A Top-Down Approach to Teaching Introductory Computer Graphics

Kelvin Sung
Computing and Software Systems
University of Washington, Bothell
ksung@u.washington.edu

Peter Shirley
School of Computing
University of Utah
shirley@cs.utah.edu

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 fundamental algorithms such as triangle rasterization. The second proceeds in a top-down manner by analyzing 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.

Keywords: Pedagogy, Top-Down, Pragmatic Approach, Programming, Adult Students

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 syllabus of introductory CG programming courses based on bottom-up and top-down design models.

Traditional introductory CG courses [2,3,4,5] typically follow the classical CG text books (e.g. [6,7,8,9]) 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. In the course of a term, these courses present to students most of the foundation 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 transformation and matrices are examples of applications of the recently acquired mathematic skills. After these types of introductory CG courses, students are equipped with the understanding of the basic foundations 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 [10], or DirectX-3D [11]). 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, this approach seldom addresses application-level issues and does not enable students to use a powerful graphics API to design a complex application. In the near term, the bottom-up approach may seem to lack practical impact. For the 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, where they have years of practical experiences; the near-term relevancy of detailed low-level algorithms and mathematics derivations becomes a serious question [12]. For example, after a bottom-up CG course, a student will look at 3D applications like Maya [13] 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 a moderately complex application as an example to explain the requirements for supporting/implementing the application 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 [14] where many of the students are more mature. This consisted of two 10-week quarter 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 background that motivated this work, 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 Background

The University of Washington, Bothell (UWB) was established in 1990 as an upper-division-only campus that offers junior, senior, and graduate level courses [14]. One of the main missions of the campus is to serve time and place bound[1] established adult students. Many of our students are working adults who recognize the importance of education in the current information-driven economy and return to pursue college degrees [15,16]. The Computing and Software Systems (CSS) degree program [17] at UWB is designed with the understanding that successful adult degree programs are not simple reorganizations of traditional undergraduate classes [18] and has a continuous and concerted effort in re-examining pedagogical and technical approaches to teaching software design and development (e.g. [19,20,21,22,23]).

When we taught the introductory CG course following the traditional textbooks (e.g. [6]), our students were disengaged. According to our student population, we have found the cause of the problems to be: **(1)** unclear relevancy of knowledge; **(2)** unrealistic hands-on exercises.

2.1. Relevancy of Knowledge

> ‘‘*while I appreciate that the DDA line drawing algorithm is part of the foundation building blocks of 3D applications like Maya [13], I also notice the complexity of the software system, and wonder if time may not be better spent learning other more interesting aspect of the graphics system*’’ – CSS450 Student, Fall 1999.

Among CG educators there is a collective understanding of what are the essential philosophy and concepts that define the field [5]. Constraint by the time limit in academic terms, traditional CG courses cover subsets of these essential concepts based on respective student learning outcomes (e.g. [24,25]). These approaches ([2,3,4,5])are characterized as bottom-up because they present the CG field focusing on the individual low-level foundation concepts [21]. In the near term this approach does not enable students to use a powerful graphics API to design complex applications and thus may seem to lack practical relevancy.

2.2. Practicality of Hands-on Exercise

> ‘‘*while it was interesting to practice multi-way symmetry in mid-point ellipse algorithm, I still do not understand how these exercises relate to the graphics programs I use everyday*’’ – CSS450 Student, Fall 1999.

This sentiment is shared by many educators. Kubitz suggested that the traditional (bottom-up) approach is ‘‘*mostly wrong*’’, that graphics courses should study higher level issues based on latest APIs [3]. Cunningham pointed out API-based CG courses that cover higher level issues are better equipped to reach a wider audience and thus are more suitable for students who only take one CG course in their undergraduate education [26,27,28]. To provide students with a more holistic understanding of the CG field, Lowther and Shene [29] traded breath for depth of coverage and described strategies to cover Rendering, Modeling, Animation, and Postprocessing in one introductory CG course. These are pragmatic bottom-up approaches because they concentrate on studying individual issues (*bottom*) of general CG applications (*up*) based on modern APIs.

2.3. Top-Down Approach to Introducing CG

A top-down approach to teaching CG turns the pragmatic bottom-up approach around by identifying a category of applications and decomposing the applications into *functional modules*. The course would then cover the modules while relating each module back to the target applications. 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 *essential concepts* identified in the bottom-up approach. However, given the sophistication of the modern graphics applications, this ideal decomposition process is non-trivial and typically cannot be accomplished in a 16 week semester (or 10 week quarter). This is the same reason why a typical bottom-up CG course does not have time to complete a moderately complex graphics application starting from the foundations. The popular graphics 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.

[1] ‘‘Time-bound’’ – Students working full/part time and thus cannot attend classes during day time. ‘‘Space-bound’’ – Students cannot afford to relocate to on-campus housing during the course of their university education.

Top-down is complementary to the traditional bottom-up approaches [30,31] because it trades high-level system architecture understanding (e.g. scene graph design) for low-level foundational knowledge (e.g. rasterization algorithms). Of course, the two approaches are not strictly mutually exclusive. For example, the bottom-up approach often uses a simple target application framework for students to investigate the implementation of different algorithms. Whereas, in the top-down approach, it is possible to cover some basic low-level algorithms. The course syllabus should be designed according to the expected student-learning outcome.

One of the difficulties in designing a top-down syllabus for introductory CG courses is in identifying an appropriate *top*. The *top* in this context refers to a *target software system*. As mentioned, in the bottom-up approach, CG educators have a collective understanding of what are the essential philosophy and concepts that defines the field [5]. These concepts are the basic building blocks of *general CG systems*. Whether it be a hardware CG system, a batch software CG system, or an interactive software CG system, etc. The key is that the basic building blocks are suitable for building any of these CG systems. In a top-down approach, where a *system* is decomposed for identifying the supporting requirements, the ''*target software system*'' must be well defined. It is important to identify a target system that demands a sufficiently large set of common supporting requirements shared by many CG software systems.

By identifying an appropriate *target software system*, syllabi based on the top-down approach can support courses suitable for a variety of contexts [32]. For example, elective and introductory CG programming courses can be designed based on moderate 2D applications; advanced CG courses can be designed based on complex 3D applications; while courses suitable for students from other science and engineering disciplines (e.g. [33,34]) can be designed based on simple interactive visualization applications.

As cautioned [26], courses based on popular applications (e.g. Maya) and/or APIs (e.g. Microsoft DirectX, OpenGL) run the risk of teaching students to be *users* rather than *practitioners* of CG. As an example, the top-down approach should identify major functional modules in Maya (e.g. user interaction module, or hierarchical modeling module) and examine how to design and implement these modules based on functionality supported by OpenGL or DirectX. This is different from *learning how to use Maya* [35], or *learning how to use specific APIs* (e.g. [10,36,37,38]). As newer versions of the software and/or APIs are released, the knowledge students gained must continued to be valid and applicable.

3 A Top-Down Introductory CG Course

The past few offerings of *CSS450: Introduction to Computer Graphics* [39] 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 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.

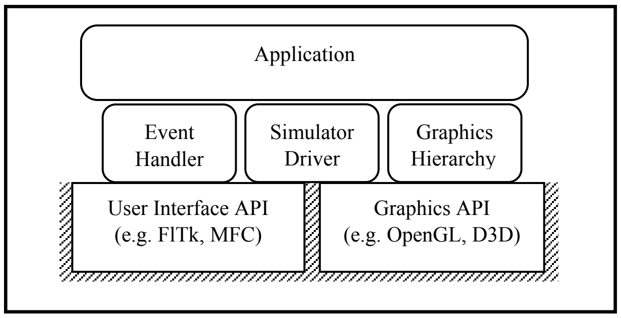
Application
Event Handler
Simulator Driver
Graphics Hierarchy
User Interface API (e.g. FlTk, MFC)
Graphics API (e.g. OpenGL, D3D)

Figure 1: Components in Interactive Applications

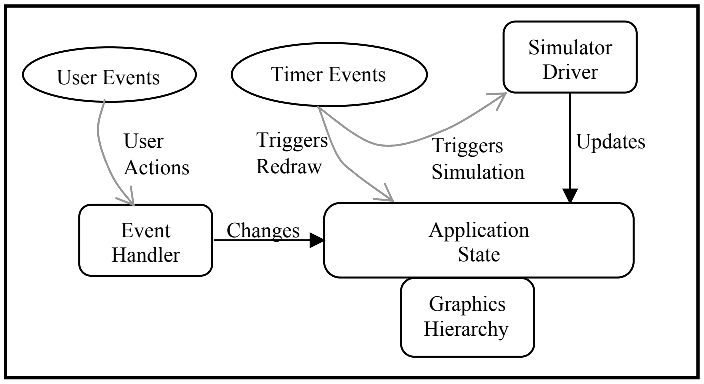

Figure 2: Application Architecture

3.1. The Target Application System

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 [40], or MFC [41]) and Graphics APIs (e.g. OpenGL, or MS Direct3D). 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.

```
class TgraphicsObject {
    protected:
        CompoundObjectAttribute(XformInfo)              // Object transformation support
        BoolObjectAttribute(HiLight)                    // Object highlight on/off flag
        BoolObjectAttribute(Visibility)                 // Object visible on/off flag

    // Following methods define the behavior of a TgraphicsObject. Subclasses are responsible for
    // overriding the selected methods to define/modify their specific behaviors.
        virtual void InitializeGraphicsObject();        // initializes the object (XformInfo, HiLight, Visibility)
        virtual void SetupTransform(TtransformStack *); // sets up transformation for drawing
        virtual bool ShouldDrawObject();                // should this object be drawn (e.g. is it visible?)

    // Following are behaviors defined (but not implemented) for TgraphicsObject.
    // Sub-classes must implement these methods.
        virtual void SetObjectAttributes() = 0;         // set the object's attributes
        virtual void UnsetObjectAttributes() = 0;       // restore  the global state
        virtual void DrawObjectGeom() = 0;              // draw the actual geometry
        virtual void EchoHighLight() = 0;               // show highlighted for this object

    public:
        void DrawObject(TtransformStack *stack) {
            if (ShouldDrawObject() ) {
                // save current top of transformation stack ...
                stack->Push();
                    SetupTransform(stack);
                    stack->LoadTopToDevice();
                    SetObjectAttributes();
                    if (fHiLighted) EchoHighLight();
                    DrawObjectGeom();
                    UnsetObjectAttributes();
                // Now restore the transformation stack
                stack->Pop();
                    stack->LoadTopToDevice();
        } }
```

Figure 3: The TgraphicsObject Base Class

3.2. Event and Simulator Driven Programming Model

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 [42]. 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 involved, and the programming assignments are typically interesting interfaces that students can play with.

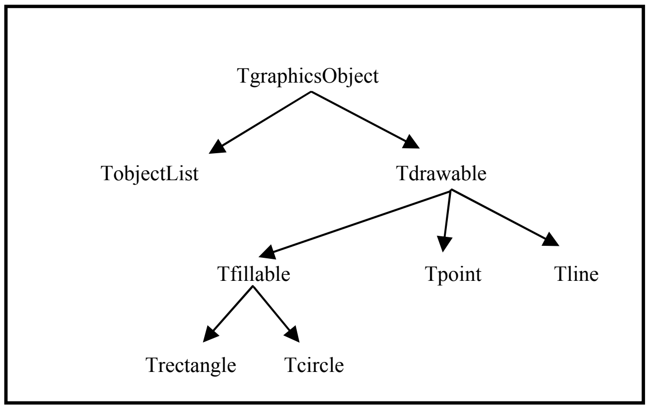

Figure 4: The Graphics Hierarchy

The programming assignment for this topic is 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.

3.3. 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. Figure 3 shows the pseudo code of an abstract base class for objects that know how to draw themselves. Notice that the TgraphicsObject::DrawObject() function is defined based on mostly pure virtual functions. This solution design approach where implementation of functionality is based on defined yet unimplemented behaviors is an important element in working with large scale software systems. Figure 4 shows an example of how a graphics hierarchy can be defined based on the TgraphicsObject class.

The programming assignment for this topic would be developing software that demonstrates how to interact with the behaviors of abstract graphics objects (e.g. TgraphicsObject) without knowledge of what actual primitives are being processed. For example, design a primitive drawing program based on the behaviors of TgraphicsObject, 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 TgraphicsObject behaviors.

3.4. 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 4, 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.

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

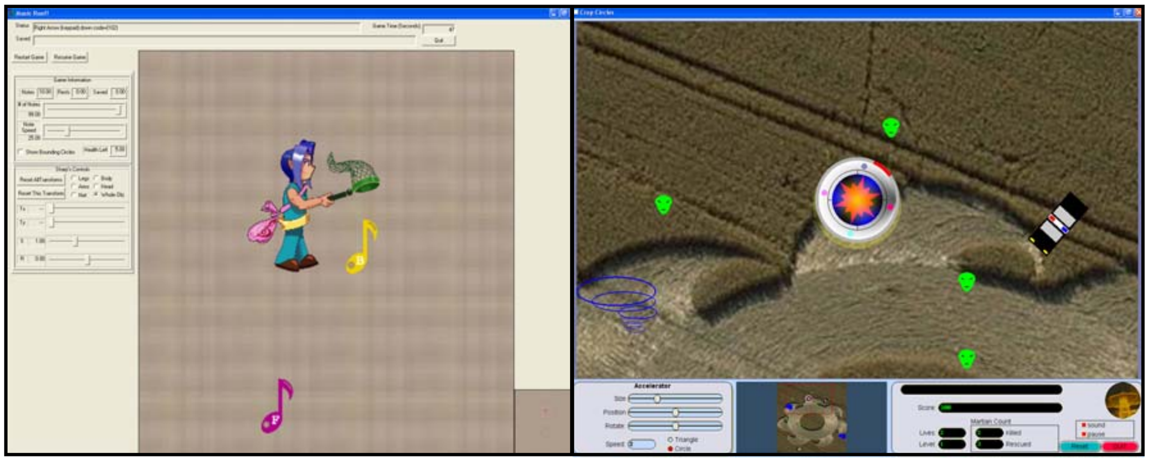
Figure 5: Example Final Projects

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.

3.5. Discussion

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. The goal here is to introduce students to these ideas that will support their future self studying. 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.

Top-down approach focuses on studying the modules of existing systems. The objectives are not only for students to understand the design and implementation of the modules, but at a deeper level, for students to be able to utilize the ideas behind these modules in constructing new systems based on general technical specifications. For this reason, a significant size final project is an important tool for students to practice what they have learned, and for evaluating students' learning outcome. 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* [43], 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. Figure 5 shows examples of student final projects from 2002 and 2001. Similar technical difficulties can be observed, where in both applications there are two views into the world with *hero* objects that can be extensively transformed. Under the user control these hero objects would interact with the surrounding objects based on collision detection. One interesting fact is that the system on the left is implemented based on MFC and Direct3D, while the system on the right is in FlTk and OpenGL. 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 currently no CG text that uses the described top-down approach. As a result, the course has been based on a few reference texts 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 a single text.

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 the Fall 2002 CSS450 offering, students were 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 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.

5 Conclusion

The traditional approaches to introducing a new engineering discipline begin by providing an overview of the discipline. These approaches then move on to study the details of the foundational building blocks in the discipline. This is a bottom-up solution for gaining knowledge in a new discipline. This is a solution designed for traditional university students. These students are usually in their late teens or early twenties. They typically have been full time students all their life; do not have real world working experiences; and lack the maturity in solution formulation for real-world problems. The bottom-up approach taps into students' strength/familiarity in basic mathematics skills and presents to students examples of solution formulations.

Top-down approaches introduce the knowledge based on analyzing one holistic perspective of the discipline. These approaches identify specific systems and study the requirements for understanding and implementing such systems. The advantage of top-down approach is that the holistic-system perspective supports the learning of practical knowledge. The potential problem is that the holistic-system perspective typically only presents *a specific perspective* of the discipline. There is always the danger that students may learn a restricted perception from top-down approach (e.g. interactive graphics application or batch image generation system). For more mature students with rich life experience, and/or students with more pragmatic need (e.g. non-major students who need to apply the knowledge), carefully designed top-down approach can be both effective and efficient.

This paper presents a top-down approach to teach introductory CG course. This approach focuses on topics required to implement a ''*popular 2D interactive graphics application*''. The foundation of the top-down analysis is based on popular

APIs. The balance between skill set training and fundamental concepts are achieved by performing the analysis based on more than one API from different vendors; and by designing programming assignments that practice the basic concepts. From the small number of students taken this class over the pass three years, it appears this approach is successful in arousing students' interests and preparing students for a development career in the CG field.

Finally, it is important to point out that students who have done well in top-down introductory CG courses may have trouble providing detailed answers to some ‘‘*standard textbook questions*’’. For example, how to fill the interior of a triangle given that we know how to turn on pixels? Students would fair much better at higher level type questions like: how to implement grouping operations, or data structure requirements in supporting the drawing operations in systems like *powerpoint*.

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 discussions with Frank Cioch. It is great to have a colleague who disagrees and yet is willing to listen. The final projects in Figure 5, *Music Run* is by Aida Sakkal; and *Crop Circles* is by Steve Baer.

REFERENCES

[1] Edward Yourdon, and Larry Constantine, ‘‘*Structured Design: Fundamentals of a Discipline of Computer Program and System Design*,’’ Yourdon Press, Englewood Cliffs, N.J., 1979.

[2] Judith Brown, Robert Burton, Steve Cunningham, and Mark Ohlson., ‘‘*Varieties of Computer Graphics Courses in Computer Science*,’’ 19th ACM SIGCSE Technical Symposium on Computer Science Education, 1988, SIGCSE Bulletin, Vol. 20, No. 1, PP 313.

[3] Scott Grissom, Jack Bresenham, Bill Kubitz, and Scott Owen, ‘‘*Approaches to Teaching Computer Graphics*,’’ 26th ACM SIGCSE Technical Symposium on Computer Science Education, March 1995, PP. 382-383.

[4] M. M. Larrondo-Petrie, J. Bresenham, C. Laxer, J. Lansdown, G. S. Owen, ‘‘*Approached to Teaching Introductory Comtpuer Graphics*,’’ Proceedings of the ACM SIGGRAPH 94, pp. 479-480, 1994.

[5] Lew Hitchner, Steve Cummingham, Scott Grissom, and Rosalee Wolfe, ‘‘*Compter Graphics: The Introductory Grows Up*,’’ 30th ACM SIGCSE Technical Symposium on Computer Science Education, March 1999, PP. 341-342.

[6] Donald Hearn and Pauline Baker, ‘‘*Computer Graphics – C Version*,’’, second edition, Prentice Hall, 1997.

[7] F. S. Hills, Jr, ‘‘*Computer Graphics using OpenGL*,’’ second edition, Prentice Hall, 2001.

[8] James Foley, Andries, Van dam, Steven Feiner, John Hughes, and Richard Phillips, ‘‘*Introduction to Computer Graphics*,’’ Addison Wesley, 1994.

[9] Peter Shirley, ‘‘*Fundamental of Computer Graphics,*’’ A. K. Peters, 2002.

[10] Mason Woo, Jackie Neider, and Tom Davis, ‘‘*Programming Guide, 2nd Edition: The Official Guide to Learning OpenGL*,’’ Version 1.1, Addison Wesley.

[11] Home Page, Microsoft DirectX, http://msdn.microsoft.com/directx/, 2003.

[12] Barbara Boucher Owens, Gene Bailey, Ted Mims, Shell Heller, and Laurie White, ‘‘*The Non-Traditional Student in Computing: Characteristics, Needs and Experiences*,’’ 27th ACM SIGCSE Technical Symposium on Computer Science Education, 1996, PP. 372-373.

[13] Maya Unlimited, Alias|Wavefront, Toronto, Canada, http://www.aliaswavefront.com, 2002.

[14] Home Page, University of Washington, Bothell, http://bothell.washington.edu, 2002.

[15] ‘‘*Serving Adult Learners in Higher Education: Principles of Effectiveness,*’’ Executive Summary, Chicago: Council for Adult and Experiential Learning, 2000 (pdf document available at http://www.cael.org).

[16] Tonkin, H. ‘‘*Continuing education's time of promise*,’’ Continuing Higher Education Review, No. 62, PP. 40-56, 1998.

[17] Home Page, Computing and Software Systems Program, University of Washington, Bothell, http://bothell.washington.edu/CSS, 2002.

[18] ‘‘*Adult Degree Programs: Quality Issues, Problem Areas, and Action Stpes,*’’ Council for Adult and Experiential Learning and the American Council on Education, March 1993 (pdf document available at http://www.cael.org).

[19] Michael Stiber, ‘‘*Multimedia and Signal Computing Course Redesign*,’’ Worthington Technology Award, University of Washington, Bothell, 2003.

[20] Arnold Berger, ‘‘*Experience and results of teaching CSS 422: in-person and in-DL*’’, Presentation of pilot distance learning experiment to the CSS faculty, Spring 2003.

[21] Kelvin Sung, Peter Shirley, ‘‘*A Top-Down Approach to Teaching Introductory Computer Graphics*,’’ SIGGRAPH 2003 Educator's Program, Conference CD/DVD, July 2003.

[22] Kelvin Sung, Peter Shirley, ‘‘*Teaching Computer Graphics Programming To Non-Traditional Returning Adult Students*,’’ Extended Abstract, to appear in Eurographics/ACM SIGGRAPH Workshop on Computer Graphics Education 2004, June 2004.

[23] Kelvin Sung, Peter Shirley, ‘‘*Algorithm Analysis for Returning Adult Students*,’’ Submitted for publication, November 2003.

[24] Scott Owen, ‘‘*Experience in Teaching an Advanced Computer Graphics Course*,’’ SIGCSE Bulletin, vol. 22, no. 1, 1990, pp. 162-166.

[25] Dino Schweitzer, and Tom Appoloni, ‘‘*Integrating Introductory Courses in Computer Graphics And Anomation*,’’ 26th ACM SIGCSE Technical Symposium on Computer Science Education, March 1995, PP. 186-190.

[26] Steven Cunningham, ‘‘*Powers of 10: The Case for Changing the First Course in Computer Graphics*’’, Proceedings of the SIGCSE 2000 conference, Austin, TX, March 2000, pp. 293-296.

[27] Steve Cunningham, ‘‘*Computer Graphics: Programming, Problem Solving, and Visual Communication*,’’ Available for download from http://www.cs.csustan.edu/~rsc/NSF/, 2003.

[28] Steve Cunningham, ‘‘*Re-Inventing the Introductory Computer Graphics Course: Providing Tools for a Wider Audience*,’’ Computer & Graphics, Vol. 24, No. 2, April 2000, pp 293-296.

[29] John Lowther, and Ching-Kuang Shene, ‘‘*Rendering + Modeling + Animation + Postprocessing = Computer Graphics*,’’ The Jornal of Computing in Small Colleges, Vol. 16, No. 1, November 2000, PP. 20-28.

[30] Rosalee Wolfe, ‘‘*A Syllabus Survey: Examing the State of Current Practice in Introductory Computer Science Courses*,’’ Computer Graphics, Vol. 33, No. 1, February 1999, pp. 32-33.

[31] Rosalee Wolfe, ‘‘*New Possibilities in the Introductory Graphics Course for Computer Science Majors*,’’. Computer Graphics, Vol. 33, No. 2, May 1999, pp. 35-49.

[32] Rosalee Wolfe, ‘‘*Bringing the Introductory Computer Graphics Course into the 21st Century*,’’ Proceedings of the Graphics and Visualization Education (GVE’99) Workshop. pp. 3-6.

[33] Steve Cunningham, Sylvia Clark Pulliam, Charles D. Swanson, and Peter R Turner, ‘‘*Computational Science and Engineering – Tools and Techniques for Teaching*,’’ ACM SIGCSE Technical Symposium on Computer Science Education, February 2002, pp. 135-136.

[34] Peter R Turner, Angela Shiflet, Steve Cunningham, Kris Stewart, Andrew Phillips, and Ignatios Vakalis, ‘‘*Undergraduate Computational Science and Engineering Programs and Courses*,’’ ACM SIGCSE Technical Symposium on Computer Science Education, February 2002, pp. 96-97.

[35] ‘‘*Learning Maya 3*,’’ Alias|Wavefront Education, Alias|Wavefront, Toronto, Canada, 2000.

[36] Mason McCuskey, ‘‘*Special Effects GAME programming with DIRECT X*,’’ Premier Press, 2002.

[37] Peter Walsh, ‘‘*Advanced 3-D GAME Programming using DirectX 8.0*,’’ Wordware Publishing, 2002.

[38] Ian Parberry, ‘‘*Introduction to Computer Game Programming with DirectX 8.0*,’’ Wordware Publishing, 2001.

[39] Home Page, CSS450: Introduction to Computer Graphics, http://courses.washington.edu/css450, 2003.

[40] Home Page, FlTk, http://www.fltk.org, 2003.

[41] Jeff Prosise, ‘‘*Programming Windows with MFC*,’’ Second Edition, Microsoft Press, 1999.

[42] Brad Myer, ‘‘*Separating Application Code From Toolkits: Eliminating the Spaghetti of call-backs,*’’ Proceedings of the UIST’91, pp. 211-220, November 1991.

[43] Edward Angel, ‘‘*Interactive Computer Graphics – a Top-Down Approach with OpenGL*,’’ Second Edition, Addison Wesley, 2000.