much different or more difficult than what you are used to as a programmer—provided you do it in the right way. Let us take some time to explain what this "right way" is.