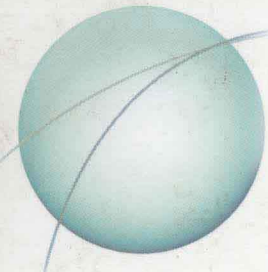

SECOND EDITION

GAME PHYSICS

DAVID H. EBERLY

MORGAN KAUFMANN

GAME PHYSICS

SECOND EDITION

DAVID H. EBERLY

常州大学图书馆
藏书章

ELSEVIER

AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Morgan Kaufmann Publishers is an imprint of Elsevier

MK®
MORGAN
KAUFMANN

Morgan Kaufmann is an imprint of Elsevier
30 Corporate Drive, Suite 400, Burlington, MA 01803, USA
The Boulevard, Langford Lane, Kidlington, Oxford, OX5 1GB, UK

Copyright © 2010, Elsevier Inc. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or any information storage and retrieval system, without permission in writing from the publisher. Details on how to seek permission, further information about the Publisher's permissions policies and our arrangements with organizations such as the Copyright Clearance Center and the Copyright Licensing Agency, can be found at our website: *www.elsevier.com/permissions.*

This book and the individual contributions contained in it are protected under copyright by the Publisher (other than as may be noted herein).

Notices

Knowledge and best practice in this field are constantly changing. As new research and experience broaden our understanding, changes in research methods, professional practices, or medical treatment may become necessary.

Practitioners and researchers must always rely on their own experience and knowledge in evaluating and using any information, methods, compounds, or experiments described herein. In using such information or methods they should be mindful of their own safety and the safety of others, including parties for whom they have a professional responsibility.

To the fullest extent of the law, neither the Publisher nor the authors, contributors, or editors assume any liability for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions, or ideas contained in the material herein.

Library of Congress Cataloging-in-Publication Data
Eberly, David H.
Game physics / David H. Eberly. – 2nd ed.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-12-374903-1 (hardcover : alk. paper)
1. Computer games–Programming. 2. Physics–Computer simulation. 3. Computer graphics.
4. Three-dimensional display systems. I. Title.
QA76.76.C672E24 2010
794.8'1526–dc22
2009049828

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library.

ISBN: 978-0-12-374903-1

For information on all Morgan Kaufmann publications
visit our Web site at *www.elsevierdirect.com*

Typeset by: diacriTech, India

Printed in the United States of America
10 11 12 13 10 9 8 7 6 5 4 3 2 1

Working together to grow
libraries in developing countries
www.elsevier.com | www.bookaid.org | www.sabre.org
ELSEVIER BOOK AID International Sabre Foundation

CONTENTS OF THE CD-ROM

The CD-ROM contains the full distribution of Wild Magic 5.0, including source code files and data files. The directory hierarchy of library source files is described here.

CORE LIBRARY

ASSERT. Support for handling assertions in the code.

DATATYPES. Template classes for tuples of objects and minimum heaps.

INPUTOUTPUT. Buffer and file input and output, including handling native byte ordering. Environment variable and path support.

MEMORY. A memory manager for tracking memory usage and memory leaks. A smart pointer implementation for singleton objects and for arrays of objects.

OBJECTSYSTEMS. Pre-main initialization and post-main termination semantics. Base object class that supports run-time type information, object naming, and streaming (serialization and deserialization) of objects.

THREADING. Basic support for mutual exclusion (mutexes) and multithreading.

TIME. Support for querying the current time.

MATHEMATICS LIBRARY

ALGEBRA. Vector, matrix, quaternion, and polynomial classes.

APPROXIMATION. Fitting of point sets with Gaussian distributions, lines, planes, quadratic curves, quadric surfaces, and polynomials.

BASE. Basic mathematics support. Convenient floating-point tuple classes.

COMPUTATIONALGEOMETRY. Convex hulls (2D and 3D), Delaunay triangulation (2D and 3D), polygon triangulation.

CONTAINMENT. Containment of point sets by various geometric objects. Point-in-polygon and point-in-polyhedron tests, separation of points. Containment by minimum-area rectangles and circles, and by minimum-volume boxes and spheres.

CURVESSURFACESVOLUMES. Curve classes (2D and 3D), surface classes. B-spline curves, surfaces, and volume. B-spline fitting algorithms for curves and surfaces. Geodesic paths on surfaces.

DISTANCE. Distance between pairs of geometric objects (2D and 3D algorithms).

INTERPOLATION. Various algorithms for interpolating data (2D and 3D).

INTERSECTION. Intersection testing and finding for pairs of geometric objects (2D and 3D algorithms).

MESHES. Various implementations of vertex-edge-triangle graphs.

MISCELLANEOUS. Algorithms for compression of normal vectors, perspective projection of ellipsoids to ellipses, mappings between convex quadrilaterals (quadratic and perspective), generation of random points on hyperspheres, computing tangents to pairs of circles.

NUMERICALANALYSIS. Root finding, eigensolvers for symmetric matrices, integration, linear system solving, minimization without derivative calculations, differential equation solvers, singular value decomposition.

OBJECTS2D. Definitions of standard 2D objects that occur in the Wild Magic library.

OBJECTS3D. Definitions of standard 3D objects that occur in the Wild Magic library.

QUERY. Standard queries to support computational geometry, implemented for floating-point arithmetic, for exact rational arithmetic, and for filtered predicates.

RATIONAL. An implementation of exact rational arithmetic.

GRAPHICS LIBRARY

CONTROLLERS. A controller system for keyframe animation, vertex-based morphing, skinning, inverse kinematics, and point and particle systems.

CURVESSURFACES. Classes for curves and surfaces that support tessellation at run time. Special surface classes include B-spline surfaces, rectangle surfaces, revolution surfaces, box surfaces, and tube surfaces.

DATATYPES. Classes for bounding volumes, transformations, specialized buffer and file input-output for Wild Magic classes, color conversions, and 16-bit floating-point representations.

DETAIL. Level of detail including billboards, switch nodes, discrete level of detail, and continuous level of detail.

GLOBALEFFECTS. Base class for effects that apply globally to a scene (more required than just shader programs). Examples include planar shadows and planar reflections.

IMAGEPROCESSING. Support for postprocessing effects that require drawing to render targets. Includes support for 2D effects and for 3D effects (such as fluid dynamics).

此为试读,需要完整PDF请访问: www.ertongbook.com

GAME PHYSICS

SECOND EDITION

DEDICATION

After surgery for Stage 4 gallbladder cancer, aggressive chemotherapy, and radiation treatments, it was not clear I would ever have a chance to write the second edition of this book. My survival is a testament to the advances that modern medicine has made. The unselfish caregiving by my loving wife, Shelly, her support through difficult times, and her encouragement for me to recover and continue writing are things for which I am extremely grateful. I have finished this book and look forward to writing many more with her at my side.

TRADEMARKS

The following trademarks, mentioned in this book and the accompanying CD-ROM, are the property of the following organizations.

- DirectX, Direct3D, Visual C++, DOS, Windows, and Xbox 360 are trademarks of Microsoft Corporation.
- Playstation 3 is a trademark of Sony Corporation.
- OpenGL is a trademark of Silicon Graphics, Inc.
- Radeon is a trademark of ATI Technologies, Inc.
- NVIDIA, GeForce, PhysX, and the Cg Language are trademarks of NVIDIA Corporation.
- NetImmerse and R-Plus are trademarks of Numerical Design, Ltd.
- MathEngine is a trademark of Criterion Studios.
- The Havok physics engine is a trademark of Havok.com Inc.
- SoftImage is a trademark of Avid Technology, Inc.
- Prince of Persia 3D is a trademark of Brøderbund Software, Inc.
- XS-G and Canyon Runner are trademarks of Greystone Technology.
- Mathematica is a trademark of Wolfram Research, Inc.
- Turbo Pascal is a trademark of Borland Software Corporation.
- The 8086 and Pentium are trademarks of Intel Corporation.
- Macintosh and Xcode are trademarks of Apple Corporation.
- Gigi and VAX are trademarks of Digital Equipment Corporation.
- MASPAR is a trademark of MasPar Computer Corporation.

Preface to the Second Edition

The first edition of *Game Physics* appeared in December 2003. At that time physics was quite a popular topic in games – it still is. I had mentioned in the introductory chapter of the first edition that we should expect physics hardware soon. In fact, Ageia Technologies produced their *physics processing unit* (PPU), but since then, NVIDIA Corporation acquired the company and provides PhysX® to access the physics hardware. However, some physics computations can even be performed on the graphics processing unit (GPU). And with all the power of multiple processors and cores on current generation desktop computers and game consoles, highly performing complex physical simulations are becoming a reality at minimal cost to the consumer.

A lot of research has been done over the years, so much so that it is difficult to keep up with and determine which are the important results and which are not. In this, the second edition, I have added some new material to address some of the criticisms about the first edition. The main criticism was that the impulse-based approach I discussed in the physics engine chapter is not what most commercial or Open Source engines use. The Wild Magic physics engine did (and does) contain a global LCP solver, and there was (and is) a sample application that illustrates its use. However, the number of interacting objects is small. The global solver is not suitable for a large number of objects. I had not discussed iterative dynamics in the first edition, but have remedied that in the second edition with a new section that describes the more common velocity-based dynamics. I had planned on adding a section on position-based dynamics, but the papers of interest to me are very dense in content. I prefer spending a lot of time dissecting the methods and researching the references of those papers before I am comfortable writing about them. Position-based dynamics will have to wait until a third edition.

The second edition also contains a lengthy chapter about fluid dynamics, both in 2D and 3D. The vector calculus background is extensive, and the goal is to derive the partial differential equations of motion from conservation of mass (density equation) and conservation of momentum (Navier–Stokes equation for velocity). Solving these equations on moderate-sized grids still takes a large number of CPU cycles, so much so that it would not be correct to say they are real-time algorithms. However, with multiple cores and processors these days, it is possible to obtain real-time rates. In fact, the source code on the CD-ROM contains a GPU-based implementation for a 2D Navier–Stokes solver that runs in real time. I have provided the framework in the Wild Magic libraries to solve the 3D problem on a GPU. A sample application that uses the library involves Gaussian blurring of a 3D image. I leave it as a project to implement the 3D fluid solver on top of this framework, although I am certain I will post my own implementation at some time after this book is released. The framework

is where most of the work is done anyway. The application work involves writing the shaders and setting up all the render targets necessary to run the application with a minimum amount of video memory and a minimum of transfer of data from system memory to video memory. This work is at a high level and relatively easy to do.

The broad-phase collision culling using axis-aligned boxes and space-time coherency was a topic in the first edition, and there was an implementation for the CPU on the CD-ROM. I greatly expanded this section to discuss how to move the broad-phase culling onto its own CPU core and use multithreading to support it. This is useful when programming on a Microsoft Xbox 360. I also added a discussion on how to move the broad-phase culling onto a specialized processor such as an SPU of the Sony Playstation 3. I have implemented this on the aforementioned consoles, and it was absolutely essential for good performance in a physics-heavy racing game with lots of interacting objects.

Of course, books have bugs just as source code does. The second edition contains as many corrections to typographical errors and content errors as I could possibly find.

The CD-ROM for the book contains a new version of Wild Magic, version 5.0. This is a significant rewrite of Wild Magic 4, except for the mathematics library (some ancient topics never change). I have greatly modified the graphics engine to have its own general FX system. Adding specialized shaders in Wild Magic 4 was a bit tedious and not general. The same thing in Wild Magic 5 is straightforward – the engine design for managing renderer resources does not get in your way. For example, the rewrite made it easy to implement the fluid solvers that are part of the sample applications. Perhaps superficial yet wanted by many users, I have eliminated the verbose Hungarian-style variable naming conventions (and there was great rejoicing ...). Finally, the Wild Magic 5 engine uses the Boost License, which is as liberal as you can get. I hope you enjoy playing with the new version.

As always, my thanks go to the Elsevier and Focal Press folks for their diligence and hard work to make this book possible: Laura Lewin (Senior Acquisitions Editor), Chris Simpson (Associate Acquisitions Editor), Anaïs Wheeler (Assistant Editor), Julie Ochs (Project Manager), and the entire production team at Elsevier.

Preface to the First Edition

The evolution of the games industry has been motivated clearly by the gamers' demands for more realistic environments. 3D graphics on a 2D graphics card requires necessarily a classical software renderer. Historically, rasterization of triangles was the bottleneck on 2D cards because of the low *fill rate*, the rate at which you can draw pixels during rasterization. To overcome fill rate limitations on consumer cards the *graphics hardware accelerator* was born in order to off-load the rasterization from the 2D card and the *central processing unit* (CPU) to the accelerator. Later generations of graphics cards, called 3D graphics cards, took on the role of handling the standard work of a 2D graphics card (drawing windows, bitmaps, icons, etc.) as well as supporting rasterization that the 3D graphics requires. In this sense the adjective accelerator for a combined 2D/3D card is perhaps a misnomer, but the term remains in use.

As fill rates increased, the complexity of models increased, further driving the evolution of graphics cards. Frame buffer and texture memory sizes increased in order to satisfy the gamers' endless desires for visual realism. With enough power to render a large number of triangles at real-time rates, the bottleneck of the cards was no longer the fill rate. Rather it was the front end of the graphics pipeline that provides the rasterizers with data. The process of transforming the 3D triangle meshes from world coordinates to camera coordinates, lighting vertices, clipping, and finally projecting and scaling to screen coordinates for the purposes of rasterization became a performance issue. The next generation of graphics cards arrived, called *hardware transform and lighting* (HW T&L) cards, the name referring to the fact that now the work of the graphics pipeline has been off-loaded from the CPU to the *graphics processing unit* (GPU). Although the intent of HW T&L cards was to support the standard graphics pipeline, most of these cards also supported some animation, namely *skin-and-bones* or *skinning* where the vertices of a triangle mesh (the skin) are associated with a matrix hierarchy (the bones), and a set of offsets and a set of weights relative to the bones. As the matrices vary during run time, the vertices are computed from the matrices, offsets, and weights, and the triangle mesh deforms in a natural way. Thus, we have some hardware support for *deformable bodies*.

The standard graphics pipeline is quite low-level when it comes to lighting of vertices. Dynamic lights in a scene and normal vectors at vertices of a triangle mesh are combined to produce vertex colors that are interpolated across the triangles by the rasterizer. Textured objects are rendered by assigning texture coordinates to the vertices of a mesh, the coordinates used as a lookup into a texture image. The rasterizer interpolates these coordinates during rasterization, then performs a lookup on a per-pixel basis for each triangle it rasterizes in the mesh. With a lot of creativity on the artists' end, the vertex coloring and texturing functions can be used to produce high

此为试读,需要完整PDF请访问: www.ertongbook.com

quality, realistic renderings. Fortunately, artists and programmers can create more interesting effects than a standard graphics pipeline can handle, producing yet more impetus for graphics cards to evolve. The latest generation of graphics cards now are programmable and support *vertex shading*, the ability to incorporate per-vertex information in your models and tell the rasterizer how to interpolate them. Clever use of vertex shading allows you to control more than color. For example, displacement mapping of vertices transfers some control of positional data to the rasterizer. And the cards support *pixel shading*, the ability to incorporate per-pixel information via images that no longer are required to represent texture data. Dot3 bumpmapping is the classic example of an effect obtained by a pixel shader function. You may view vertex shading as a generalization of the vertex coloring function and pixel shading as a generalization of the basic texturing function.

The power of current generation graphics cards to produce high quality visual effects is enormous. Much of the low-level programming you would do for software rendering is now absorbed in the graphics card drivers and the graphics APIs built on top of them, such as OpenGL and DirectX, allowing the programmers to concentrate at a higher level in a graphics engine. From a visual perspective, game designers and programmers have most of what they need to create realistic looking worlds for their gamer customers. But since you are reading this preface, you already know that visual realism is only half the battle. *Physical realism* is the other half. A well-crafted, good-looking character will attract your attention for the wrong reasons if it walks through a wall of a room. And if the characters cannot realistically interact with objects in their physical environment, the game will not be as interesting as it could be.

Some day we programmers will see significant hardware support for physics by off-loading work from the CPU to a *physics processing unit* (PPU). Until that day arrives we are, so to speak, at the level of software rendering. We need to implement everything ourselves, both low level and high level, and it must run on the CPU. Moreover we need real-time rates. Even if the renderer can display the environment at 60 frames per second, if the physics system cannot handle object interactions fast enough, the frame rate for the game will be abysmally low. We are required to understand how to model a physical environment and implement that model in a fast, accurate, and robust manner. Physics itself can be understood in an intuitive manner – after all it is an attempt to quantify the world around us. Implementing a physical simulation on a computer, though, requires more than intuition. It requires mathematical maturity as well as the ability and patience to synthesize a large system from a collection of sophisticated, smaller components. This book is designed to help you build such a large system, a *physics engine* as it were.

I believe this book is a good companion to my book *3D Game Engine Design*, a large tome that discusses the topic of constructing a real-time graphics engine for consumer hardware. *Game Physics* focuses on the topic of real-time physical simulation on consumer hardware. The two, of course, will be used simultaneouly in a game application. *Game Physics* has a similar philosophy to *3D Game Engine Design* in two ways. First, both books were conceived while working on commercial engines and tools to be used for building games. The occurrence of the word "game" in the

titles reflects this. The material in both books applies to more than just game applications. For example, it is possible to build a virtual physics laboratory for students to explore physical concepts. Second, both books assume that the reader's background includes a sufficient level of mathematics. In fact, *Game Physics* requires a bit more background. To be comfortable with the material presented in this book, you will need some exposure to linear algebra, calculus, differential equations, and numerical methods for solving differential equations. All of these topics are covered in an undergraduate program in mathematics or computer science. Not to worry, though. As a refresher, the appendices contain a review of the essential concepts of linear algebra, affine algebra, calculus, and difference equations that you will need to read this book. Two detailed chapters are included that cover differential equations and numerical methods for solving them.

I did not call the book *3D Game Physics* because the material is just as appropriate for one- or two-dimensional settings. Many of the constrained physical models are of lower dimension. For example, a simple pendulum is constrained to move within a plane, even though a rendering of the physical system is in three dimensions. In fact, the material is applicable even to projects that are not game related, for example, supporting a virtual physics laboratory for students to explore physical concepts. I did call the book *Game Physics* and I expect that some readers might object to the title when in fact I do not cover all possible topics one might encounter in a game environment. Moreover, some topics I discuss are not in as much depth as some might like to see. Spending even a few years to write a book, I believe it is impossible to cover all the relevant topics in significant detail to support building a fully featured physics engine that rivals what you see commercially. Some projects just require a team of more than one. For example, I specifically avoided getting into fluid dynamics because that is an enormous topic all on its own. I chose to focus on the mechanics of rigid bodies and deformable bodies so that you can build a reasonable, working system for physical simulation. Despite this restricted coverage, I believe there is a significant amount of content in this book that will make it worth every minute of your reading time. This content includes both the written text *and* a vast amount of source code on the CD-ROM that accompanies the book, including both the Wild Magic graphics engine and components and applications for physics support. I have made every attempt at presenting all the content in a manner that will suit your needs.

As in the production of any book, the author is only part of the final result. The reviewers for an early draft of this book have been extremely helpful in providing guidance for the direction the book needed to take. The original scope of the book was quite large, but the reviewers' wisdom led me to reducing the scope to a manageable size by focusing on a few topics rather than providing a large amount of background material that would detract from the main purpose of the book – showing you the essentials of physical simulation on a computer. I wish to personally thank the reviewers for their contributions: Ian Ashdown (byHeart Consultants), Colin Barrett (Havok), Michael Doherty (University of the Pacific), Eric Dybsand (Glacier Edge Technology), David Eberle (Havok), Todd Growney (Electronic Arts), Paul Hemler (Wake Forest University), Jeff Lander (Darwin 3D), Bruce Maxim (University of

Michigan–Dearborn), Doug McNabb (Rainbow Studios), Jon Purdy (University of Hull), and Craig Reinhart (California Lutheran University). Thanks also go to Tim Cox, my editor, Stacie Pierce, editorial coordinator, and Rick Camp, editorial assistant for the book. Tim has been patient with my seemingly endless delays in getting a final draft to him. Well, the bottom line is that the draft arrived. Now it is your turn to enjoy reading the book!

ABOUT THE CD-ROM

License Agreement

The accompanying CD-ROM contains source code that illustrates the ideas in the book. Each source file has a preamble stating that the source code is subject to the Boost License (*http://www.boost.org/LICENSE_1_0.txt*), which is quite simple:

> Boost Software License – Version 1.0 – August 17th, 2003
>
> Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:
>
> The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.
>
> THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Installing and Compiling the Source Code

The Wild Magic 5 engine is portable and runs on desktop computers running the Microsoft Windows XP/Vista/7 operating systems or Linux operating systems. Renderers are provided for both OpenGL (version 2.x) and Direct3D (version 9). The engine also runs on Apple computers, whether PowerPC- or Intel-powered, with the Macintosh OS X operating system (version 10.5.x or higher). Project files are provided for Microsoft Visual Studio (version 2009) on Microsoft Windows. Make files are provided for Linux. Xcode 3.x project files are provided for the Macintosh.

The CD-ROM includes a file, *Wm5p0InstallationRelease.pdf*, that contains the installation directions. You must read this first as it addresses various computing environment issues that you must be aware of, such as setting environment variables and project path settings.

Updates and Bug Fixes

Regularly visit the Geometric Tools, LLC web site, *http://www.geometrictools.com*, for updates and bug fixes. Histories of source code changes, book corrections, and known problems are maintained at the web site.

CONTENTS

此为试读,需要完整PDF请访问: www.ertongbook.com