Preface

Welcome to the fourth edition of *Computer Graphics Through OpenGL®: From Theory to Experiments*! The first edition appeared late 2010, the second four years after in 2014 and the third in early 2019. I was happy with what I had been able to put together by the third edition and did not think after it came out that I would be writing another one any time soon. However, a couple of factors caused me to change my mind.

Firstly, WebGL. WebGL is OpenGL for the web. WebGL has become the de facto standard for publishing 3D graphics to browsers and web developers are increasingly using it both for the wow factor as well as the utility of 3D gadgets over 2D. Soon after finishing the third edition, a personal project had me delve into WebGL, as well as into JavaScript, the "carrier" of WebGL, which I hadn't much familiarity with before. The experience was thoroughly enjoyable and put the thought in my mind of adding a chapter on WebGL. And I knew exactly how to go about this: a detailed exposition of pre-shader OpenGL had helped me present shader-based modern OpenGL efficiently; now, I would use the reader's grasp of shaders to lay out WebGL.

The second factor was feedback I got that the book needed to be streamlined. Readers mostly wanted a practical introduction to computer graphics (CG) using OpenGL as the application programming interface (API) – the keyword being *practical* – however, I had fairly substantial material on design theory and the mathematics of projective spaces. These theoretical topics are not irrelevant obviously to CG but could be distracting to a reader (or instructor) wanting a straight line from zero knowledge of CG to mastery of programming. I realized I could offer the best of both worlds. I would move the chapters on B-spline and Hermite theory and the lengthy appendix on projective spaces and transformations to the book's website for those interested. I would keep the chapter on Bézier design (as a simple enough introduction to mathematical modeling which a CG intro should have) and I would rewrite the chapter on applications of projective spaces (in particular, those that are indispensable to CG foundations) to be self-contained.

I started on this edition early 2021 with the two goals above in mind. But, as always happens when one revisits one's own writing after some amount of time, bits and pieces here and there began to catch my eye as improvable. Given the pandemic restrictions on most other activities, mission creep wasn't entirely unwelcome and I ended up with a fair revamp of the whole text.

About the Book

This is an introductory textbook on computer graphics with an emphasis on 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, while some familiarity with C++ is assumed.

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

After Chapters 1-17 – 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 3D scenes and animation, including games and movies. On the coding track, we begin with classical pre-shader OpenGL before proceeding to the modern OpenGL 4.x and concluding with WebGL 2.0 (do read the section explaining our pedagogical approach further on this preface). Chapters 18-20, though advanced, but still mainstream, could be selected topics for an undergraduate course.

Specs

This book comprises 20 chapters, as well as appendices containing a math self-test (for a reader to determine their preparedness) and its solutions. It comes with approximately 440 programs and experiments (programming tasks to help understand the main programs), 700 exercises, 100 worked examples, and 650 four-color illustrations, include drawings and screenshots. An instructor's manual containing solutions to selected exercises is available from the publisher. The book was typeset using LaTeX and figures drawn in Adobe Illustrator.

Target Audience

- Students in a first university CG course, typically offered by a CS department at a junior/senior level. This is the primary audience for which the book was written.
- Students in a non-traditional setting, e.g., studying alone or in a professional course or an on-line program. The author has tried to be especially considerate of the reader on her own.
- Professional programmers using the book as a reference.

Prerequisites

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

- 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.
- 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 A, with solutions in Appendix B. The test should tell exactly how ready you are and where weaknesses are.

Resources

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

- Sample chapters, table of contents, preface, subject and program index, math self-test and solutions.
- Program source code, developed on a Windows 10 platform using the Microsoft Visual Studio Community 2019 IDE, which should run on other versions of Windows/MSVS also, as well as 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 on a Windows/MSVS platform.

- Multiplatform *Experimenter* software to run the experiments. *Experimenter*'s interface is `Experimenter.pdf`, a file containing all the experiments from the book; except for those in Chapter 1, each is clickable to bring up the related program and workspace. *Experimenter* is only an aid and not mandatory – each program is stand-alone. However, it is the most convenient way to run experiments in their book order.
- Book figures in `jpg` format arranged in sequence as one PowerPoint presentation per chapter.
- Instructor's manual with solutions to 100 problems – instructors who have adopted the textbook can submit an online request.

Pedagogical Approach

Code and theory have been intertwined as far as possible in what may be called a theory-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 fan of the 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 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* resource makes this convenient. Slides other than the plentiful book figures, the latter all available on-line as PowerPoint files, 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 fourth-generation OpenGL, `squareShaderized.cpp`, complemented with a vertex and a fragment shader in Chapter 15, 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 and projection. However, in the classical fixed-function pipeline the user can simply issue commands such as `glTranslatef()`, `glFrustum()`, 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 `glRotatef()`, and, only later, how to define these transforms mathematically.

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 significantly 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 15, is near trivial.

We take a similarly laissez-faire attitude to classical OpenGL syntax. As 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 thereby the distraction of a vertex array and the call `glDrawArrays(GL_TRIANGLE_STRIP, 0, 4)`, as in the equivalent 4.x program `squareShaderized.cpp`, our goal being a simple introduction of the synthetic-camera model.

With these thoughts in mind the book starts in Chapter 2 with classical pre-shader OpenGL, progressing gradually deeper into the API, developing CG ideas in parallel, in a so-called theory-experiment-repeat loop. So, what exactly is an experiment? An experiment consists either of running a book program – each usually simple for the purpose of elucidating a single idea – or attempting to modify one based on an understanding of the theory in order, typically, to achieve a particular visual result.

By the end of Chapter 13 the student will have acquired proficiency in pre-shader OpenGL, a perfectly good API in itself. As well, equally importantly, she will have an understanding of CG principles and those underlying the OpenGL pipeline, which will dramatically ease her way through the concepts and syntax of OpenGL 4.x, the newest generation of the API, covered in Chapters 15-16. In fact, we continue the principle of leveraging the old to teach the new in Chapter 17, when we use the reader's understanding of shaders from OpenGL 4.x to introduce WebGL 2.0.

Does this kind of introduction to modern OpenGL via the old not ingrain bad habits? Not at all, from our experience. When push comes to shove, how hard is it to replace polygons with triangle strips? Or, use vertex buffer objects (VBOs) and vertex array objects (VAOs) to store data? Does our approach cost timewise? If the goal is OpenGL 4.x, then, yes, it does take somewhat longer, but there are various possible learning sequences through the book and 4.x certainly can be reached and covered in a semester.

In short, then, we believe the correct way to modern OpenGL is through the classical version of the API, because this allows the learning process to begin at a high level, so that the student can concentrate on gaining an overall end-to-end understanding of the CG pipeline first, leaving the OpenGL system to manage low-level processes (i.e., those inside the pipeline like setting transformation and projection matrices, defining fragment colors, and such). Once the student has a high-level mastery, subsequently "descending" into the pipeline to take charge of fixed-function parts in order to program them instead will, in fact, be far less arduous than if she tried to do both – learn the basics and program the pipeline – at the same time.

Another point to note in this context is that, as observed before, classical OpenGL is a complete API in itself which, in fact, can be more convenient for certain applications (e.g., it allows one access to the readymade GLUT objects like spheres and toruses). There are, as well, thousands of currently live applications written in classical 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 can be useful for the intending professional.

What about Vulkan?
We thought you might ask. Vulkan is the much-hyped "successor" to OpenGL. It is a highly explicit API, taking the programmer close to the hardware and asking her to specify almost all facets of the pipeline from end to end. Benefits of programming near the hardware include thin drivers, reduced run-time overhead and the ability to expose parallelism in the graphics processing unit (GPU). Vulkan is not only a 3D graphics API, but used to program GPU-heavy compute applications as well.

However, Vulkan's explicitness and consequent verbosity make it highly unsuitable as an introductory CG API. Here are some program sizes to begin with. The first OpenGL program in the book, `square.cpp`, which draws a black square on a white background, is about 90 lines of code in pre-shader 2nd generation OpenGL; a functionally equivalent program, `squareShaderized.cpp`, written in OpenGL 4.3 later on in the book is 190 lines plus 25 lines of shader code; a minimal equivalent Vulkan program, `squareVulkanized.cpp`, written separately by the author is 1,100 lines (no, that's no misprint – the reader will find the program at the book's website) plus 30 lines of shader code. Figure 1 is a screenshot.

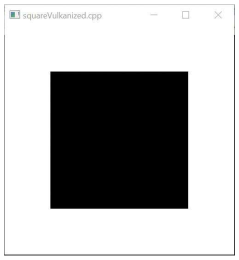

Figure 1: Screenshot of `squareVulkanized.cpp`, a 1000+ line Vulkan program.

Moreover, explicitness requires a Vulkan programmer to be familiar with the functioning of the graphics pipeline at a low level in order to specify it, which almost instantly disqualifies it from being a beginner's API. Further, delaying programming until after the pipeline has been covered goes utterly against our own pedagogical approach which is to engage students with code the first day.

So, is OpenGL, or for that matter, this book, of any use for someone intending to learn Vulkan? Well:

(a) The Vulkan graphics pipeline is essentially the same as OpenGL's. Therefore, learning OpenGL is progress toward Vulkan. Moreover, once a programmer has mastered OpenGL, she has most of what's needed to "take full charge" of the pipeline, which is what Vulkan is all about.

(b) The runtime gains of Vulkan don't begin to show up in any significant way until one gets to complex scenes with lots of textures, objects and animation. Less complicated applications - including, obviously, those in any introductory CG book - benefit little performance-wise from being written (or, rewritten) in Vulkan, not justifying the huge overhead in code.

This means that many OpenGL apps are going to stay that way and new ones continue to be written. It's a matter of knowing which tool to use: pre-shader OpenGL, OpenGL 4.x, Vulkan, ... (Hooking up a U-Haul trailer to the back of a Ferrari is never a good idea.)

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. Shows how to import objects from external design environments. 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 and render to a texture.

Chapter 13: Special Visual Techniques
Describes a set of special techniques to enhance the visual quality of a scene, including, among others, blending, billboarding, stencil buffer methods, image and pixel manipulation, cube mapping a skybox, and shadow mapping.

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: Programming Pipe Dreams

Chapter 15: OpenGL 4.3, Shaders and the Programmable Pipeline: Liftoff
Introduces 4th generation OpenGL and GLSL (OpenGL Shading Language) and how to vertex and fragments shaders to program the pipeline, particularly to animate, light and apply textures.

Chapter 16: OpenGL 4.3, Shaders and the Programmable Pipeline: Escape Velocity
Continuing onto advanced 4th generation OpenGL topics, including, among others, instanced rendering, shader subroutines, transform feedback, particle systems, as well as tessellation and geometry shaders.

Part IX: A Shader's Web

Chapter 17: WebGL
Introduces WebGL 2.0 through a series of programs covering features from the basic to the advanced.

Part X: Anatomy of Curves and Surfaces

Chapter 18: Bézier
Describes the theory and programming of Bézier primitives, including curves and surfaces.

Part XI: Well Projected

Chapter 19: Applications of Projective Spaces: OpenGL's Projection Transformations
Applies the theory of projective spaces and their transformations to deduce the projection transformations in the graphics pipeline.

Part XII: Time for a Pipe

Chapter 20: Pipeline Operation
Gives a detailed view of the synthetic-camera and ray-tracing pipelines and introduces radiosity.

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

Appendix B: Math Self-Test Solutions
Solutions for the math self-test.

Color Coding of Section Titles

As said earlier a motivation behind this edition is to streamline a reader's way through the material. In this spirit section titles have been color coded (in the section itself, not in the contents). The default color is green which indicates matter which should be read. Orange suggests that the topic is optional in that it can be skipped on a first reading, while red means advanced, often because the topic is somewhat mathematical. Of course, all topics are relevant to the subject at hand, or they would not be there, but we are just trying to give the readers options in case they wish to prioritize.

Incidentally, the color (and stylized fonts) of "Exercise", "Experiment", "Remark", etc., do not mean anything as such. The reason for making them look different is that there are so many that we wanted each to separately catch the eye of the reader flipping pages.

Chapter Dependences

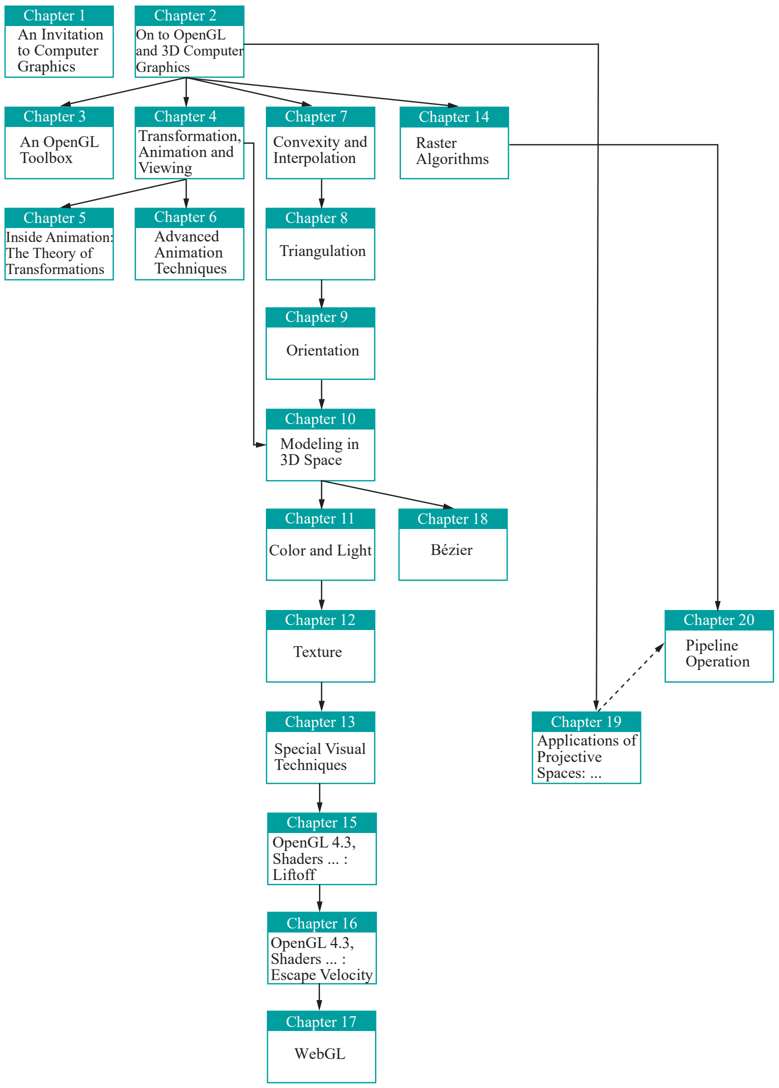
Chapter 1
An Invitation to Computer Graphics
Chapter 2
On to OpenGL and 3D Computer Graphics
Chapter 3
An OpenGL Toolbox
Chapter 4
Transformation, Animation and Viewing
Chapter 7
Convexity and Interpolation
Chapter 14
Raster Algorithms
Chapter 5
Inside Animation: The Theory of Transformations
Chapter 6
Advanced Animation Techniques
Chapter 8
Triangulation
Chapter 9
Orientation
Chapter 10
Modeling in 3D Space
Chapter 11
Color and Light
Chapter 18
Bézier
Chapter 12
Texture
Chapter 20
Pipeline Operation
Chapter 13
Special Visual Techniques
Chapter 19
Applications of Projective Spaces: ...
Chapter 15
OpenGL 4.3, Shaders ... : Liftoff
Chapter 16
OpenGL 4.3, Shaders ... : Escape Velocity
Chapter 17
WebGL

Figure 2: Chapter dependence chart.

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 (UWM) and then the Asian Institute of Technology (AIT).

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 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, Abdulrahman Otman, Sushanta Paudyal, Akila de Silva, Nitchanun Saksinchai, Thee Thet Zun, Suwanna Xanthavanij, Visutr Boonnateephisit, and our ever-helpful secretaries K. Siriporn and K. Tong.

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

I owe an enormous debt of gratitude to my former student Chansophea Chuon for hundreds of hours of help with the first edition, which got this book off the ground in the first place. I thank Somying Pongpimol for her brilliant Illustrator drawings. She drew several of the figures based on my rather amateurish original Xfig sketches.

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 finished product to the series he was then editing. Relatedly, I thank my two-floors-down neighbor Doug Cooper for putting me in touch with Brian at the time Brian was scouting prospective authors.

I want to acknowledge the team at Taylor & Francis, particularly Randi Cohen and Elliott Morsia, who went out of their way for this book.

I am grateful to readers of the first three editions, as well as reviewers who looked over drafts and proposals, whose comments led, hopefully, to significant improvements.

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 through the few years that I spent writing it.

Website and Contact Information

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