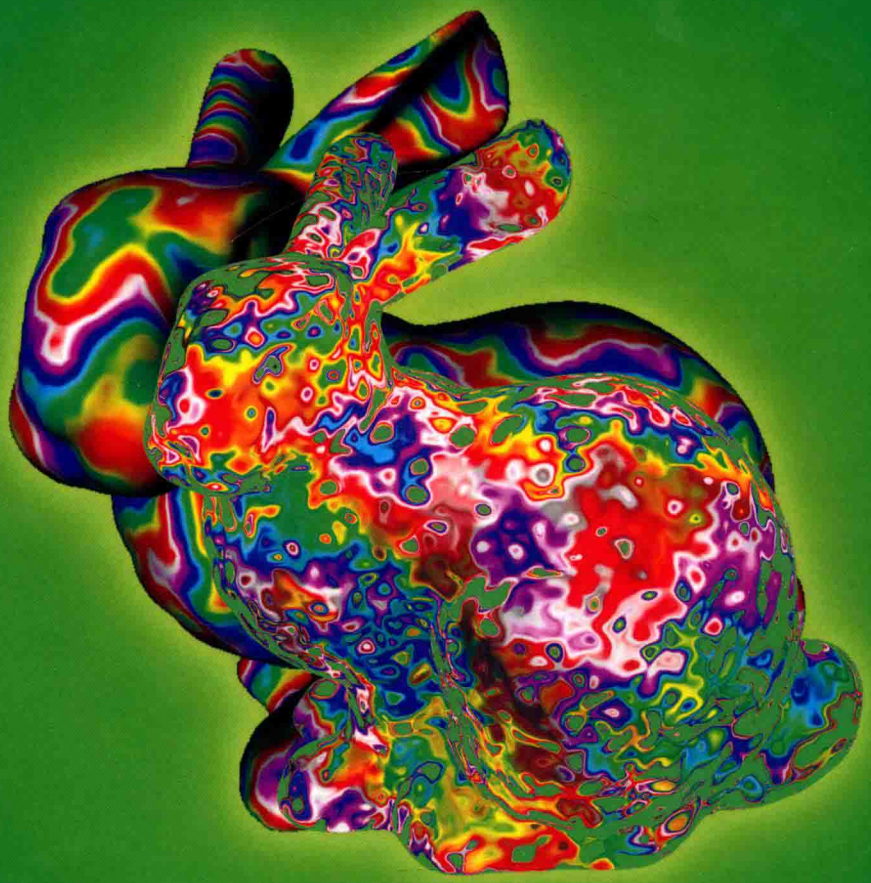
INCLUDES OPENGL 4.X

CRC Press
Taylor & Francis Group
AN A K PETERS BOOK

Graphics Shaders

THEORY AND PRACTICE

SECOND EDITION



MIKE BAILEY · STEVE CUNNINGHAM

Graphics Shaders

Second Edition

Theory and Practice

Mike Bailey

Steve Cunningham

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an Informa business
AN A K PETERS BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed in the United States of America on acid-free paper
Version Date: 2011913

International Standard Book Number: 978-1-56881-434-6 (Hardback)

Library of Congress Cataloging-in-Publication Data

Bailey, Michael (Michael John), 1953-
Graphics shaders : theory and practice / Mike Bailey, Steve Cunningham.-- 2nd ed.
p. cm.
Summary: "This book uses examples in OpenGL and the OpenGL Shading Language to present the theory and application of shader programming. It explains how to program graphics shaders effectively for use in art, animation, gaming, and visualization. Along with improved graphics and new examples and exercises, this edition includes a discussion on handling OpenGL's evolution beyond its original built-in functionality, including four new appendices that provide C++ class code to help in this transition. It includes a new chapter on tessellation shaders. It also discusses shaders in multipass rendering and presents new applications including terrain bump mapping, morphing 3D geometry, and wavy glass."-- Provided by publisher.
Includes bibliographical references and index.
ISBN 978-1-56881-434-6 (hardback)
1. Computer graphics. 2. OpenGL. 3. Three-dimensional display systems. I. Cunningham, Steve. II. Title.

T385.B3455 2011
006.6'6--dc23 2011031720

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

Graphics Shaders

Second Edition

To my parents,
Ted and Anne Bailey,
whose respect for both curiosity and books
made a project like this inevitable sometime.
– MJB

To the other writers in my family:
Judy,
for her collaboration on so many projects
and her patience with my work on this one, and
Rob and Rick,
for their own past and future writing projects.
– SC

Foreword

Excellent! I am glad that you are reading this book. You might want to skip straight ahead to the good stuff, but as long as you are here…

Computer graphics is a fascinating and fast-changing field that didn't even exist when I was born. I was attracted to it because it is a field with a unique mix of engineering and artistry. In the computer graphics industry, people with engineering skills design graphics software and hardware products that offer ever-increasing levels of performance and image quality. These products inspire people with artistic skills to use the resulting products to create amazing visual experiences that entertain, teach, or help others create or design. This in turn inspires the engineers to create even better hardware and software in order to improve the visual experiences created by artists. This symbiotic relationship between engineers and artists has never let up and has

resulted in photorealistic effects for movies and near-cinematic quality experiences for computer games.

You might be reading this book because of your interest in the computer graphics field. Perhaps you are an engineer looking to develop another tool for your toolbox of software development skills for computer graphics. Perhaps you are an artist who is interested in learning a little more about the bits and bytes of how computer graphics images are created. Perhaps you are that rare breed, an engineer/artist, and you have in your mind's eye a vision of what you want to create, and you need only to develop an understanding of this new medium in order to bring your vision to reality. If any of these are true, you have selected an excellent guide book to help you on your journey.

You are holding in your hands a book written by two people who share two passions. Mike Bailey and Steve Cunningham both love computer graphics, and they are absolutely passionate about teaching. This book allows them to combine both of these passions into a form that is sure to benefit you, the reader.

Actually, the word "passionate" understates the impact that Mike and Steve have had on computer graphics education. Mike is a "lifer" in the computer graphics industry. I met him some 15 years ago when we asked him to lead an effort to define industry-standard benchmarks for computer graphics systems (which he graciously agreed to do). He has been teaching or practicing computer graphics for almost 30 years now. He has won numerous awards as a professor of computer graphics. His dedication to educating people new to graphics is demonstrated by the fact that he annually prepares and delivers the "Introduction to Computer Graphics" tutorial at SIGGRAPH (ACM's Special Interest Group on Graphics).

Steve is a similarly dedicated, accomplished, and award-winning educator. He was a co-founder of the SIGGRAPH Education Committee and co-chaired this activity for many years. He served in countless leadership positions in the SIGGRAPH organization and for the SIGGRAPH conference itself (the largest, most prestigious, and longest-lived conference focusing on computer graphics). For his lifelong efforts, he was given the 2004 ACM SIGGRAPH Outstanding Service Award. His influence on the computer graphics industry is global, as witnessed by the fact that he was the first Eurographics Education Board chair and he has been named a Eurographics Fellow.

So it is certainly the case that these two authors can tell you a thing or two about computer graphics. But even more importantly, they can tell it to you in a way that you will understand and remember.

The topic of this book, writing shaders with the OpenGL Shading Language, is both important and timely. OpenGL and its companion shading

language are industry standards. This means that they are supported by a variety of hardware companies on a variety of operating environments. OpenGL and GLSL are available on Macs, PCs, and Linux systems; on workstations, towers, desktops, laptops, and handhelds. The goal of a standard is simple: to make it easy for you, the programmer, to deploy your code on a diverse range of products without requiring any changes to the source code. The resulting portability amortizes the cost of the software development by creating a bigger market for software products based on industry standards.

But the most important part of this book is that while it is teaching you how to write programmable shaders, it is also teaching and reinforcing the fundamentals of computer graphics. As a result, you will be able to easily adapt the lessons learned here to other shading languages and graphics paradigms. This is becoming increasingly important since the trend for graphics hardware is to offer more general programmability and less fixed functionality built into hardware. In other words, we are returning to the days where computer graphics innovation occurs in software. The knowledge and skills that you learn while reading this book can be adapted to the even more general graphics programming environments of the future.

At the end of each chapter in this book, you will find some exercises that will help develop your knowledge of graphics and programmable shading. In that spirit, here are the exercises that I would prescribe for you:

1. Read this book.
2. Use computer graphics and programmable shading to create beauty.
3. Share your creation and your knowledge with others.

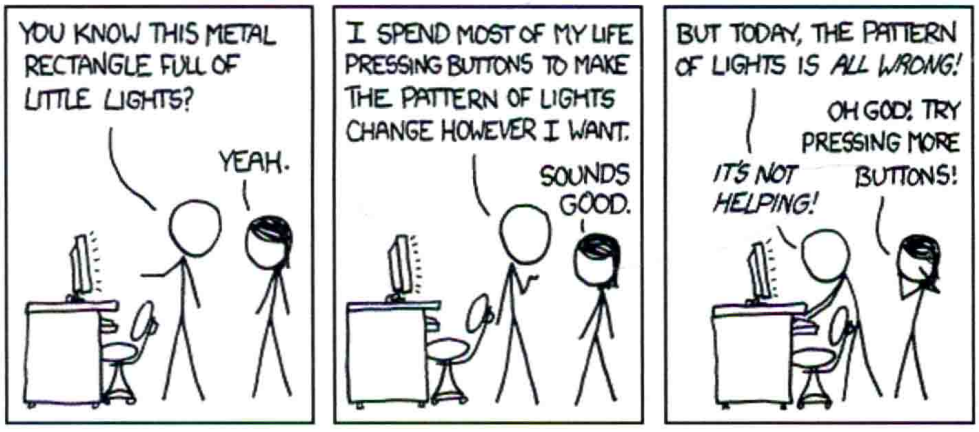
Most importantly,

4. Have fun!

Randi Rost
December 31, 2008

Preface

Does this remind you of yourself?



http://xkcd.com

You have lots of great, creative ideas in your head, but can't seem to get the right pixels to come out onto your graphics screen. Then, you are our type of person. And, this is your type of book.

Welcome to the second edition of *Graphics Shaders: Theory and Practice*. As the name implies, this book deals with both the theory and equations behind what shaders do, as well as lots and lots of code examples of putting the theory into practice. To help you, this book has been printed with color throughout. That means that the lots of examples have lots of images to go with them to help understand the concepts. So stop and stay for a while. Put your feet up and start reading. You are really going to enjoy this.

This book has over 100 more pages than the first edition did. Here are the major improvements:

1. This book is written against the most-recent specification releases: OpenGL 4.x and GLSL 4.x0.
2. All code examples have been brought up-to-date with the current standard of the GLSL language.
3. There is an entire chapter (with examples) on the new tessellation shaders.
4. All chapters have more examples and more exercises.
5. Many diagrams have been improved. The ones involving GLSL functionality levels have been brought up to 4.x0.
6. The OpenGL Architecture Review Board (ARB) has depecated some portions of OpenGL, but has not eliminated them. This edition discusses that, and presents a strategy to write your own code with that in mind. All code examples in this book now follow that strategy. Also, by following that strategy, you will be prepared for migration to OpenGL-ES 2.0.
7. Appendices have been added showing the use of C++ classes to make writing OpenGL shader applications easier, and help with the post-deprecation strategy.

Programmable computer graphics shaders have had an interesting history. In not-too-distant memory, at least for some of us, *all* aspects of computer graphics were programmable. In fact, "programmable" is probably not a good term, because that implies that there was a programmability option when creating an image. There wasn't. If you wanted *anything* to happen, you had no choice but to program it. Yourself. "Involuntary programmability" might be a better way to put it.

Computer graphics APIs changed that for most graphics practitioners. With a good API, you could write very good graphics programs much more easily because you could let the API's functionality take over large portions of the graphics process. However, you paid for this in giving up any functionality that the API didn't know how to handle. A good example is surface shading, where most of the 1990s APIs could not support anything beyond simple smooth lighted surfaces.

Fortunately, neither the computer graphics research community nor advanced graphics practitioners were satisfied with this. First in software and then in hardware, as graphics processors were developed, specific functionality was developed to support the programming of features that fixed-function graphics APIs had fenced off. This functionality has now developed its own standards, including the GLSL shader language that is part of the OpenGL standard. Programmable graphics shaders, programs that can be downloaded

to a graphics processor to carry out operations outside the fixed-function pipeline of earlier standards, have become a key feature of computer graphics.

This process is now being paralleled in the teaching and learning of computer graphics. Just as students usually first learned computer graphics through a graphics standard, most often OpenGL, students now need to understand the role of programmable shaders and to have experience in writing and using them. One of the remarkable things about shader-level programming is that it brings us all back to the same kind of graphics questions that were being examined in the 1970s. We can now manipulate vertices and individual pixels while still having the full OpenGL API high-speed support whenever we want to use it. This gives students and practitioners a wonderful range of capabilities that can be used in games, in scientific visualization, and in general graphical communication.

This book is designed to open computer graphics shader programming to students, whether in a traditional class or on their own. It is intended to complement texts based on fixed-function graphics APIs, specifically OpenGL. It introduces shader programming in general, and specifically the GLSL shader language. It also introduces a flexible, easy-to-use tool, *glman*, which helps you develop and tune shaders outside an application that would use them.

This book is intended as a text for a second course in computer graphics at either the undergraduate or graduate level. It is not a textbook for a first course in computer graphics, because it assumes knowledge of not only OpenGL, but of general graphics concepts. Knowledge of another graphics API, such as Direct3D, will work, but we focus on GLSL and will use OpenGL terminology consistently. Because shader programming lets you work in areas that APIs might hide from you, sometimes you will need to work at fundamental levels of geometry, lighting, shading, and similar concepts. You will benefit from a prior understanding of these. You will also find that shader programming exposes some areas of API operation that you may not have fully understood, so you may need to review some of these details.

Our choice of GLSL as the vehicle for teaching shaders is based on its integration into the widely-used OpenGL multiplatform API and its solid performance. The concepts presented here will also help anyone who works with other shader APIs such as Cg or HLSL, because the basic ideas of shaders are all similar. The book is designed to take the student from a review of the fixed-function graphics pipeline through an understanding of the basic role and functions of shader programming to solid experience in writing vertex, fragment, and geometry shaders for both *glman* and actual applications.

While it might seem logical to treat shaders in the order in which they are applied in the expanded graphics pipeline, with vertex shaders first, followed

by geometry shaders and then fragment shaders, we have chosen to lay out their order a little differently. Again, it might seem logical to treat shaders in the order of frequency of use, with fragment shaders first, followed by vertex shaders and then geometry shaders, but that also does not quite seem to work. Because many of the operations of a fragment shader depend on things that come out of a vertex shader, we treat vertex shaders first, followed by fragment shaders, and finally geometry and tessellation shaders.

The overall outline of the text is straightforward. In the first chapters, which make up the background for the rest of the book, we begin by covering the fixed-function graphics pipeline of OpenGL in **Chapter 1,** and OpenGL shader evolution in **Chapter 2**. We then present the basic principles of vertex, fragment, geometry, and tessellation shaders in **Chapter 3**, including several examples, using the GLSL shader language. **Chapter 4** introduces the *glman* tool with a kind of mini-manual on its use. Finally, **Chapter 5** presents the GLSL shader language and discusses its similarities and differences from the C programming language.

The next set of chapters sets up vertex and fragment shader concepts. **Chapter 6** covers lighting from the point of view of shaders and introduces the ADS (ambient, diffuse, specular) lighting function that we will use several times in later chapters. This is fundamental in both vertex and fragment shaders, since vertex shaders often need to compute lighting for each vertex, and fragment shaders may want to compute lighting for each pixel. In **Chapter 7** we cover vertex shaders, emphasizing their inputs and outputs as well as the ways they can be used to modify vertex geometry. Finally, in **Chapter 8** we cover fragment shaders, again emphasizing their inputs and outputs and showing how they can be used to replace the usual fixed-function fragment operations.

The next three chapters discuss particular capabilities of fragment shaders. In **Chapter 9** we describe the way fragment shaders handle texture mapping, including bump mapping, cube mapping, and rendering a scene to a texture. **Chapter 10** discusses noise functions and their role in writing textures and shaders, and introduces a tool, *noisegraph*, that lets you experiment with the properties of 1D and 2D noise functions. Finally, **Chapter 11** examines some ways you can manipulate 2D images, treated as textures, with the tools that fragment shaders make available.

Chapter 12 presents geometry shaders, including how they are related to vertex and fragment shaders as well as their own capabilities. Several examples highlight the way geometry shaders can expand the geometric capability of your models or show the capability of geometry shaders to handle simple level-of-detail operations. **Chapter 13** discusses tessellation shaders. We

show how they are somewhat similar to geometry shaders but have important enhancements.

The final set of chapters focuses on computer graphics shaders in applications. **Chapter 14** describes the GLSL API that lets you compile, link, and use shaders in an application. It also discusses passing data and graphics state information to shader programs and introduces a simple C++ class that encapsulates the process of incorporating shader programs in an application. In **Chapter 15**, we focus on how shaders can be used in scientific visualization applications, and show examples of a number of specific visualization operations. And in **Chapter 16** we explore some fun things you can do with computer graphics shaders, under the guise of getting real work done. (Don't tell anyone.)

Four appendices have been added showing the use of C++ classes to help write OpenGL applications and handle some of the post-deprecation challenges.

While many of the topics in this text are straightforward, some are tricky or deserve special attention. We have followed the lead of the Nicholas Bourbaki mathematics texts of the early 20th century and have highlighted these with a "dangerous curves ahead" sign as shown to the right. We hope this will help you notice these points.

Because shader functions are changing, there are times when we want to highlight things that have evolved or things we introduce to deal with these changes. We have used a second sign, shown at right, to draw your attention to these points.

We are confident that the tools and capabilities we describe in this book will both make you a better graphics programmer and make graphics programming a much more interesting experience for you. As OpenGL evolves toward the future and shaders become the only way that geometry and rendering are handled, we believe that you will find this text to be an invaluable guide.

Thanks

The authors of this book owe thanks to a number of people, primarily on Mike Bailey's side.

To faculty colleagues at Oregon State University for their support and camaraderie: Ron Adams, Bella Bose, Terri Fiez, Karti Mayaram Ron Metoyer, Eric Mortensen, Cherri Pancake, Sinisa Todorovic, and Eugene Zhang.

To the superbly talented UCSD and OSU graphics students who have shared this shader expedition: Tim Bauer, William Brendel, Guoning Chen, Matt Clothier, John Datuin, Will Dillon, Jonathan Dodge, Chuck Evans, Nick Gebbie, Kyle Hatcher, Nick Hogle, Chris Janik, Ankit Khare, Vasu Lakshmanan, Adam Leibel, Jessica McGregor, Daniel Moffitt, Chris Moore, Patrick Neill, Jonathan Palacios, Nadia Payet, Randy Rauwendaal, Dwayne Robinson, Avneet Sandhu, Nick Schultz, Sudarshanram Shetty, Evon Silvia, Ian South-Dickinson, Madhu Srinivasan, Michael Tichenor, Christophe Torne, Ben Tribelhorn, Ben Weiss, and Alex Wiggins.

To professional colleagues: Ryan Bailey, Mike Gannis, Jenny Orr, Todd Shechter, and Justin Spencer.

To the folks at NVIDIA for their support, especially Gary Brown, Greg Gritton, Jen-Hsun Huang, David Kirk, Dave Luebke, and David Zier.

To the folks at AMD/ATI for their support, especially Bill Licea-Kane.

To Randi Rost, for his support from positions at both 3D Labs and Intel, and for writing his "Orange Book," from which so much of what went into this book was learned.

To Paramount Pictures for their permission to reprint the image in Figure 2.2. and to Pixar for providing the original image.

To xkcd.com for the comic used at the front of this Preface.

We also thank Alice Peters and Sarah Cutler for their advice and assistance in developing this project, and the reviewers for helping us refine some key points in the text.

Mike Bailey
Corvallis, Oregon

Steve Cunningham
Coralville, Iowa

Contents