Preface

Welcome to the second edition of *Computer Graphics Through OpenGL: From Theory to Experiments*! The first edition appeared in late 2010. In the nearly four years since, I have been fortunate enough to have received much thoughtful and, mostly, positive feedback. Happily, too, there was a fair bit of reassurance that my way of doing things, somewhat different from my peers', was on the right track. And, of course, the field of computer graphics as always has been evolving rapidly, of a particular impact being the maturing of the fourth generation of OpenGL. The upshot was that about a year and a half ago I began working on a new edition and am glad now that the finished text is in your hands. Let's get to the facts.

About the Book

This is an introductory textbook on computer graphics with equal emphasis on theory and practice. The programming language used is C++, with OpenGL as the graphics API, which means calls are made to the OpenGL library from C++ programs. OpenGL is taught from scratch.

After Chapters 1-14 – the undergraduate core of the book – the reader will have a good grasp of the concepts underpinning 3D computer graphics, as well as an ability to code fairly sophisticated 3D scenes and animation, including games and movies. With, additionally, Chapters 20-21, which can, in fact, be read following Chapter 13, she will have command over fourth-generation OpenGL, particularly version 4.3. Chapters 15-19, though advanced, but still mainstream, could be selected topics for an undergraduate course or part of a second course.

The book has been written to be used as a textbook for a first college course, as well as for self-study.

Specs

This book, comprising 21 chapters, comes with approximately 170 programs, 250 experiments based on these programs, 650 exercises, including theory and programming exercises and programming projects, 100 worked examples, and 600 four-color illustrations. The book was typeset using LaTeX and figures drawn in Adobe Illustrator.

New in the Second Edition

- 30 more programs, 50 more experiments, 50 more exercises
- Vertex buffer objects
- Vertex array objects
- Occlusion culling
- Occlusion queries and conditional rendering
- Texture matrices
- Multitexturing and texture combining
- Multisampling
- Point sprites
- Image and pixel manipulation
- Pixel buffer objects
- Shadow mapping
- OpenGL 4.3, shaders and the programmable pipeline:
  - Complete coverage over two chapters
  - OpenGL Shading Language (GLSL)
  - Vertex, fragment, tessellation and geometry shaders
  - From basic methods, such as animation, lighting and textures, to advanced topics, including instanced rendering, shader subroutines, transform feedback, texture buffer objects, several others
  - 19 example programs

Pedagogical Approach

Code and theory have been intertwined as far as possible in what may be called a discuss-experiment-repeat loop: often, following a theoretical discussion, the reader is asked to perform validating experiments (run code, that is); sometimes, too, the other way around, an experiment is followed by an explanation of what is observed. It's kind of like discovering physics.

Why use an API?
Needless to say, I am not a believer in an API-agnostic approach to teaching CG, where focus is on principles only, with no programming practice.

Undergrads, typically, love to code and make things happen, so there is little justification to denying the new student the joy of creating scenes, movies and games, not to mention the pride of achievement. And, why not leverage the way code and theory reinforce one another when teaching the subject, or learning on one's own, when one can? Would you want Physics 101 without a lab section?

Moreover, OpenGL is very well-designed and the learning curve short enough to fully integrate into a first CG course. And, it is supported on every OS platform with drivers for almost every graphics card on the market; so, in fact, OpenGL is there to use for anyone who cares to.

Note to student: Our pedagogical style means that for most parts of the book you want a computer handy to run experiments. So, if you are going to snuggle up with it at night, make it a threesome with a notebook.

Note to instructor: Lectures on most topics – both of the theory and programming practice – are best based around the book's experiments, as well as those you develop yourself. The *Experimenter* teaching resource makes this convenient. Slides, otherwise, are rarely necessary.

How to teach modern shader-based OpenGL?

Our point of view needs careful explanation as it is different from some of our peers'. Firstly, to push the physics analogy one more time, even though relativistic mechanics seems to rule the universe, in the classroom one might prefer doing classical physics before relativity theory.

Shaders, which are the programmable parts of the modern OpenGL pipeline, add great flexibility and power. But, so too, do they add a fair bit of complexity – even a cursory comparison of our very first program `square.cpp` from Chapter 2 with its equivalent in OpenGL 4.3, `squareShaderized.cpp` complemented with a vertex and a fragment shader in Chapter 20, should convince the reader of this.

Consider more carefully, say, a vertex shader. It must compute the position coordinates of a vertex, taking into account all transformations, both modelview – such as translation, rotation, scaling and viewing – and projection. In the classical fixed-function pipeline, the user can simply issue commands such as `glTranslatef()`, `glRotatef()`, etc., leaving to OpenGL actual computation of the transformed coordinates; not so for the programmable pipeline, where the reader must write herself all the needed matrix operations in the vertex shader.

We firmly believe that the new student is best served learning first how to transform objects according to an understanding of simply how a scene comes together *physically* (e.g., a ball falls to the ground, a robot arm bends at the elbow, etc.) with the help of ready-to-use commands like `glTranslatef()`, and, only later, the actual mathematics behind them.

Such consideration applies as well to other automatic services of the fixed-function pipeline which allow the student to focus on phenomena, disregarding *initially* implementation. For example, as an instructor, I would much prefer to teach first how diffuse light lends three-dimensionality, specular light highlights, and so on, gently motivating Phong's lighting equation, leaving OpenGL to grapple with its actual implementation, which is exactly what we do in Chapter 11.

In fact, we find an understanding of the fixed-function pipeline makes the subsequent learning of the programmable one easier because it's then clear exactly what the shaders should try to accomplish. For example, following the fixed-function groundwork in Chapter 11, writing shaders to implement Phong lighting, as we do in Chapter 20, is near trivial.

We take a similarly laissez-faire attitude to classical OpenGL syntax. So long as it eases the learning curve we'll put up with it. Take for example the following snippet from our very first program `square.cpp`:

```
glBegin(GL_POLYGON);
   glVertex3f(20.0, 20.0, 0.0);
   glVertex3f(80.0, 20.0, 0.0);
   glVertex3f(80.0, 80.0, 0.0);
   glVertex3f(20.0, 80.0, 0.0);
glEnd();
```

Does it not scream square – even though it's immediate mode and uses the discarded polygon primitive? So, we prefer this for our first lesson, avoiding thus the distraction of a vertex array and the call `glDrawArrays(GL_TRIANGLE_STRIP, 0, 4)`, as in the 4.3-program `squareShaderized.cpp`, our goal on Day 1 being a simple introduction of the synthetic camera model.

Of course, as we move along, we introduce each modern construct in its logical place, but with an eye always toward the overall learning process. For example, we introduce vertex arrays and their drawing commands in Chapter 3 on OpenGL gadgets, from then on making a point of using them, except for objects with few vertices when the overhead seems more distraction than convenience. Vertex buffer objects (VBOs) and vertex array objects (VAOs) are introduced in Chapter 3, as well, following logically vertex arrays; however, we counsel the reader against using them, until she gets to OpenGL 4.3, where they are mandatory, because they add a layer of coding complexity one can very well do without when learning fundamental concepts.

Does this kind of staggered introduction to modern OpenGL, with the old still around, not lead to bad practice? Not at all from our experience. When push comes to shove, how hard is to replace polygons with triangle strips? Or, for that matter, use VBOs and VAOs to store data? In fact, as we remarked earlier, grasp of the old motivates the step up to the new (there's virtue it seems then in retracing the path of the graybeards!).

So, practically, our code is backward-compatible OpenGL 4.3, which allows use of legacy syntax, for the first nineteen chapters. Then, Chapters 20-21, which together give a comprehensive coverage of OpenGL 4.3, use forward-compatible core OpenGL 4.3 (the strictest form).

The reader might note, as well, that OpenGL ES (Embedded Systems) 3.0, the latest OpenGL version for mobile devices, and WebGL, the emerging

3D standard supported by almost all the newer browsers, are syntax-wise very close to OpenGL 4.3, so assimilation of the latter means ability to code 3D graphics on multiple platforms.

On the other hand, there are millions of currently live applications written in legacy OpenGL, which are not going to be discarded or rewritten any time soon – the reason, in fact, for the Khronos Group to retain the compatibility version of the API – so familiarity with older syntax might well be useful for the intending professional.

Does our approach cost timewise? If the goal is OpenGL 4.3, then, yes, it does take a bit more time, but not much. Chapters 20-21 can be read after Chapter 13; in fact, they can be taught in parallel with Chapters 11-13. So, a one-semester course can perfectly well cover OpenGL 4.3. We discuss various possible learning sequences through the book later on in the preface.

Target Audience

- Students in a first university CG course, typically offered by a CS department at a junior/senior level (though, often, graduate students can take it for credit). This is the primary audience for which the book was written.
- Students in a second or advanced CG course, who may use the book as preparation or reference, depending on the goals. For example, the book would be a useful reference for a study of 3D design – particularly, Bézier, B-spline and NURBS theory – and of projective transformations and their applications to CG.
- Students in a non-traditional setting, e.g., studying alone or in a short course or an on-line program. The author has tried to be especially considerate of the reader on her own.
- Professional programmers, to use the book as a reference.

Prerequisites

Zero knowledge of computer graphics is presumed. However, the student is expected to know the following:

(1) Basic C++ programming. There is no need to be an expert programmer. The C++ program serves mainly as an environment for the OpenGL calls, so there's rarely need for fancy footwork in the C++ part itself.

(2) Basic math. This includes coordinate geometry, trigonometry and linear algebra, all at college first-course level (or, even strong high school in some cases). For intended readers of the book who may be unsure of their math preparation, we have a self-test in Appendix B, with solutions in Appendix C. The test should tell exactly how ready you are and where the weaknesses are.

Resources

The following are available through the book's website `www.sumantaguha.com`:

- Program source code – which runs on Windows, Mac OS and Linux platforms. The programs are arranged chapter-wise in the top-level folder `ExperimenterSource`.
- Guide to installing OpenGL and running the programs.
- Multiplatform *Experimenter* software to help run the experiments – whose interface is a pdf file containing all the experiments from the book, each being clickable to bring up the related program and, in a Windows environment, the workspace as well. *Experimenter* is only an aid and not mandatory – each program is stand-alone. However, it is the most convenient way to run the book's code, and instructors are strongly encouraged to use it.
- Book figures in `jpg` format arranged in sequence as one PowerPoint presentation per chapter.
- Instructor's manual with solutions to 100 problems (only for instructors who have adopted this textbook).
- Contributory resource bank with homework and examination questions, experiments and other teaching and learning aids.
- Other resources as they are developed (suggestions welcome).

Capsule Chapter Descriptions

Part I: Hello World

Chapter 1: An Invitation to Computer Graphics

A non-technical introduction to the field of computer graphics.

Chapter 2: On to OpenGL and 3D Computer Graphics

Begins the technical part of the book. It introduces OpenGL and fundamental principles of 3D CG.

Part II: Tricks of the Trade

Chapter 3: An OpenGL Toolbox

Describes a collection of OpenGL programming devices, including vertex arrays, vertex buffer and array objects, mouse and key interaction, pop-up menus, and several more.

Part III: Movers and Shapers

Chapter 4: Transformation, Animation and Viewing

Introduces the theory and programming of animation and the virtual camera. Explains user interactivity via object selection. Foundational chapter for game and movie programming.

Chapter 5: Inside Animation: The Theory of Transformations

Presents the mathematical theory behind animation, particularly linear and affine transformations in 3D.

Chapter 6: Advanced Animation Techniques

Describes frustum culling, occlusion culling as well as orienting animation using both Euler angles and quaternions, techniques essential to programming games and busy scenes.

Part IV: Geometry for the Home Office

Chapter 7: Convexity and Interpolation

Explains the theory of convexity and the role it plays in interpolation, which is the procedure of spreading material properties from the vertices of a primitive to its interior.

Chapter 8: Triangulation

Describes how and why complex objects should be split into triangles for efficient rendering.

Chapter 9: Orientation

Describes how the orientation of a primitive is used to determine the side of it that the camera sees, and the importance of consistently orienting a collection of primitives making up a single object.

Part V: Making Things Up

Chapter 10: Modeling in 3D Space

Systematizes the principles of modeling both curves and surfaces, including Bézier and fractal. Foundational chapter for object design.

Part VI: Lights, Camera, Equation

Chapter 11: Color and Light

Explains the theory of light and material color, the interaction between the two, and describes how to program light and color in 3D scenes. Foundational chapter for scene design.

Chapter 12: Textures

Explains the theory of texturing and how to apply textures to objects.

Chapter 13: Special Visual Techniques

Describes a set of special techniques to enhance the visual quality of a scene, including, amongst others, blending, billboarding, aliasing and multisampling, stencil buffer methods, and image and pixel manipulation.

Part VII: Pixels, Pixels, Everywhere

Chapter 14: Raster Algorithms
Describes low-level rendering algorithms to determine the set of pixels on the screen corresponding to a line or a polygon.

Part VIII: Anatomy of Curves and Surfaces

Chapter 15: Bézier
Describes the theory and programming of Bézier primitives, including curves and surfaces.

Chapter 16: B-Spline
Describes the theory and programming of (polynomial) B-spline primitives, including curves and surfaces.

Chapter 17: Hermite
Introduces the basics of Hermite curves and surfaces.

Part IX: Well Projected

Chapter 18: Applications of Projective Spaces
Applies the theory of projective spaces to deduce the projection transformation in the graphics pipeline, following up with shadow mapping as a case study. Introduces rational Bézier and B-spline, particularly NURBS, theory and practice.

Part X: The Time is Pipe

Chapter 19: Fixed-Functionality Pipelines
Gives a detailed view of the synthetic-camera and ray-tracing pipelines and introduces radiosity.

Part XI: Rendering Pipe Dreams

Chapter 20: OpenGL 4.3, Shaders and the Programmable Pipeline: Liftoff
Introduces OpenGL 4.3, GLSL (OpenGL Shading Language) 4.3, and writing vertex and fragments shaders to program the pipeline, particularly to animate, light and apply textures.

Chapter 21: OpenGL 4.3, Shaders and the Programmable Pipeline: Escape Velocity
Continuing the previous chapter onto advanced OpenGL 4.3 topics, including, amongst others, instanced rendering, shader subroutines and transform feedback, as well as tessellation and geometry shaders.

Appendix A: Projective Spaces and Transformations

A CG-oriented introduction to the mathematics of projective spaces and transformations. Provides a complete theoretical background for Chapter 18 on applications of projective spaces.

Appendix B: Math Self-Test
A self-test to assess math readiness for intended readers.

Appendix C: Math Self-Test Solutions
Solutions for the preceding self-test.

Suggested Course Outlines

See the chapter dependencies in Figure 1.

(1) Undergraduate first CG course:

This course should be based on Chapters 1-14 + Chapters 20-21, though full coverage might be ambitious for one semester. Instructors may pick topics to emphasize or skip, depending on their goals for the course and the chapter dependence chart.

For example, for more practice and less theory, a possible sequence would be 1 → 2 → 3 → 4 → 6 (only frustum culling) → 7 → 8 → 9 → 10 (skip curve/surface theory) → 11 → 12 → 13 (→ 14, low-level raster algorithms are independent of the higher-level topics of the preceding chapters, and may be taught depending on time) → 20 → 21 (20-21, on OpenGL 4.3, can be taught in parallel with 11-13, with discussion of a topic using the fourth-generation pipeline following its discussion using the classical one, e.g., Section 20.5 on shader-based lighting following Section 11.7 deducing Phong's lighting equation).

Time permitting, selected topics may come from Chapter 5 (theory of transformations), Chapters 15-16 (Bézier and B-spline modeling, respectively, which should be taught in sequence), Chapter 17 (Hermite curves and surfaces), Chapter 18 (rational Bézier and NURBS modeling), and Chapter 19 (graphical pipelines, including the synthetic-camera and ray-tracing), which may be read independently of each other.

Note to instructor: The most effective teaching method with this book is to base discussion around experiments – both from the book and those you develop yourself. Our *Experimenter* software makes this especially convenient. Students should be involved in the experiments, running code simultaneously on their own machines in class. Minimize use of slides except, possibly, for the book figures; for your convenience these are available to download, arranged as one PowerPoint presentation per chapter.

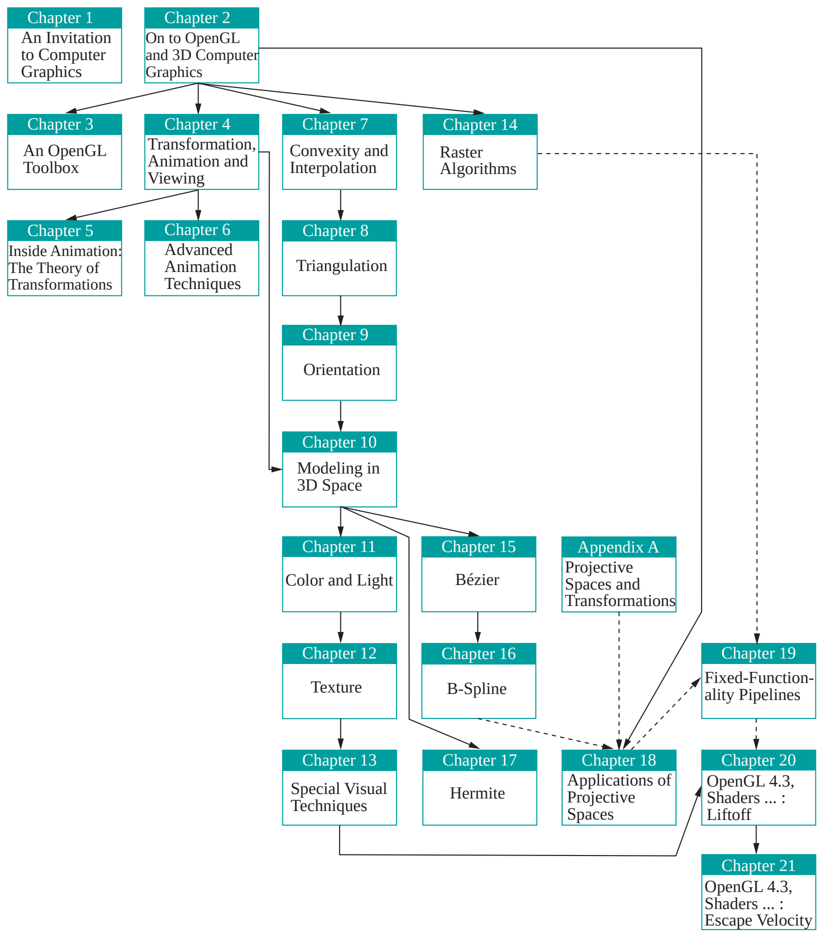

Figure 1: Chapter dependence chart: dashed arrows represent weak dependencies.

(2) Advanced CG courses:

This book could serve as a reference for a study of 3D design – particularly, Bézier (Chapter 15), B-spline (Chapter 16) and rational Bézier and NURBS theory (Chapter 18) – and of projective transformations and their applications (Appendix A and Chapter 18). From a practical point of view, Chapters 20-21 go fairly deep into the fourth generation of OpenGL and the GLSL, useful for students who may be familiar with only the classical pipeline.

(3) Self-study:

A recommended first pass would be 1 → 2 → 3 → 4 → 7 → 8 →

9 (go light on 7-9 if your math is rusty) → 10 (skip theory) → 11 → 12 → 13 → 20 → 21.

Following this the student should take up a fair-sized programming project, returning to the book as needed. For the theoretically-inclined student there's a lot to keep her busy in Chapters 5 and 15-19.

Code

All the book's programs, written in C++ with OpenGL, were developed in a Microsoft Visual Studio 2010 IDE running on Windows 7. However, they can run as well on Linux and Mac OS platforms with possibly some modification depending on the exact environment. The programs can be downloaded from `www.sumantaguha.com`, where they are arranged chapter-wise in the top-level folder `ExperimenterSource`. The reader will find there, as well, a guide to installing OpenGL and running the programs on various platforms.

Acknowledgments

I owe a lot to many people, most of all students whom I have had the privilege of teaching in my CG classes over the years at UW-Milwaukee and the Asian Institute of Technology.

I thank Tarun Mukherjee at Jadavpur University for being a constant source of inspiration, not to mention help with various technical questions.

I thank KV, Ichiro Suzuki, Glenn Wardius, Mahesh Kumar, Le Phu Binh, Maria Sell and, especially, Paul McNally, for their support at UWM, where I began to teach CG and learn OpenGL.

I am grateful to my colleagues and the staff and students at AIT for such a pleasant environment, which allowed me to combine teaching and research commitments with the writing of a book.

Particular thanks at AIT to Vu Dinh Van, Nguyen Duc Cong Song, Ahmed Waliullah Kazi, Hameedullah Kazi, Long Hoang, Songphon Klabwong, Robin Chanda, Sutipong Kiatpanichgij, Samitha Kumara, Somchok Sakjiraphong, Pyae Phyo Myint Soe, Adbulrahman Otman, Sushanta Paudyal (a summer's worth of help revising code), Akila de Silva, Nitchanun Saksinchai, Thee Thet Zun, Suwanna Xanthavanij, and our ever-helpful secretaries K. Siriporn and K. Tong.

I am grateful to Kumpee Teeravech, Kanit Tangkathach, Thanapoom Veeranitinun and Pongpon Nilaphruek, students in my CG course at AIT, for allowing me to use programs they wrote.

I thank Somying Pongpimol for her Illustrator drawings. She drew most of the figures for the first edition based on my original sketches done rather amateurishly in Xfig, and then revised several and created new ones for the current edition. Somying also designed the cover for both editions.

I would like to thank Olivier Nicole for revising the book's website for the new edition.

My special thanks to reader Denis Dalpé for an extensive list of typos.

I am especially grateful to Brian Barsky for encouraging me to persevere after seeing an early and awkward draft of the first edition, and subsequently inviting the book to his series. I want to acknowledge the production team at Taylor & Francis who went out of their way for this book. Particularly, I want to thank my editor Randi Cohen who has been simply tremendous to work with. I really appreciate her making so stress-free the "business" of publishing. I am grateful to Mimi Williams for her careful and expeditious proofreading.

I am grateful to the numerous reviewers and readers of the first edition whose comments helped immeasurably improve the current one.

I acknowledge the many persons and businesses who were kind enough to allow me to include images to which they own copyrights.

On a personal note, I express my deep gratitude to Dr. Anupam De for keeping Kamaladi healthy enough that I could concentrate on the first edition thorough the few years that I spent writing it.

By far my biggest debt of gratitude is to my student and friend Chansophea Chuon. Helping me with the first edition, Chansophea developed the LaTeX style sheet, supervised the drawings while doing several of the Illustrator figures himself, laid out the manuscript, developed the multi-platform program template, designed the *Experimenter* software to help run the book experiments and created the book's initial website, all the while putting out countless fires as they happened. Chansophea's layout, in particular, transformed a rather dowdy set of notes into a handsome four-color textbook. There is no doubt that, without Chansophea working shoulder to shoulder with me, I would not have finished even the first edition and this book would never have happened.

Finally, I must say that had I not had the opportunity to study computer science in the United States and teach there, I would never have reached a position where I could even contemplate writing a textbook. It's true, too, that had I not moved to Thailand, this book would never have begun to be written. This is an enchanting country with a strangely liberating and lightening effect – to which thousands of expats can attest – that encourages one to express oneself.

Website and Contact Information

The book's website is at `www.sumantaguha.com`. Users of the book will find there various resources, including downloads, and a few links specially for instructors. The author welcomes feedback, corrections and suggestions for improvement emailed to him at `sg@sumantaguha.com`.