



Introduction

Almost every laptop and desktop computer sold now has both a high-resolution screen and a mouse. Inside these systems there is usually a central processing unit (CPU) as well as a graphics processing unit (GPU). The GPU is of similar overall complexity to the CPU, but it has a very different overall architecture and purpose. Most of a programmer's education typically deals with the CPU and text and file input/output. This book, on the other hand, deals with the screen, the mouse, and the GPU. These are the essential entities that computer graphics programs control. Much of "standard" programming applies in graphics, but there are many additional things to be mastered by a graphics programmer. This book deals with those additional things.

The rest of this book develops graphics programs one step at a time and in full detail. This is not a book to sit and read and ponder, as the many mundane details presented are quite boring. However, they are vital details for actually doing graphics programming. So, as you read the book, test what you learn by doing. By the end of the book, you should have in your skill set the ability to write a simple—even complex—application. We begin with handling user input to the program, displaying 2D graphics, and then harnessing the full 3D power of the GPU.

Rather than discussing graphics at a high level, we will dive right in and begin programming starting with Chapter 2. Thus the rest of this introduction discusses some details to be aware of when using the book. These are important nuts and bolts and should not be skipped!



Although foundation concepts are covered, this book is the *how to* of computer graphics, rather than the more traditional *what is* text that usually covers computer graphics. More specifically, we concentrate on *how to* build interactive computer graphics applications. The book ensures sufficient foundation concepts are covered so that readers can begin coding interactive graphics applications as they read the text, thus providing readers with proper preparation in order to continue learning about foundations of computer graphics after finishing this book. As far as this book is concerned, an *application* is a *computer program*. We interchange the use of *application* and *program*.

The Tutorials

The presentation of the material in this book is tightly integrated with example programs. We refer to these examples as *tutorials*, because they are designed to teach and demonstrate the implementation of concepts. For each topic area, we typically study the concepts followed by discussion of implementations with source code snippets. Since the topic areas build on top of each other, so does the source code for the tutorials. Initially, the tutorials are bare-bones with a handful of files; as we learn more concepts, the complexity of the tutorials will also increase. Toward the end, the tutorials will be moderately elaborate software systems with thousands of lines of source code. It is important that you follow and understand the source code as you read the text. We have found that one of the best ways to follow this book is by practicing graphics program development based on the provided source code. In general, it may be difficult to understand this book without a good understanding of the tutorials.

Organization of the Tutorials

The tutorials and source code can be found on the accompanying CD.⁵ The tutorials for each chapter are grouped under separate folders with the chapter being the name of the folder. Inside each chapter folder, the source code for each tutorial is separated into subfolders, with the tutorial's number and page number being the sub-folder's name. The final folder is named after the API and the programming language that the tutorial is implemented in. For example, Chapter 10 Tutorial 10.1 is first introduced on page 258, and there are two implementations

⁵Also available online at: <http://depts.washington.edu/cmmr/bigal/>.



of this tutorial: OpenGL and D3D, both based on C++ and MFC. The source code for this tutorial is located at

```
chapter_tutorials/Chapter_10/Tutorial_10.1_OnPage258/C++_D3D_MFC/  
chapter_tutorials/Chapter_10/Tutorial_10.1_OnPage258/C++_0GL_MFC/
```

We are currently working on

```
chapter_tutorials/Chapter_10/Tutorial_10.1_OnPage258/C#_XNA_UWBGUI/
```

IDE and APIs

We have chosen the Microsoft Visual Studio 2005 integrated development environment (IDE) as our target development environment. All tutorials are provided as separate Visual Studio projects. For this first edition of the book, we do not have support for any other IDEs.

The source code for the tutorials is in C++. We have developed two versions of most tutorials: one based on the OpenGL and the other based on the Microsoft DirectX Direct3D (D3D) graphics application programming interface (API). For now, the graphical user interface (GUI) library of most tutorials is based on the Microsoft Foundation Classes (MFC).

The choices for IDE, programming language, and libraries are based on the belief that these are probably the most widely available and most familiar environments to most of our readers. Our choices do not reflect endorsements for any of the systems we used. In fact, we believe that the concepts covered in this book, including implementation and specific skills, are independent of any particular software system. One should be able to port the learning from this book into any program or programming language and develop moderately complex interactive graphics programs, based on any modern IDE, object-oriented programming language, graphics library, and/or GUI library.

Our Libraries and Naming Conventions

In Chapter 2 we introduce a library to help abstract and customize the MFC library. In Chapter 4 we introduce another library to customize and interface to the OpenGL and D3D graphics libraries. From Chapter 5 onwards, we begin to work with the UWBGGL libraries. We will work with one GUI library and two graphics libraries. The libraries have names of the form

UWBGGL_API_Type_LibVersion_Number

UWB and UWBGGL. UWB is the abbreviation for University of Washington, Bothell, the home institution of the first author where most of the design and initial development efforts took place. “UWBGGL” stands for “UWB Graphics Library.”



where **API_Type** can be

MFC—Microsoft Foundation Classes (GUI library)

D3D—Microsoft DirectX Direct3D (graphics library)

OpenGL—OpenGL (graphics library)

and **Version_Number** is an integer. For example, `UWBGL_D3D_Lib1` is the first D3D library, and `UWBGL_OPENGL_Lib1` is the first OpenGL library we will develop. As we learn more concepts, we will continue to introduce new classes into each library, and the **Version_Number** of the corresponding library will increase.

We have established the following naming conventions to avoid collisions of identifiers.

- **Files.** Files have names that begin with `uwbgl_API_Type`. For example, `uwbgl_D3DCircleGeom.h` or `uwbgl_OPENGLCircleGeom.h`. For files that contain API-independent code, there will be no corresponding `API_Type`. For example, `uwbgl_Utility.h` declares the utility functions (e.g., the random number functions) for the library.
- **File versions.** As we refine concepts, the contents of many files (e.g., classes) will continue to evolve. A *file version number* is attached to files to avoid confusion (e.g., `uwbgl_D3DCircleGeom1.h` and `uwbgl_D3DCircleGeom2.h`). Notice that the file's version number is independent of the library's version number. For example, when evolving from `UWBGL_D3D_Lib1` to `UWBGL_D3D_Lib2`, the file `uwbgl_D3DGraphicsSystem1.h` did not change. The same file version 1 file exists in both versions of the libraries.
- **Classes.** Classes have names that begin with the following.
 - **UWB.** General classes that are common to all Graphics APIs (OpenGL and D3D). These classes typically define pure virtual interfaces that specify functionality. For example, we will discuss the `UWB_WindowHandler` interface to the windowing system.
 - **UWBD3D.** Classes that customize and build on top of D3D functionality. For example, the `UWBD3D_CircleGeometry` class is built on top of D3D drawing routines and encapsulates the functionality of a circle.
 - **UWBOpenGL.** Classes that customize and build on top of OpenGL functionality. For example, the `UWBOpenGL_CircleGeometry` class is built on top of OpenGL drawing routines and encapsulates the functionality of a circle.



- UWMFC. Classes that customize and build on top of MFC functionality. For example, the `UWMFC_UIWindow` class extends the MFC window class with customized support for our graphics applications.
- **Utility functions.** General utility function names begin with `UWB`. For example, `UWB_RandomNumber()` returns a random number.

When necessary, we refer to each class/file by its entire identifier string (e.g., `UWB_RandomNumber()`). However, for readability, we refer to the names without the “`UWB_`” prefix, (e.g., `RandomNumber()` instead of `UWB_RandomNumber()`).

Appendix C summarizes the changes of the library, when necessary, with associated static class UML diagrams. As new libraries are introduced in the text, refer to this appendix.

Library Design Trade-Off

The UWBGL library is designed to present graphics concepts. Although we are concerned with efficiency issues, execution speed is always of lower priority when compared with logical organization for presenting graphics concepts. For example, our simple organization of resetting/setting all attributes before drawing each primitive necessarily means that the graphics rendering context must be updated many times during each redraw. The alternative of collecting similar primitives and setting the attributes once for all these primitives during redraw would be much more efficient. However, such organization would introduce complexity independent of graphics concepts. In all cases, we opted for simplicity at the expense of efficiency in order to provide clear and simple presentation of graphics concepts.

Learning with the Provided Source Code

While reading this book, it is absolutely important to constantly remember that the elaborated source code is provided as a tool for learning and demonstrating the underlying concepts. We believe it is easier to grasp the foundations if readers can experience and experiment with the concepts in moderately complex environments. As stated, the other goal of this book is for readers to begin to develop moderately complex interactive computer graphics applications. The source is provided as a demo.



There are different levels of using the source code and the elaborate libraries provided. Readers are encouraged to be:

- **Users of the source code.** Readers should be encouraged to start from Chapter 2, where we demonstrate how to be a programmer of and use the source code provided to experience concepts and develop applications. You don't need to understand every single line of the provided code to be a proficient user of the library and to program something complex as well as experience and learn the underlying computer graphics concepts.
- **Evaluators of the source code.** For those who can understand the source system, we encourage you to critically evaluate our solutions to implementing the graphics concepts. We believe we have a pretty good solution, but as in all design problems, there are no perfect answers. For example, we believe the `IWindowHandler` hierarchy (introduced in Chapter 5) is a pretty solution for abstracting graphics and GUI APIs while supporting view/controller functionality. However, we certainly understand that this is not a perfect solution, and readers are encouraged to evaluate the merits of our solution and to seek out alternative ideas.
- **Developers of their own systems.** From our experience using this source system in the classroom, the best students often challenge the provided source code and develop their own libraries while using the provided source as an example. Many students choose to implement their systems based on different programming languages with different APIs, for example, Java, Java Swing, and OpenGL, C++, MFC, and OpenGL, C#, WinForm, and XNA Framework, and so on.

In all cases, the most important point we hope to make is that readers should avoid getting bogged down with our implementation and being unable to continue learning as a result of our source code system. That would be the worst-case scenario. Always remember: find ways to use the source code and examples to experience and experiment with the concepts. You don't have to like the implementation. In fact, you are welcome to hate it and critically evaluate it; just don't allow your attitude toward the source code to hinder your learning of the subject matter.