A Top-Down Approach to Teaching Introductory Computer Graphics

Kelvin Sung
Computing and Software Systems
University of Washington, Bothell

Peter Shirley
School of Computing
University of Utah

Abstract

There are two common strategies for teaching introductory computer graphics (CG) programming. The first and most traditional covers the CG field in a bottom-up manner starting from foundational algorithms such as triangle rasterization. The second is top-down and analyzes the functional modules of applications. This paper argues that the top-down approach is well-suited for mature adult students. A course that has successfully implemented a top-down approach is then described.

1 Introduction

The syllabus of a course represents the *design* of a solution for teaching the materials involved. In this way, we can examine the different approaches to teaching Introductory Computer Graphics (CG) based on the classic Structured System Design Methodologies [1]. In particular, we can analyze the syllabi of introductory CG programming courses based on *bottom-up* and *top-down* design models.

Traditional introductory CG courses (e.g. [2]) typically follow the classical CG textbooks (e.g. [3]**)** and usually begin by introducing and surveying the CG field as a whole. These courses then identify the important foundational building blocks (e.g. raster-level algorithms, transformations, etc.) in modern CG systems, and move on to study each of the building blocks in detail. These courses present students most of the foundational building blocks of modern CG systems and students gain knowledge of what is "under the hood" of modern graphics coprocessors. This bottom-up approach is well-suited for traditional undergraduate students. The in-depth coverage of basic algorithms not only prepares them for future and more advanced courses; it also serves the important role of demonstrating problem solving approach and formulation of solutions. For these students the detailed study of matrix transformations are examples of applications of the covered mathematic skills. These students are then equipped with the understanding of the basics of the field. They typically have technical knowledge of the underlying implementation of the basic components in popular graphics Application Programming Interface (*API*) libraries (e.g. *OpenGL* [4], or *DirectX-3D* [5]). In addition, it is possible for the more advanced students to apply some of these basic ideas in more advanced areas.

While the strength of the bottom-up approach is that it teaches the basic mathematics and methodology of graphics engine design, in the near term it does not enable students to use a powerful graphics API to design complex applications and thus may seem to lack practical impact. For traditional undergraduate students this may not be a serious problem for they have the time and opportunity to apply this knowledge in their future classes and/or career development. However, for more mature students in their mid-career, the near-term relevancy of low-level algorithms and mathematics derivations becomes a serious question. For example, after a bottom-up CG course, a student will look at 3D applications like Maya [6] and be able to appreciate that the DDA line drawing algorithm is part of the foundation building block. However, this student will also notice the complexity of the software system, and wonder if time may be better spent learning the higher level and more visible functionality.

The alternate approach of using an example application to explain CG issues is a natural way of addressing the above concerns, in other words, a top-down approach. Three years ago such a top-down course was started at the University of Washington, Bothell [7] where many of the students are more mature. This consisted of two 10-week courses based on interactive graphics application development. These courses attempted to balance fulfilling the students' expectations and covering sufficient basic concepts to assist students in future self-learning. This paper concentrates on the first of these two courses where students are introduced to 2D interactive graphics programming. The second course is similar to the first except concepts are extended to the third dimension. This paper begins by discussing the idea behind top-down approach to teaching CG and identifying the reasons why this approach is well suited for the more mature adult students. An approach based on interactive graphic application development is then described. The paper concludes with what has been learned and how this curriculum should be modified for future offerings.

2 Top-Down Approach to Introducing CG

A top-down approach to teaching CG would identify a moderately complex application and decompose the system into *functional modules*. The course would then cover the modules while relating each module back to the target application. In this way, students learn the foundations and structure of graphics applications while practicing the more visible application-level knowledge and skills. Ideally the *functional modules* from the top-down approach should be continuously decomposed into smaller units until the units become the *foundational building blocks* identified in the bottom-up approach. However, given the sophistication of the modern graphics applications, this ideal decomposition process cannot be accomplished in a single term. This is the same reason a typical bottom-up CG course does not have time to complete a moderately complex graphics application. The API libraries can serve as a convenient convergence point for the two approaches. A top-down approach would teach students to implement functional modules based on the popular graphics APIs. Besides serving as a practical skills training, using an API extensively in building a moderately complex system helps students understand the design and appreciate the pros and cons of the API.

One difficulty in designing a top-down syllabus for introductory CG courses is identifying an appropriate *top*. The *top* in this context refers to a *target software system*. In the bottom-up approach, it is straightforward to identify the *bottom* as the basic building blocks of *general CG systems*. The key is that the basic building blocks are, in general, suitable for building any of these CG systems. In a top-down approach, the ``*target software system*'' must be well defined. It is important to identify a target

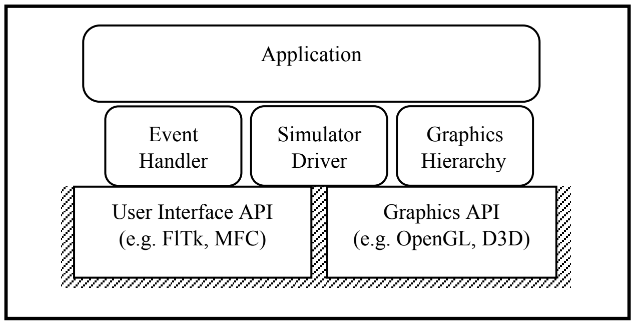
Application
Event Handler
Simulator Driver
Graphics Hierarchy
User Interface API (e.g. FlTk, MFC)
Graphics API (e.g. OpenGL, D3D)

Figure 1: Components in Interactive Applications

system that demands a sufficiently large set of common supporting requirements shared by many CG software systems.

The bottom-up and top-down approaches to solving the learning and teaching of introductory CG programming problem are almost exactly complementary. Given a time limit, the top-down approach trades the high-level system architecture understanding (e.g. event handling models, scene graph design and traversal approaches) for bottom-up's foundation knowledge (e.g. rasterization algorithms). Of course, the two approaches are not strictly mutually exclusive. For example, bottom-up approach often uses a simple target application framework for students to investigate the implementation of different algorithms. Whereas in top-down approach; it is possible to cover some low level basic algorithms. Course syllabus should be designed according to the expected student-learning outcome.

3 A Top-Down Introductory CG Course

The University of Washington, Bothell (UWB) was established in 1990 as an upper-division-only campus that offers junior, senior, and graduate level courses [7]. Situated in the midst of the Northwest's technology corridor, many of our students pursue the Computing and Software Systems (CSS) [8] degree for career transition and/or as a means to begin a career in the technology field. Our students typically hold full or part time jobs, have family obligations, and have somewhat rusty mathematic skills. They are also motivated and concerned with their near-term marketability. The materials covered in our courses are constantly under tension between the foundational scientific concepts and technical skills training

The past few offerings of *CSS450: Introduction to Computer Graphics* [9] approached the teaching of CG programming in a top-down manner. As in most of our courses, CSS450 is a 5-credit 10-week quarter course with about 200 minutes of lecture per week. With the understanding of our students' background and expectations, this introductory CG course is designed to spark students' interests in the field. The documented goals of this course are to analyze the components that are *under the hood* of popular interactive graphics applications (e.g. Power-point like and/or drawing/sketching programs); and to study these components such that students can design and implement such applications based on popular APIs. The *undocumented* and yet very important objectives are to *motivate* and to ensure the *coverage of sufficient conceptual knowledge* to facilitate students' future self learning.

These *undocumented* objectives form the main guidelines for the design of the class. To motivate and excite students (especially into polishing their rusty mathematic skills), all topics covered are evaluated against popular software that are familiar to students. In addition, all programming assignments in the class are

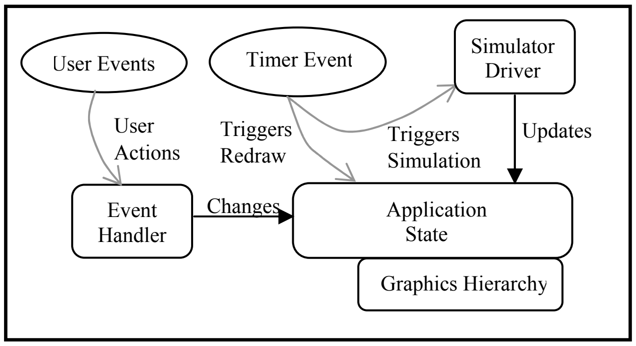

Figure 2: Application Architecture

disguised as interactive games development. To ensure the proper balance between conceptual knowledge and skill set training, all components covered are evaluated based on more than one API from different vendors. This is coupled with constant emphasis that the ideas learned are technology independent and students must appreciate that the concepts can be applied to any modern technologies.

One of the most important tasks in the top-down approach is the selection of an appropriate target software system to commence the analysis of functional modules. Based on the *motivation* guideline, the ''*popular interactive graphics application*'' is selected as the target application. We observed that most of the ''*popular interactive graphics applications*'' can be described as applications that allow users to interactively update their internal states. These applications provide real-time visualization of their internal states with the graphics subsystem. In addition, these applications typically support some mechanisms that allow the user to define simple animations. Figure 1 shows one way of decomposing these types of applications into major functional modules. In general, it can be assumed that the modules are implemented based on existing User Interface APIs (e.g. FlTk [10], or MFC [11]) and Graphics APIs (e.g. OpenGL, or DirectX-3D). Based on this framework, the syllabus of our introductory CG course becomes a mapping of the requirements to understand and implement these functional modules into specific topics in CG. Three general topic areas are identified: **Event and Simulator Driven Programming**, **Graphics API Abstraction**, and **Transformation**. These topics are scheduled in our course based on drawing from students' strength in software development. We begin with discussion/practicing of programming models (Event and Simulator Driven Programming**)**, followed by graphics hierarchy design/implementation (Graphics API Abstraction**)**, and finally topics in Transformation. These three topics are covered in the first 6 weeks' of class. At which point students commence to work on their final project. The remaining 4 weeks are divided between discussions of topics related to students' final project development (e.g. collision detection algorithms, texture mapping, etc.) and the fundamental algorithms in CG (e.g. color models, raster level scan conversions, etc). In the rest of this section, each of these topics is briefly discussed and summarized with how we approach in designing interactive-game-like programming assignments for students to practice and reinforce the concepts involved.

Event and Simulator Driven Programming. The prerequisites of CSS450 are advanced data structure classes. By the time students enroll in CSS450, they typically have extensive and polished basic programming skills working with data structures. These skills are usually confined to solving simple problems based on the internal control model [12]. Drawing on students' programming skills and practicing problem solving with the external control model of event driven programming is a good way to introduce them to the field of interactive graphics programming. Figure 2 depicts an implementation architecture based on the functional modules identified in Figure 1. In this case, the *Event Handler* module would support the interaction with the user for updating the state of the application; the *Simulator Driver* would trigger and run simulations (which would result in application state change and show up as animations); and the *Graphics Hierarchy* would support the visualization and/or displaying of the application state. Arguably, this topic is biased towards technical skill set training. The concepts of programming models and problem solving approaches can be enhanced by discussing how different User Interface APIs support the implementation of these ideas. Other reasons for introducing the course with this topic are that it is more straightforward to overcome the technical challenges involve; and the programming assignments are typically interesting interfaces that students can play with.

The programming assignment for this topic would be developing software that supports user actions in modifying and interacting with simple internal states. One example would be implementing a system that supports defining/drawing a circle with simple click and drag mouse actions. The real-time simulation module would be activated after the circle is defined. For example, a random velocity can be assigned to the circle such that the circle constantly moves on the screen. Finally the entire assignment can be disguised as a game where students have to implement interactive functionality to keep the circle in a specific area of the screen by colliding the circle with the mouse pointer.

Graphics API Abstraction. This topic is covered to achieve two major objectives. The first is to introduce and utilize the abstract graphics engine presented by popular Graphics APIs. Once again, the balance between technical skill training and basic conceptual knowledge is achieved by comparing and contrasting at least two of such abstractions (e.g. DirectX-3D vs. OpenGL). The second goal is to prepare students for large scale software development by demonstrating how to take advantage of object oriented design in the CG settings.

The programming assignment for this topic would be developing software that demonstrates how to interact with the behaviors of abstract graphics objects without knowledge of what actual primitives are being processed. For example, design a primitive drawing program based on the behaviors of an class, where simple primitives (e.g. points, lines, circles, etc.) can be defined interactively with the exact same interaction routines. Once again, the simulation and game playing modules would be activated after the primitives are defined. In this example, the program can be extended to support pushing (initiating a velocity based on mouse's collision) the primitives on the screen based on the class behaviors.

Transformation. By this time, students would have experience with developing moderately complex systems (e.g. several thousands of lines of C++ code) interacting with users drawing different types of graphics primitives. They would begin to realize the restrictions of working directly in the screen coordinate system and begin to appreciate the need to have more than one view into the world with zooming and panning functionality. These serve as motivations for introducing coordinate transformations. The coverage of coordinate transformation pipeline leads naturally to hierarchical modeling where the object coordinate space can be decomposed into coordinate spaces of each individual component. With the graphics hierarchy introduced in Figure 3, compound objects can be defined based on TobjectList composing of other TgraphicsObject primitives. Since TobjectList is itself a TgraphicsObject, it is straightforward to build homogenous list-of-list of TgraphicsObjects. This general homogeneous list-of-list serves as simple examples of scene graphs.

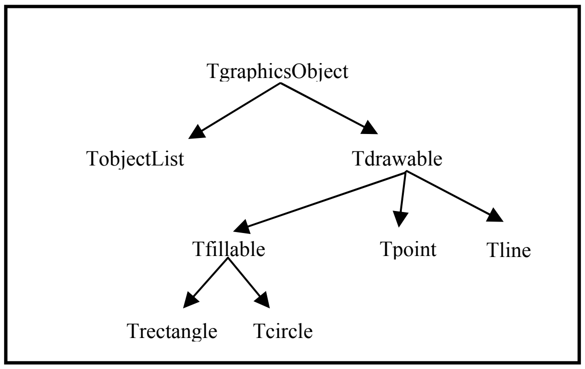

Figure 3: The Graphics Hierarchy

The programming assignment for this topic would be developing software with multiple views of compound objects based on simple primitives. The software must support general transformations of components in the compound object. For example, design a ''stick-figure human'' based on simple circles and rectangles where the software must support transforming (scale/rotate/translate) each body parts (e.g. a hand) individually and transforming compound body parts (e.g. the entire arm system, including the hand) as components. As in previous cases, the simulation and game modules would be separately activated to support real-time interaction with the user. With the ''stick-figure human'' example, game-like interaction could be based on user manipulating the various body parts via transformations. For example, the user can control the stick-figure to defend a goal post. The simulator would generate random shots toward the goal post, and the user must manipulate the arm system based on the transformation controls to fend off the incoming shots.

After covering the above three topic areas, the final project specification is handed to the students at the end of the 6th week. At this point, students have sufficient knowledge and practical skills to build a moderately complex interactive graphics system. However, students have a serious lack of knowledge when it comes to the foundation CG algorithms. Part of the last 4 weeks of classes is dedicated to the discussion of the more traditional fundamental topics. Topics covered include: color models, human vision system, raster level drawing and clipping algorithms, etc. With the time restrictions and students' attentions on their final project development, the coverage of these topics is necessarily less extensive than most traditional introductory CG courses. For example, there are no specific assignments designed for students to practice these algorithms. To help maintain students' interests and engagement in lecture, other more popular topics such as: texture mapping, alpha blending, and/or special effects with animated textures, etc. are also covered at an introductory level. The final project in this class is a 3 to 4 week giant assignment. Based on a set of strictly defined technical requirements, students are free to design any system of their choice. To assist students in meeting their schedules, each student must present user interface design and progress demonstrations to their peers. These presentations turned out to be the highlights of the course, where students show-off their ideas and constructively criticize/complement each other's work.

4 Evaluation

When comparing our approach to that described in Edward Angel's textbook: *Interactive Computer Graphics – a Top-Down Approach with OpenGL* [13], the top-down idea is the same. However, Angel chose a simple application as his target system for commencing the top-down analysis. This choice is to ensure that the functional modules in his target system will be the foundational algorithms. When comparing to our choice of ''*popular interactive graphics application*'' there is a vast difference in software system complexity. In addition, our objectives are much more limited in scope (to only 2D). Finally and very importantly, we have very different underlying approaches to conveying the concepts involved. While our approach stresses on API independence and the importance of application of the ideas to all APIs, Angel chose to exemplify the concepts involved exclusively with one popular graphics API.

We believe a limited success has been achieved in the three years' offering of this course with our new approach. We evaluate our achievement based on three criteria: enrollment in the class, quality of students' final project, and students' further interests after the completion of this course. This course is a free elective in our curriculum, and students only select this course out of interest. Over the pass three years, the overall student population in our department has remained somewhat constant and yet the enrollment of this course has risen from 10 in Fall 2000, to 20 students in Fall 2002. The quality of students' final projects is subject to interpretation. Two of the ways to analyze students' *further interests* are to examine the subsequent courses students take, and students career development after graduation. The follow-up course of CSS450 is *CSS 451: 3D Computer Graphics*. This year, 18 of the 20 students (90%) in CSS450 are registered for CSS451. Of the 20+ graduates from the previous CSS450 courses, 6 are currently working or interning at local graphics/games companies. Although these numbers are based on very small sample size, it does reflect limited success and show encouraging trend.

However, there are still many difficulties in implementing the presented top-down syllabus.

1. Text book. There is no CG text book that agrees with the described top-down approach. As a result, the course has been based on a few reference classic text books with constant extra in-class handouts. Although it is possible to convey the essence of the knowledge involved, the learning suffers from the lack of a continuous flow that can only be found in one text book.

2. Learning new tools. Students must learn fairly sophisticated new tools in a relatively short amount of time. For example, during the first week of Fall 2002 CSS450 offering, students are expected to learn sufficient MFC and Win32 programming by themselves to begin developing interactive systems. Although system manuals and on-line tutorials are very helpful, these are not designed for our course. The amount of information presented are typically too extensive in some areas and yet insufficient in others.

3. Large software system development. Our top-down analysis is based on ''*popular interactive graphics systems*''. This implies students' final projects are also such systems. Typically by the third week of the course, after the introduction of the graphics class hierarchy, students are working with more than 3000 lines of C++ code. It is always a challenge to balance between system development and learning of new CG concepts.

4. Fundamental CG algorithms. As described, the fundamental algorithms/issues in CG are covered in the latter part of the quarter. Because of the large scale programming assignment developments, the course is often behind schedule. As a result, the coverage of these fundamental topics can sometimes be hurried. In addition, the focus of the first 6 weeks and the last 4 weeks are quite different and the transition between the two parts is fairly difficult. After studying/implementing the application level issues and highly visible topics (e.g. texture mapping, particle systems, etc.) the foundation algorithms often appear extra tedious. Together with the deadline pressure from the final project, students often find it difficult to concentrate on learning these topics.

5. Source code documentation and version control. One important lesson we have learned is that our time and place bound students do not spend nearly as much time on campus. In addition, our students typically work on their programming assignments late at night, and/or over the weekends. These mean students do not have as much opportunity to interact with their peers and they often discover the lack of understanding when there is no one around to help them. Our solution for this situation is to provide a large number of examples (with source code) that illustrate the concepts discussed in lectures. For example, the Fall 2002 offering of CSS450 has about 40 different examples and the average size of these examples are about 2000 lines of C++ code. These examples are typically non-trivial, with the course schedule pressure; the source code is usually not well documented. In addition there is no version control to keep track of bug fixes.

Acknowledgements

Thanks to all CSS450 students for working so hard with the first author on constantly providing feedback, and to Charles Jackels for his encouragements and support. The detailed conceptualization of the top-down vs. bottom-up approaches is a result of countless discussion with Frank Cioch. It is great to have a colleague who disagrees and yet is willing to listen. Thanks also to Diane Gillpesie for organizing the Research Writing Circle at the UWB, in which Leslie Olsen and Cinnamon Hillyard were able to meet and provide valuable feedback to the first author throughout the writing of this paper.

References

[1] E. Yourdon, and L. Constantine, ''*Structured Design: Fundamentals of a Discipline of Computer Program and System Design*,'' Yourdon Press, 1979.
[2] http://www.cs.utah.edu/classes/cs5600/, March 2003.
[3] D. Hearn and P. Baker, ''*Computer Graphics – C Version*,'', second edition, Prentice Hall, 1997.
[4] M. Woo, J. Neider, and T. Davis, ''*Programming Guide, 2nd Edition*,'', Addison Wesley, 2001.
[5] http://msdn.microsoft.com/directx/, March 2003.
[6] Maya Unlimited, Alias|Wavefront, Toronto, Canada, http://www.aliaswavefront.com, 2002.
[7] http://bothell.washington.edu, March 2003.
[8] http://bothell.washington.edu/CSS, March 2003.
[9] http://courses.washington.edu/css450, Mach 2003.
[10] Home Page, FlTk, http://www.fltk.org, 2002.
[11] Jeff Prosise, ''*Programming Windows with MFC*,'' Second Edition, Microsoft Press, 1999.
[12] Brad Myer, ''*Separating Application Code From Toolkits: Eliminating the Spaghetti of call-backs,*'' Proceedings of the UIST'91, pp. 211-220, November 1991.
[13] Edward Angle, ''*Interactive Computer Graphics – a Top-Down Approach with OpenGL*,'' Second Edition, Addison Wesley, 2000.