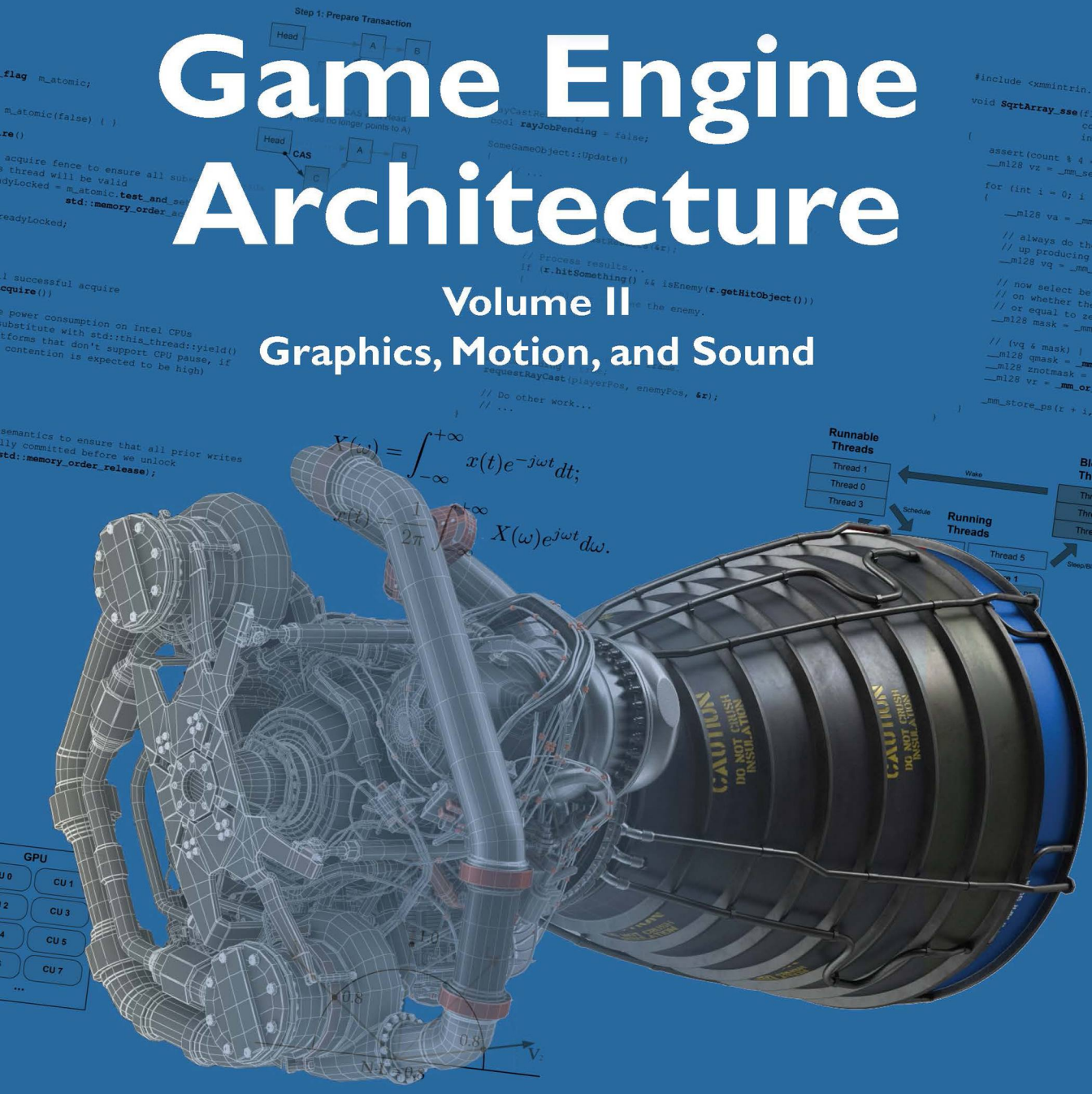
FOURTH EDITION

Game Engine Architecture

Volume II
Graphics, Motion, and Sound



Jason Gregory

CRC Press
Taylor & Francis Group

Game Engine Architecture

In this new and improved fourth edition of the highly popular **Game Engine Architecture**, Jason Gregory draws on his nearly three decades of experience at Midway, Electronic Arts, and Naughty Dog to present both the theory and practice of game engine software development. In this book, the broad range of technologies and techniques used by AAA game studios are each explained in detail, and their roles within a real industrial-strength game engine are illustrated.

The fourth edition offers the same comprehensive coverage of game engine architecture provided by previous editions, and this second volume delves into all of the **major subsystems** found in **every** game engine, including the animation engine, the 3D rendering engine, the lighting engine, the physics engine, the audio engine, and the game object model.

This book is intended to serve as an introductory text, but it also offers the experienced game programmer a useful perspective on aspects of game development technology with which they may not have deep experience. As always, copious references and citations are provided in this edition, making it an excellent jumping off point for those who wish to dig deeper into any particular aspect of the game development process.

Key Features

- Covers both the theory and practice of game engine software development
- Examples are grounded in specific technologies, but discussion extends beyond any particular engine or API
- Includes all mathematical background needed
- Comprehensive text for beginners and also has content for senior engineers

Jason Gregory has worked as a software engineer in the games industry since March 1999, and as a professional software engineer since 1994. He got his start in game programming at Midway Home Entertainment in San Diego, where he worked on tools, engine technology and gameplay code for *Hydro Thunder*™ 2 (arcade). He also wrote the PlayStation® 2/Xbox animation system for *Freaky Flyers*™ and *Crank the Weasel*. In 2003, Jason moved to Electronic Arts Los Angeles, where he worked on engine and gameplay technology for Medal of Honor:™ Pacific Assault and served as a lead engineer on the *Medal of Honor*:™ *Airborne* project. Jason is currently a Lead Programmer at Naughty Dog Inc., where he and his team are working on an exciting new game title for PlayStation® 5. He also developed engine and gameplay software for Naughty Dog's *Uncharted: Drake's Fortune*™, *Uncharted 2: Among Thieves*™, *Uncharted 3: Drake's Deception*™, *Uncharted 4: A Thief's End*™, *Uncharted: The Lost Legacy*™, *The Last of Us Part I*™ and *The Last of Us Part II*™ for PlayStation® 5, PlayStation® 4, PlayStation® 3, and PC. Jason has also developed and taught courses in game technology at the University of Southern California.

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Game Engine Architecture

Volume II
Graphics, Motion, and Sound

Fourth Edition

Jason Gregory

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

Designed cover image: TurboSquid

Fourth edition published 2026
by CRC Press
2385 NW Executive Center Drive, Suite 320, Boca Raton FL 33431

and by CRC Press
4 Park Square, Milton Park, Abingdon, Oxon, OX14 4RN

CRC Press is an imprint of Taylor & Francis Group, LLC

First edition published by Taylor and Francis 2009
Third edition published by Taylor and Francis 2019

Library of Congress Cataloging-in-Publication Data
Names: Gregory, Jason, 1970- author
Title: Game engine architecture / Jason Gregory.
Description: Fourth edition. | Boca Raton : CRC Press, 2026. | Includes bibliographical references and index. | Contents: volume I. Foundations of game engine development -- volume II. Graphics, motion and sound.
Identifiers: LCCN 2025030306 (print) | LCCN 2025030307 (ebook) | ISBN 9781032443065 v. 1 hardback | ISBN 9781032443089 v. 1 paperback | ISBN 9781041162582 v. 2 hardback | ISBN 9781041162575 v. 2 paperback | ISBN 9781003371519 v. 1 ebook | ISBN 9781003683605 v. 2 ebook
Subjects: LCSH: Video games--Programming--Computer programs. | Software architecture. | Video games--Design.
Classification: LCC QA76.76.C672 G77 2026 (print) | LCC QA76.76.C672 (ebook)
LC record available at https://lccn.loc.gov/2025030306
LC ebook record available at https://lccn.loc.gov/2025030307

ISBN: 978-1-041-16258-2 (hbk)
ISBN: 978-1-041-16257-5 (pbk)
ISBN: 978-1-003-68360-5 (ebk)

DOI: 10.1201/9781003683605

Typeset in URW Palladio font
by KnowledgeWorks Global Ltd.

Publisher's note: This book has been prepared from camera-ready copy provided by the authors.

Dedicated to
Trina, Evan and Quinn Gregory,

in memory of our heroes,
Joyce Osterhus and Kenneth Gregory.

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Contents

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Preface

Welcome to *Game Engine Architecture.* This book aims to present a complete discussion of the major components that make up a typical commercial game engine. Game programming is an immense topic, so we have a lot of ground to cover. Nevertheless, I trust you'll find that the depth of our discussions is sufficient to give you a solid understanding of both the theory and the common practices employed within each of the engineering disciplines we'll cover. That said, this book is really just the beginning of a fascinating and potentially lifelong journey. A wealth of information is available on all aspects of game technology, and this text serves both as a foundation-laying device and as a jumping-off point for further learning.

Our focus in this book will be on game engine technologies and architecture. This means we'll cover the theory underlying the various subsystems that comprise a commercial game engine, the data structures, algorithms, and software interfaces that are typically used to implement them, and how these subsystems function together within a game engine as a whole. The line between the game engine and the game is rather blurry. We'll focus primarily on the engine itself, including a host of low-level foundation systems, the rendering engine, the collision system, the physics simulation, character animation, audio, and an in-depth discussion of what I call the gameplay foundation layer. This layer includes the game's object model, world editor, event system, and scripting system. We'll also touch on some aspects of gameplay programming, including player mechanics, cameras, and AI. However, by necessity, the scope of these discussions will be limited mainly to the ways in which gameplay systems interface with the engine.

This book is intended to be used as a course text for a two- or three-course college-level series in intermediate game programming. It can also be used by amateur software engineers, hobbyists, self-taught game programmers, and existing members of the game industry alike. Junior engineers can use this text to solidify their understanding of game mathematics, engine architecture, and game technology. And some senior engineers who have devoted their careers to one particular specialty may benefit from the bigger picture presented in these pages as well.

To get the most out of this book, you should have a working knowledge of basic object-oriented programming concepts and at least some experience programming in C++. The game industry routinely makes use of a wide range of programming languages, but industrial-strength 3D game engines are still written primarily in C++. As such, any serious game programmer needs to be able to code in C++. We'll review the basic tenets of object-oriented programming in Chapter 3, and you will no doubt pick up a few new C++ tricks as you read this book, but a solid foundation in the C++ language is best obtained from [51], [52], [40], and [41]. If your C++ is a bit rusty, I recommend you refer to these or similar books to refresh your knowledge as you read this text. If you have no prior C++ experience, you may want to consider reading at least the first few chapters of one or more of those books and/or working through a few C++ tutorials online before diving into this book.

The best way to learn computer programming of any kind is to actually write some code. As you read through this book, I strongly encourage you to select a few topic areas that are of particular interest to you and come up with some projects for yourself in those areas. For example, if you find character animation interesting, you could start by installing OGRE and exploring its skinned animation demo. Then you could try to implement some of the animation blending techniques described in this book, using OGRE. Next you might decide to implement a simple joypad-controlled animated character that can run around on a flat plane. Once you have something relatively simple working, expand upon it! Then move on to another area of game technology. Rinse and repeat. It doesn't particularly matter what the projects are, as long as you're practicing the art of game programming, not just reading about it.

Game technology is a living, breathing thing that can never be entirely captured within the pages of a book. As such, additional resources, errata, updates, sample code, and project ideas will be posted from time to time on this book's website at [59]. You can also connect with me on LinkedIn at [60].

New to the Fourth Edition

In the time between the publication of the third edition of *Game Engine Architecture* and this, its fourth edition, real-time rendering and lighting techniques have evolved immensely in terms of their sophistication and their ability to achieve photorealism. During this period, the raw power, feature set, and architecture of the GPU have also undergone transformative change. Today's GPUs are equipped with entirely new hardware components, including a host of new rendering features and shader types, real-time ray tracing accelerators, and fast tensor compute cores for machine learning. In response to these changes, I realized that the fourth edition of *Game Engine Architecture* would require an entirely new approach to discussing 3D rendering and photorealistic lighting in the context of game engines. As a result, the book now devotes two brand new chapters to addressing these deep and fascinating topics.

As a consequence of adding so much new material to the book, its girth and mass have grown steadily from edition to edition. Some readers have told me that their copy of the book found a secondary use—as part of the weight rack in their home gym! To avoid the book becoming too physically cumbersome, it has been split into two volumes for the fourth edition. *Volume I, Foundations and Core Engine Systems* covers essential concepts, techniques, tools, and the engine systems that form the core of every game engine. *Volume II, Graphics, Motion, and Sound* delves into all of the major subsystems found in every game engine, including the animation engine, the 3D rendering engine, the lighting engine, the physics engine, the audio engine, and the game object model. While each volume can be read standalone, I highly recommend obtaining and reading both volumes so you'll have a complete picture of the game engine landscape.

This fourth edition of *Game Engine Architecture* also improves upon the treatment of various topics covered in prior editions. A discussion of undefined behavior in the context of compiler optimizations has been added. Fuller coverage of the various C++ language standards up to C++23 is included. The animation chapter has been augmented with a discussion of motion matching. And, as with all previous editions, various errata have been repaired that were brought to my attention by you, my devoted readers. Thank you! I hope you'll find that the mistakes you found have all been fixed. (Although no doubt they have been replaced by a slew of *new* mistakes, about which you can feel free to inform me!)

Of course, as I've said before, the field of game engine programming is almost unimaginably broad and deep. There's no way to cover every topic in one book. As such, the primary purpose of this book remains to serve as an awareness-building tool and a

jumping-off point for further learning. I hope you find this edition helpful on your journey through the fascinating and multifaceted landscape of game engine architecture.

Acknowledgments

No book is created in a vacuum, and this one is certainly no exception. This book—and its fourth edition, which you now hold in your hands—would not have been possible without the help of my family, friends, and colleagues in the game industry, and I'd like to extend warm thanks to everyone who helped me to bring this project to fruition.

Of course, the ones most impacted by a project like this are invariably the author's family. So I'd like to start by offering, for a *fourth time*, a special thank-you to my wife Trina. She was a pillar of strength during the writing of the original book, and she has continued to be just as supportive and invaluably helpful during my work on the second, third, and fourth editions. While I was busy tapping away on my keyboard, Trina was always there to help out our two children, Evan (now age 21) and Quinn (age 19), as they tackled the challenges of university life, often forgoing her own plans, and always giving me kind words of encouragement when I needed them the most.

I would also like to extend special thanks to my editors for the first edition, Matt Whiting and Jeff Lander. Their insightful, targeted, and timely feedback was always right on the money, and their vast experience in the game industry helped to give me confidence that the information presented in these pages is as accurate and up-to-date as humanly possible. Matt and Jeff were both a pleasure to work with, and I am honored to have had the opportunity to collaborate with such consummate professionals on this project. I'd like to extend a special thank-you to Jeff for putting me in touch with Alice Peters and helping me to get this project off the ground in the first place. Matt, thank you also for stepping up to the plate once again and providing me with valuable feedback on the new rendering and lighting chapters in the fourth edition.

A number of my colleagues at Naughty Dog also contributed to this book, either by providing feedback or by helping me with the structure and topic content of one of the chapters. I'd like to thank Marshall Robin, Carlos Gonzalez-Ochoa, and Pål-Kristian Engstad for their guidance and tutelage as I wrote the original rendering chapter, and Marshall as well as Joe Forte and Abhishek Arora for their excellent and insightful feedback on the content of the new rendering and lighting chapters. I'd like to extend a special thank-you to Abhishek for providing me with the detailed notes that evolved into the section on volumetric skies and clouds. My thanks also go to

Christian Gyrling for his feedback on various sections of past editions of the book, including the chapter on animation, and the chapter on parallelism and concurrency. I'd also like to thank Kareem Omar for his valuable insights and feedback on the concurrency chapter. I want to extend a special thank-you to Jonathan Lanier, Naughty Dog's resident senior audio programmer extraordinaire, for providing me with a great deal of the raw information you'll find in the audio chapter, for always being available to chat when I had questions, and for providing laser-focused and invaluable feedback after reading the initial draft of that chapter. My thanks also go to the entire Naughty Dog engineering team for creating the world-class exemplars of the various game engine systems that I highlight in this book, and from which I've learned so much.

Additional thanks go to Keith Schaeffer of Electronic Arts for providing me with much of the raw content regarding the impact of physics on a game, found in the section entitled "Do You Want Physics in Your Game?" I'd also like to extend a warm thank-you to Christophe Balestra (who was co-president of Naughty Dog during my first ten years there), Paul Keet (who was a lead engineer on the *Medal of Honor* franchise during my time at Electronic Arts), and Steve Ranck (the lead engineer on the *Hydro Thunder* project at Midway San Diego), for their mentorship and guidance over the years. While they did not contribute to the book directly, they did help to make me the engineer that I am today, and their influences are echoed on virtually every page in one way or another.

This book arose out of the notes I developed for a course entitled "ITP-485: Programming Game Engines," which I taught under the auspices of the Information Technology Program at the University of Southern California for approximately four years. I would like to thank Dr. Anthony Borquez, the director of the ITP department at the time, for hiring me to develop the ITP-485 course curriculum in the first place.

My extended family and friends also deserve thanks, in part for their unwavering encouragement, and in part for entertaining my wife and our two kids on so many occasions while I was working. I'd like to thank my sister- and brother-in-law, Tracy Lee and Doug Provins, my cousin-in-law Matt Glenn, and all of our incredible friends, including Kim and Drew Clark, Sherilyn and Jim Kritzer, Anne and Michael Scherer, Kim and Mike Warner, and Kendra and Andy Walther. When I was a teenager, my father Kenneth Gregory wrote *Extraordinary Stock Profits*—a book on investing in the stock market—and in doing so, he inspired me to write this book. For this and so much more, I am eternally grateful to him. I'd also like to thank my mother Erica Gregory, in part for her insistence that I embark on this project, and in part for spending countless hours with me when

I was a child, beating the art of writing into my cranium. I owe my writing skills, my work ethic, and my rather twisted sense of humor entirely to her!

I'd like to thank Alice Peters and Kevin Jackson-Mead, as well as the entire A K Peters staff, for their Herculean efforts in publishing the first edition of this book. Since that time, A K Peters has been acquired by the CRC Press, the principal science and technology book division of the Taylor & Francis Group. I'd like to wish Alice and Klaus Peters all the best in their future endeavors. I'd also like to thank Rick Adams, Jennifer Ahringer, Jessica Vega, Cynthia Klivecka, Gabrielle Perez, and Will Bateman of Taylor & Francis for their patient support and help throughout the process of creating the second, third and fourth editions of *Game Engine Architecture*, Jonathan Pennell for his work on the cover for the second edition, Scott Shamblin for his work on the third edition's cover art, and the artists at 3d Molier International for creating the beautiful 3D model and renders of the RS-25 space shuttle engine on the cover of the fourth edition.

I am thrilled to be able to say that earlier editions of *Game Engine Architecture* have been or are being translated into Japanese, Chinese, Korean, and Russian! I would like to extend my sincere thanks to Kazuhisa Minato and his team at Namco Bandai Games for taking on the daunting task of the original Japanese translation. My thanks also go out to Hiromi Onuki and Sachi Tanaka at Born Digital Inc. for their work on the Japanese translation of the third edition. I'd also like to thank the folks at Softbank Creative, Inc. for publishing the Japanese version of the book. I extend my warmest thanks to Milo Yip for his hard work and dedication to the Chinese translation project. My sincere appreciation goes to the Publishing House of the Electronics Industry for publishing the Chinese translation of the book, and to both the Acorn Publishing Company and Hongreung Science Publishing Co. for their publication of the Korean translations of the first and second editions, respectively. I'd like to extend a warm thank-you to Sanghee Park of Acorn Publishing for their work on the Korean translation of the third edition, and also to the fine folks at Sprint Books kz / Progress Kinga for their work on the Russian translation.

Many of my readers took the time to send me feedback and alert me to errors in the first, second, and third editions, and for that I'd like to extend my sincere thanks to all of you who contributed. I've tried my best to incorporate all of this feedback into the fourth edition—please keep it coming!

Jason Gregory
February 2026

11
Rendering

When most people think about computer games, the first thing that comes to mind is stunning three-dimensional graphics. Most games these days aim to produce *photorealistic* graphics—imagery with such high detail and physically accurate lighting that it can be difficult to tell the difference between the game's virtual world and reality. Some games opt for a more-stylized look, perhaps emulating the appearance of a cartoon, or a pencil sketch, or an impressionist painting. And of course, two-dimensional game graphics is still very much a thing in the game industry. In the following two chapters, we'll focus primarily on real-time three-dimensional graphics, and we'll generally assume that photorealistic rendering is the goal. However nearly all of what we'll learn can also be applied to non-photorealistic 3D rendering, and we'll briefly touch on some of the ways in which stylized effects can be achieved.

The topic of 3D computer graphics is a subset of an even broader topic known as *computer-generated imagery* (CGI). Computer graphics technologies find many uses outside the game industry, including film and special effects, simulation, computer-aided design (CAD), lighting engineering, medical visualization, and many more. In general, the goal of CGI is to produce a synthetic image as output, given some kind of non-image data as input.

Real-time 3D rendering is a specialized kind of 3D rendering which aims to produce images at the desired level of quality and photorealism at interactive frame rates to provide the illusion of motion. This means achieving a refresh rate of at least 30 *frames per second* (FPS) although frame rates of 60 FPS or higher are more typical in

DOI: 10.1201/9781003683605-11

the games industry. At a frame rate of 60 FPS, the computer has just over 16 milliseconds to generate each image. But the game engine does a lot more than just render graphics every frame; it also needs to update game logic, animate the objects in the scene, run collision detection and the physics simulation, synthesize audio, and so on. As such, a game's rendering engine needs to fulfill two equally-important requirements—to produce highly-detailed photorealistic graphics, and to do so very *very* quickly! Considering that film rendering engines often take anywhere from many minutes to many hours to render a single frame, the quality of real-time computer graphics these days is truly astounding.

Real-time 3D rendering is an exceptionally broad and profound topic, so there's simply no way to cover all of the details in two chapters. Thankfully there are a great many excellent books and other resources available on this topic. In fact, real-time 3D graphics is perhaps one of the best covered of all the technologies that make up a game engine. The goal of these chapters, then, is to provide you with a broad understanding of real-time rendering technology and to serve as a jumping-off point for further learning. After you've read through these pages, you should find that reading other books on 3D graphics seems like a journey through familiar territory.

11.1 The Rendering Problem

A 3D rendering engine is a collection of computer hardware and software whose purpose is to solve a problem we'll call the *rendering problem*: Given a virtual world comprised of *scene elements* (virtual solid objects, fluids, clouds of smoke, and so on), one or more *virtual light sources*, and a *virtual camera*, the task is to produce a two-dimensional color image of the scene as viewed by the camera. We say that the scene elements, light sources, and camera are *virtual* because they don't exist in the real world—they only exist as descriptions residing in the computer's memory. We want the output image to mimic, to some desired and practically feasible degree of realism or stylization, the image that would have been generated had a real-world camera been viewing that same real-world scene.

11.1.1 Conceptualizing the Rendering Problem

There are a number of ways to conceptualize the rendering problem, and hence multiple ways to break down the problem to make it tractable.

11.1.1.1 Rendering as Data Conversion

A program that solves the rendering problem can be viewed as a data conversion algorithm: Its inputs are mathematical/numerical models of scene elements, light sources, and a camera, specified in three-dimensional space. The output is a two-dimensional rectangular array of data, with some desired dimensions $W \times H$, each element of which contains color information encoded in a numeric format that is suitable for display on a TV screen, CRT monitor, or other display device. The data elements making up the output image are called "picture elements" or *pixels* for short.

11.1.1.2 Rendering as a Geometric Problem

The rendering problem can be conceptualized from a geometric point of view. As we'll learn in Section 11.2, each scene object can be represented or approximated by a mathematical description of its exterior surface. This surface geometry can be broken down into a patchwork of simple *geometric primitives* such as triangles, each of which is defined by three points known as *vertices*. To render the scene, the vertices are transformed into the Cartesian coordinate space of the virtual camera and *projected* from three-dimensional space into the two-dimensional space of the display screen.

Once in screen space, the primitive shapes can be drawn as simple wireframes by connecting the transformed vertices with lines. Or they can be *shaded* by filling the screen pixels that lie interior to each primitive with a color. The color of each pixel is typically selected either by reading color information from a 2D image known as a *texture map* and/or by performing approximate or photorealistic lighting calculations.

11.1.1.3 Rendering as Visibility Determination and Shading

The image produced by a 3D rendering engine is typically intended to mimic the output of a real-world camera. A digital camera contains an array of light sensors; the intensity and spectral characteristics of the light detected by each sensor determines the color of the corresponding pixel in the image. Most of the photons received by one of these sensors will have bounced off the exterior surface of some object in the scene prior to arriving at the light sensor.

Thinking about the rendering problem in this way permits us to break it down into two sub-problems:

1. *The visibility problem.* The rendering engine needs to determine which objects in the scene can be "seen" by the camera. Assuming our camera contains N light sensors (and thus generates an output image comprised of N pixels), the solution to the visibility problem provides the rendering engine with a collection of