Teaching Computer Graphics Based on a Commercial Product

Gregory Smith and Kelvin Sung

Computing & Software Systems
University of Washington Bothell

Abstract

The challenges in designing an introductory Computer Graphics (CG) course include selecting an appropriate and coherent set of topics, keeping up-to-date with the rapidly evolving industry, and aligning with the many students' fascinations that tend to stem from flashy popular media. This paper analyzes and classifies existing introductory CG classes according to their approaches in trading-off between covering foundation algorithms and focusing on application-level knowledge. The paper then observes that many application-level courses challenge students in learning and applying relevant CG concepts by building familiar graphical applications. Within this context, the paper points out that many modern commercial graphical applications, including popular game engines and 3D modeling systems, support well-defined and robust run-time scripting interfaces that allow modification and/or replacement of default system functional modules. These observations suggest the potentials of delivering an introductory CG class based on one of these commercial graphical systems. This paper proposes a set of guidelines to ensure such a class will educate CG practitioners rather than commercial product users. Based on these guidelines and an existing application-based introductory CG course, a new set of learning outcomes is derived which is independent of any specific commercial product. The paper continues to describe the implementation of a new course using the Unity3D game engine as the delivery vehicle. This paper then describes the associated teaching materials, details the hands-on programming assignments, and discusses student learning from the Unity3D-based introductory CG class. The results from two consecutive batches of students demonstrated that a commercial graphical product-based approach to teaching an introductory CG class could be effective, welcomed by students, and supply students the concepts to build practical graphical applications after the class.

CCS Concepts

• ***Social and professional topics*** → ***Computer science education;*** • ***Computing methodologies*** → ***Computer graphics;***

1. Introduction

Designing an introductory Computer Graphics (CG) class can be challenging: CG is a vast field [BWF17] that is driven by a highly competitive and rapidly changing industry, and CG is appealing to many young students because of their experience with popular software, e.g., Maya [May19] or Unity3D [Uni19], or media, e.g., animated movies or video games [CHV*02]. Since undergraduates in a typical Computer Science (CS) degree program [ACM19] can only take one elective class in CG, this class should cover essential concepts to facilitate continuous student self-learning, and, at the same time, attempt to align with students' fascination with CG applications.

Educators continuously identify and evolve the set of essential concepts in CG [BWF17], and selectively cover these topics with varying degrees of depth based on the philosophy of their classes. Some CG classes attract and engage students based on exotic devices [Wer12] or by associating relevant concepts and algorithms with exciting modern application areas, e.g., video games [SSRR07], or visualization [Cun07].

Many of the popular commercial CG software systems in these areas have been available for more than a decade (e.g., Maya was first released in 1998, and Unity3D in 2005); and have been and continue to be under constant and continuous refinement. These systems are considered mature because they are industry and customer-tested over a long period; and they allow dynamic behavior modifications and customizations via runtime scripting support. Due to these qualities, these sophisticated commercial CG software systems present an exciting potential for teaching CG concepts/algorithms because they allow students to learn by building and replacing default functional modules.

At the University of Washington Bothell, we have redesigned our introductory CG class based on learning and practicing concepts by implementing and replacing some of the functional modules of a commercial product. This paper details that design and implementation, presents example teaching materials, and discusses learning outcomes by examining sample student projects. Surveys of students from two consecutive offerings indicated that they have enjoyed learning the many concepts related to and based on the commercial product, appreciated the modern, sophisticated devel-

DOI: 10.2312/eged.20191031

opment settings, and welcomed the opportunity of hands-on experience in modifying and replacing functional modules of a familiar system.

2. Background and Motivation

The classical approach to introductory CG typically iterates through each of the topics in relative isolation, e.g., [CBBO88, HCGW99]. More recent approaches recognize students' fascination with CG applications [AB15] and engage them by organizing and relating the topics to actual familiar applications, e.g., [SSRR07, Hu10]. Based on the general philosophical approaches to covering CG concepts, most of the introductory CG classes can be classified broadly into three categories.

Concept-Based. Focusing on the fundamentals, these approaches typically examine concepts and challenge students to implement without any specific API support, e.g., [Hu10, CXR18]. In this way, students gain hands-on experience in developing low-level graphics algorithms. However, due to time constraints, these classes typically trade off application-level knowledge, such as color models, transformations, or event-driven interactions, with supplementary readings [Hu10]. Additionally, students from these classes may lack an overall system-level understanding of CG concepts. For example, a student may know how to trace a ray to compute visibility and yet may not appreciate how grouping behaviors are based on the transformations they learned [ACSS06].

Graphics API- Based. These classes focus on covering the concepts based on tools adopted by the industry—OpenGL [BWF17], Vulkan [Vol19],or GLSL for shading computations [Cli15]. These approaches have demonstrated effective learning outcomes with many suitable popular textbooks, e.g., [HB03, AS12]. However, a significant challenge with these types of courses is that, with the competitive and fast-moving CG industry, the associated tools are continuously and rapidly evolving where educators must constantly play catch-up—modifying and updating their teaching materials or risk being left behind with obsolete tools [Wol12]. Compounding this problem is the fact that the APIs are designed and evolving based on specific industrial needs, often driven by stringent performance requirements, which may not align with the educational goals of introductory classes. For example, the deprecation of fixed function pipeline from OpenGL was a substantial challenge for many educators [Wol12]. More significantly, the recent evolution to Vulkan-API, besides the impact on the many highly effective and popular OpenGL-based textbooks, reflects the motivation for supporting hardware performance requirements and does not attempt to reflect the fundamental concepts in CG. While these can serve as excellent tools for advanced studies, they present significant challenges for learning the basic concepts in an introductory course.

Application-Based. These classes cover and focus on how the essential concepts support and implement the end-user functionality of CG applications [ACSS06, AP10]. One particular approach is to examine CG concepts in the context of and proceed to base the entire class on the building of a "moderately complex" interactive graphical application. Such a class analyzes large scale interactive graphical applications, identifies requirements, derives the composition modules, relates the modules to CG concepts/algorithms, and leads students to build the corresponding modules towards the CG application [SS04, Lew12]. These classes trade-off algorithmic- for application-level CG knowledge. For example, students from these classes will be able to design and implement a mesh manipulation system but may be challenged when attempting to describe how triangles are scan converted in the GPU.

2.1. Summary and Opportunity

While designed to teach a similar set of concepts, the outcomes from these classes are significantly different. The concept-based methodology teaches under the API-hood, the API-based classes examine the concepts using CG APIs, while the application-based technique examines and builds end-user functionality based on the concepts. Educators design and customize among these approaches to align students' needs with their own philosophies in learning.

At the University of Washington Bothell, more than 40% of incoming first-year students are first-generation college students. Many of these students pursue their degrees with the primary purpose of securing quality employment with the vibrant regional high-tech industry to improve their lives. Balancing the students' near-term employability needs is the faculty's fundamental belief in educating practitioners rather than users of CG.

Our previous application-based approach to introductory CG class [SS04, SSRR07] aligns well with both students' and faculty's needs by challenging students to work with and develop a moderately complex interactive application from scratch. While successful, the approach involved substantial investments in building the necessary software infrastructure and required students to quickly comprehend and work with the large and complex source code base. As a direct result in compromising the investment requirement and managing the complexity of the system, this class was restricted to 2D.

With many modern matured graphical applications supporting straightforward extension definition and/or run-time behavior modification via scripting, a potentially useful alternative would be to replace the custom toy framework with a real commercial product. With such an approach, instead of being distracted by a large custom source code base, students can work with well-defined and well-documented interfaces. In such a class, more time can be dedicated to focus on CG-related concepts and the new concepts learned can be practiced and implemented in the context of much more sophisticated and complete real-life commercial system.

3. Course Design Guidelines

Our goal is an effective application-based introductory CG class based on an existing commercial software application. Such a class can easily degenerate into one that educates users for the commercial product rather than general practitioners of the CG field. The following guidelines are derived to ensure proper and desirable learning outcomes can be achieved.

- Choice of concept coverage: the topic coverage must apply to any introductory CG classes and must be agnostic to any specific commercial product(s).

- Implementation of concepts: students must be able to examine and learn CG concepts and then implement and replace the corresponding functional module in the product.
- Programming language and environment: the chosen product must support proper object orientated abstraction with a modern programming language in a typical IDE with general debugging support ensuring students are exposed to contemporary and relevant hands-on development experience.
- Ease of working with the system: the chosen product must allow students to learn and work with within days with readily accessible functionality that features relevant CG concepts.

Following this guideline, a course would cover common introductory CG topics, where students will learn and practice implementing the concepts in a typical software development environment, and a minimum or negligible time will be spent on learning the commercial software system itself. The ultimate goal is for the outcome of such a class to be similar to any application-based introductory CG-class.

4. Course Design

Our course is an undergraduate upper-division or graduate introductory-level CG class with a maximum enrollment of 45 students. Our academic terms are 10 weeks with twice-weekly 2-hour lectures for a total of 40 contact hours.

Concerned with and to avoid training users of a product, a conscious decision was made early on that the class must be designed entirely independent from and before any decisions on the final chosen commercial product.

We approached this task by modifying the student learning outcomes of our previous 2D version in the context of supporting 3D interactive graphical applications. The design guideline is followed to firmly define the topic coverage and technical requirements of programming assignments. These results are then used to derive the requirements of the commercial platform for instruction. The teaching materials and assignment implementations proceeded only after the above steps are completed.

4.1. Student Learning Outcomes

Rather encouragingly, during the redesigning process, we did not find the need to modify our original course objectives. Instead, we were able to replace software development related goals with 3D and GPU-related topics in a straightforward manner. Updated student learning outcomes are listed below.

- Learning Objective 1 (LO1): Describe popular interactive graphical software systems in the context of Model-View-Controller architecture.
- Learning Objective 2 (LO2): Design and implement 3D interactive graphical applications that support: real-time user manipulation of graphical scenes; multiple camera views; scene graphs with multiple animate-able components; and custom vertex and pixel shaders with basic effects including textures and simple illumination.
- Learning Objective 3 (LO3): Discuss the programming model of contemporary graphics API.

4.2. Course Topic Coverage

In following the *Choice of concept coverage* design guideline (as specified in Section 3), the only topics removed from the original class were those related to the software library infrastructure development and evolution [SSRR07]. Those topics were replaced with polygonal modeling and GPU vertex/fragment programming. Other 2D topics were generalized into the corresponding 3D versions, e.g., windows and viewport were replaced by 3D cameras and the view transformation. The following is the approximated schedule:

Week 1: Learning Tools and Event Driven Programming
Week 2: MVC Architecture and GUI Programming
Week 3: Vectors, Matrices, and Transforms
Week 4: Coordinate Spaces and Transformations
Week 5: Scene Node, Hierarchy, and Simple Animation
Week 6: 3D Viewing and Interactive Camera Manipulation
Week 7: Mesh and Polygonal Modeling
Week 8: Texture Mapping and Shading
Week 9: Illumination Model and Final Project
Week 10: Shaders and Final Project

The first two weeks are dedicated to the fundamentals of interactive systems: the understanding of event-driven programming and explaining interactive systems based on the MVC architecture (LO1). Weeks 3 to 5 are the essential mathematics and data structures, while week 6 covers 3D viewing and camera manipulations (LO2). The second half of the academic term examines modeling, rendering, and the corresponding GPU support in these areas (LO3).

4.3. Programming Assignments

The learning objectives for the course, as expected, are centered on students' ability in designing and building interactive graphic applications. The assessment of these abilities is crucial. Inheriting the results from and mirroring the topic changes from our previous class offering, and following the *Implementation of concepts* and the *Programming language and environment* guidelines, the hands-on programming assignments were specified and used as requirements for identifying the instructional commercial platform. The following are the high-level assignment specifications.

- Warm-up: system and environment familiarization.
- GUI programming: re-usable components in MVC-framework.
- Mathematics fundamentals: implementation and interaction.
- 3D application I: camera and scene graph manipulation.
- 3D application II: 3D modeling editor.
- Final 2 or 3 person-group project: propose/design/build a "useful" interactive 3D graphical application.

When compared with those from the 2D version of the course the only substantive changes are in the two 3D application assignments: working with 3D cameras instead of 2D windows and viewports, and, 3D polygonal modeling instead of 2D interactions of multiple scene hierarchies.

4.4. Choice of Commercial Platform

The assignment specifications, together with the topics listed in Section 4.2, define the requirements for the instructional commercial system. The *Implementation of concepts* design guideline dictates that the commercial system must support interactive examination of the functionality listed in Section 4.2, and, at the same time, allow the implementation of the assignment specifications to replace the default system functional modules. Very importantly, with less than a week dedicated to the learning and familiarization, the commercial system must be easy to learn and straightforward to use.

Many of the modern 3D graphical applications [Wik19], including Unity3D, satisfy the above requirements. Our choice of Unity3D as the instructional commercial platform is based on the following additional reasons. First, being a game engine, the application domain aligns with many of our students' interests. Second, this system is designed for hobbyist and designers who are often not software developers and thus is relatively straightforward to learn, satisfying the *Ease of working* design guideline. Finally, and very importantly, the system is free for educators and students.

4.5. Discussion

The details of our course design as presented, all except Section 4.4, are rather unremarkable and is similar to any application-based introductory CG class. That this is so should not be surprising. We followed our design guidelines and the explicit goal of delivering a typical introductory CG class. The specific commercial product for content delivery is but an example of implementation.

5. Course Implementation

Implementing a course design requires an appropriate textbook, instructional materials for students to interact with and explore the topics, and assessment instruments to verify the learning outcomes.

5.1. Reading Materials

A relevant textbook is convenient for students to prepare for class, review for topics, and refer to when working on assignments. In general, it can be challenging to find appropriate textbooks for the application-based approach. An effective textbook must clearly separate the coverage of concepts into sections that are independent of those that apply specifically to the target application. The effectiveness of this separation governs the outcome of educating designers and developers rather than users of the application. To the best of our knowledge, such a book does not exist for teaching introductory CG concepts based on Unity3D.

In order to accomplish the effective separation of concepts from Unity3D, we have purposely avoided tutorial-like trade-books. Instead, we take the approach of mapping topic coverage to relevant chapters of existing concept-based textbooks: interactive systems [SSB08], vectors and matrices [DP11], general 3D topics [MS16, AS12], and GPU and shaders [BC11].

A potential pitfall of disjoint book chapter references is the possibility of inconsistent topic presentations resulting in students unable to relate to concepts holistically. Fortunately, from our experience [SS04], this problem can be avoided with a set of coherent and interactive content delivery materials, as detailed in the following.

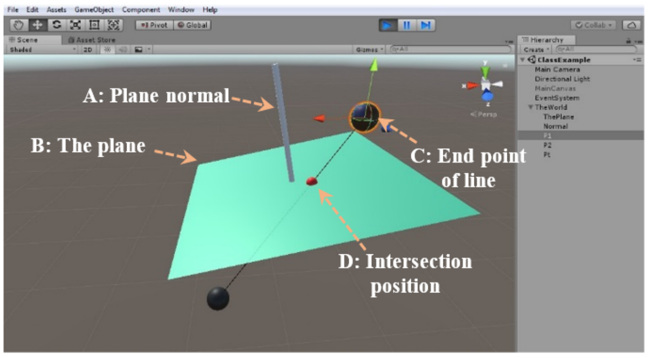

Figure 1: *Example of concept demonstration program (CDP)*

5.2. Course Content Delivery

Complementing the abstract explanations found in the different textbook chapters is a coherent and elaborate set of concept demonstration programs (CDPs) and accompanied lecture notes. The lecture notes describe how the CDPs relate to and demonstrate the abstract concepts while explaining the implementation source code. In this way, students can examine the implementation source code while interactively exploring the CDPs.

More than 55 CDPs are developed. These CDPs, as well as the accompanying lecture notes, are organized according to the corresponding topics. The following illustrates these materials by detailing an example from the coverage of vector math topics.

Essential Vector Math. As discussed in Section 4.2, this topic is covered around the third week of class. By this point, students are comfortable working with and developing in the environment. Additionally, they have hands-on experience designing and implementing MVC-based interactive applications. The goals from covering this topic are for students to understand and be able to implement functionality using vectors in interactive graphics applications. Essential concepts in vectors are organized into the following examples: speed and parametric lines, point to line distance, rotation with quaternions, plane equation and normal, point to plane distance, line to plane intersection, and reflecting a vector.

Figure 1 shows the CDP that accompanies the line to plane intersection lecture notes. The grey cylinder (A) through the green plane (B) represents the plane normal; the two black spheres are the endpoints (C) of the line segment to be intersected, and the red sphere (D) is the intersection position. With this CDP, students can interactively select and manipulate the position, orientation, and size on the green plane and the two black spheres and then observe the computation results in the grey cylinder and red sphere. The accompanying lecture notes focus on explaining the 20+ lines of implementation, relate relevant source code to vector concepts covered, and encourages students to modify the implementation to observe the corresponding effects.

Discussions. Our course is primarily an iteration over the CDPs and the accompanying lecture notes. Each CDP focuses on demonstrating one concept. This simplicity ensures that students can confidently understand the entire implementation as well as examine

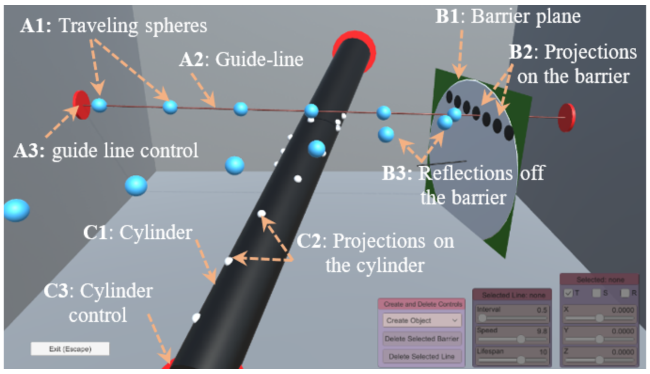

Figure 2: *Assignment 3. Credit: Bryan Veneruso.*

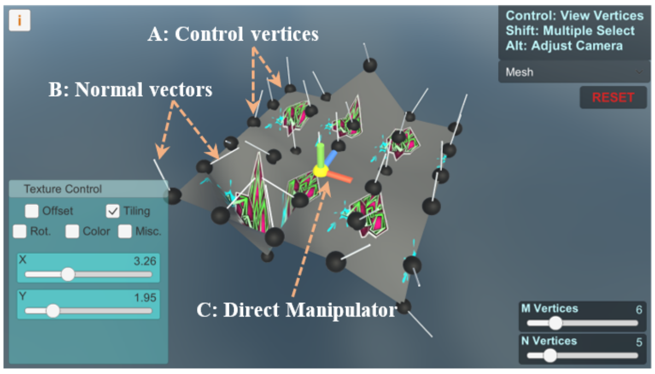

Figure 3: *Assignment 5 (Part 1). Credit: Aaron Holloway, Nicholas Lewis, and Kyla NeSmith.*

and extensively interact with all aspects of that implementation. In this way, although the reading materials are from different references, students can experiment with concepts described in the same familiar environment. The overall system-level contextual understanding and the synthesis of the concepts are delivered through the programming assignments. All CDPs and the accompanying lecture notes are available to the general public [Sun19].

5.3. Assignment Implementation

For all the assignments and the final project, students are only allowed to work with the simple matrix methods such as translate, rotate, scale, concatenate, and transform points; and the underlying vector library functions like add, dot, cross, and normalize. The more advanced composited functions that are Unity3D specific are off-limits. The following illustrates the types of work involved by detailing two of the five assignments. All screenshots are based on student implementations.

Assignment 3: Fun with Vectors. This two-week assignment is due after the coverage of essential vector math. As illustrated in Figure 2, students must build an interactive application that supports users controlling the generation interval and speed of blue traveling spheres (label-A1 in the top-left corner), these spheres travel along the direction defined by a guide-line (A2) that is under the user's control via the two red buttons that are constrained to the side walls (A3).

In the top-right, the barrier plane (B1) displays the projection of the blue traveling balls as black shadows (B2) on the plane and reflects the balls upon contact (B3). The C1-label in the bottom-left points to the brown cylinder that displays the projections of the traveling balls as small white spheres (C2). Similar to the guide-line, the endpoints of the cylinder can be manipulated by the user via the two red buttons that are constrained to the back wall and the front floor (C3). The GUI gadgets in the lower-right are for controlling the traveling balls and manipulating the barrier plane. The non-trivial user interactions are designed to encourage students to follow the MVC architecture in their implementation. All mathematics requirements and implementations are covered in CDPs.

Assignment 5: Polygonal Modeling. This is a group (two- or three-person) two-week assignment challenging students with implementing direct object manipulation (instead of via GUI widgets) and simple polygonal modeling. Students must implement a simple polygonal mesh modeler that supports texture placements, direct per-vertex user manipulation, and algorithmic modeling such as rotational sweep.

Figure 3 shows the first part of the assignment. In the middle is a 6x5 polygonal mesh with the black spheres (A) indicating the control vertex positions and narrow white cylinders (B) showing the directions of the vertex normal vectors. Label-C shows that when the user clicks on one of the control vertices, a Direct Manipulator object will appear giving the user the option to drag on one of the colored cylinders to move the selected control vertex in the corresponding direction and thus manipulating the polygonal mesh object.

On the lower-left corner of Figure 3 is a GUI widget that controls the texture placement on the polygonal mesh. In this case, the simple pattern is tiled 3.26x1.95 times in the u and v directions. Students must process the GUI values, construct a texture placement matrix, and forward the matrix to their own vertex shader to transform the texture coordinate defined on each vertex to accomplish the placement functionality.

5.4. Discussion

Through the CDPs and the programming assignments, students implement functional modules that could potentially replace those in the Unity3D system. For example, in Assignment 5 the texture transform module resembles the default texture placement functionality on Unity3D's shaders, and the Direct Manipulator object provides valuable insights into the default object manipulator in the editor. Although none of these implementations are as complete or user-friendly as the default ones, the experience allows students to begin to understand how they too can implement similar modules if given the time.

The programming assignments are rather extensive and challenging. However, it is also true that much of the implementation source code is provided in relevant CDPs. The assignments serve the important purpose of assessing student understanding and their

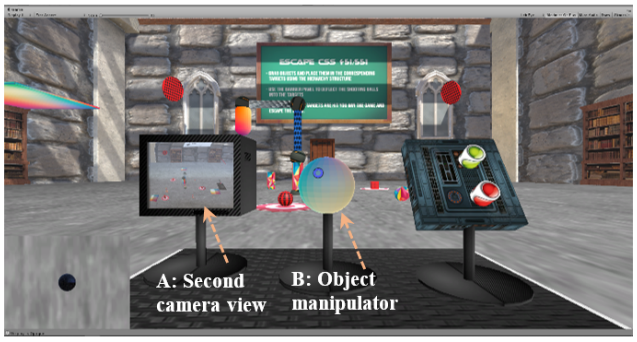

Figure 4: *Escape the Room game. Credit: Shaikh Amaan, Matt Johnson and Henry Nguyen*

ability to synthesize coherent solutions based on individual concepts and implementations.

6. Results

The following presents the results of this class from two perspectives: students' capabilities through final work and their anonymous feedback at the end of the class.

6.1. Students' Capabilities

The class final project requires students to reflect upon, propose, and build a meaningful application based on all topics learned. Through this process, students must understand the individual concepts learned, synthesize a solution, and apply their knowledge by building a usable system. In this way, this final application represents the culmination of students' understanding and can be analyzed to verify the learning outcomes.

Video game Final Projects. With the game engine delivery vehicle, it is not surprising that student final projects can be broadly categorized into games and non-games. Figure 4 shows a typical video game final project. In this game, the player can select/manipulate objects to solve puzzles to escape the room. The technical details of this type of project include multiple objects based on custom scene node hierarchies, object interaction based on vector math, multiple camera views under both user and scene node transform controls, and simple shading and textures.

In this case, since the students are not allowed to use any of the Unity3D specific functions, this type of project demonstrates that students are on their way to developing video games independent from many of the relevant Unity3D functionality. Ironically, the simple video game projects are often constrained by the technical requirements resulting in awkward gameplay. For example, in the middle of Figure 4 is a second camera view (A), and an object manipulator (B). These items are present to satisfy the technical requirements and do not facilitate the Escape the Room gameplay.

Figure 5 shows that some of the video game attempts are exploratory projects and can be exciting and fascinating. This is a two-person Augmented Reality (AR) game developed based on the

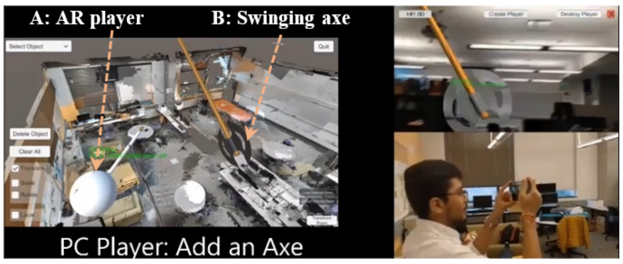

Figure 5: *AR Obstacle Course game. Credit: Anjal Doshi and Nikhil Grandhi*

Augmented Space Library [TYC*17] from the Cross Reality Collaboration Sandbox Research group [CRC19]. The first player is on the PC in a scanned environment adding obstacles for the second AR player in the actual environment to navigate. Figure 5 is composited based on three separate camera views: on the left is the PC player's view, notice that the white sphere (A) is the current position of the AR player. The top-right is the AR player's view, and bottom right is a photograph of the AR player playing the game. In this case, the PC player has added a swinging axe (B) into the room and the AR player, through the AR device (a mobile phone), is observing and must navigate the room while avoiding the axe.

As illustrated by these two projects, the actual technical specifications of final projects can vary. Straightforward video games must demonstrate a particular set of functionality, while more exploratory projects have less prescribed requirements. In the case of the AR game, due to the many unknown issues that must be resolved in the three weeks' period, the project has no defined technical requirements. The AR project represents students applying CG-related concepts in exploring a problem space. These projects demonstrate the understanding and application of CG knowledge and concepts beyond the typical artificially defined class assignments.

Non-video game Final Projects. The non-game projects are typically either some sort of editor or simulator. Figure 6 shows a keyframe animation editor. In this editor, the user can select and directly manipulate the transforms of one of the four generations of the desk lamp, save the transform state and time as a keyframe, and scroll the timeline. After defining the animation, the user can scrub the timeline to replay/edit the animation and save and retrieve the results into and from a file. This project demonstrates the exemplified outcomes of the class. All of the learning outcomes listed in Section 4.1 can be observed: a functional MVC-based interactive system that includes hierarchical models, multiple camera views under user control, and simple, but, custom GPU shaders.

Figure 7 shows an exploratory simulator where the students were interested in visualizing the propagation of waves in a water tank. This system allows the manipulation of a three-level crane system (not visible in the screenshot) to pick up different geometric shapes and drop the shapes into the water tank. The geometric shapes would sink to the bottom of the tank and generate waves. Due to the three-week time limit, the 100x100 mesh wave-display is relatively elementary. However, wave simulation and water displacement-level are diligently computed.

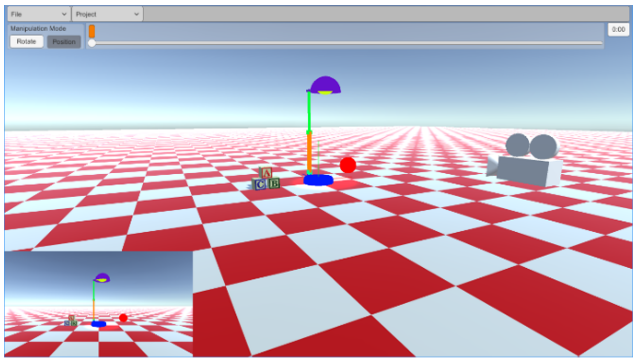

Figure 6: *Keyframe Animation Editor. Credit: Bryan Castillo and Timothy Elmer*

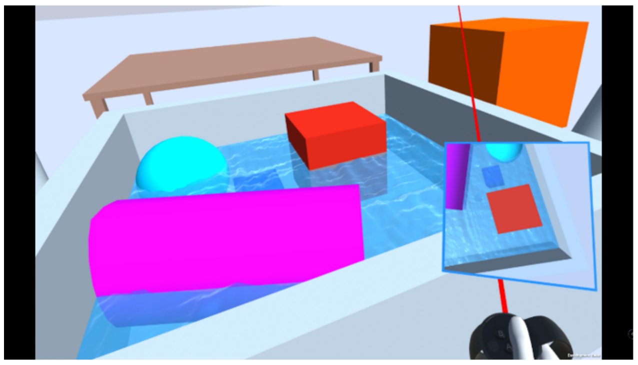

Figure 7: *Tank Wave Simulator. Credit: Archit Gupta and Nasser Alghamdi*

Discussion. The final projects demonstrated learning outcomes that straddle a significant range: from the straightforward simple video games based on a custom defined functionality (Figure 4), to the exemplified learning outcomes functional keyframe animation editor (Figure 6), to the exploration based on CG-concepts projects (Figures 5 and 7). In summary, these projects demonstrated students were able to digest the individual concepts and synthesize solutions in building non-trivial CG applications.

6.2. Student's Anonymous Feedback

After each of the two offerings, students were encouraged to participate in an anonymous survey polling for their opinion on the course (content, organization, instructor contribution) and their own engagements (involvement, intellectual challenge, hours spent). The surveys are typical Likert Scale questions with 5 being the highest. The following are aggregated results from both years.

A total of slightly more than 74% (52 of 70) of the students participated in the anonymous online survey. Overall the students gave the class a satisfactory ranking of 4.6 out of 5 and a challenge and engagement index (CEI) of 5.8 out of 7. CEI provides an estimate of how challenging students found the class and how engaged they were in it [IAS19]. When asked explicitly if they will recommend this elective class to a friend, the result was a 4.2 out of 5.

In agreement with the numeric feedback, students' written comments are also overwhelmingly positive in general. For example, when asked if the class is intellectually stimulating and why, student answers are typically along the line with this response: "*Yes, it offered new concepts in Computer Graphics ranging from learning the Unity game engine, applied vector mathematics, transformations, shaders, and meshes. Being able to apply these concepts to the assignments helped tremendously to grasp the concepts.*".

Although students did not explicitly comment on the CDPs (they are unaware of the term), evidence of their appreciation for the materials can be clearly observed. For example, when asked what aspect contributed most to their learning, many responded similarly to: "*The source code of examples used in class was most helpful in understanding how to apply the concepts taught in class.*" or "*The different Unity and programming examples given for each of the topics covered were invaluable to my learning and understanding of the subject matter.*".

The way we deployed Unity3D as a delivery vehicle turns out to be a double-edged sword. Although there were plenty of positive comments like: "*it was cool to implement what Unity already provides for us*" or, [the factors that contributed most to my learning are:] "*the use of unity, the way to construct projects in MVC*". However, it is also true that almost all negative comments about the class are around how we approach using Unity3D, including: "*There weren't any aspects that detracted from my learning. The initial 1-2 weeks are just struggling with Unity itself.*" or "*Model, view controller can be hard while initially learning Unity.*".

Clearly, more time should be spent covering Unity3D. However, the dedication of class time may lead to the confusion that the class is about the commercial system. We plan to remedy this by offering tutorial sessions outside of class time. In general, students' numeric and written feedback convey a consistent message: they are satisfied with the course as a whole, they appreciate the technical topic coverage, they find the teaching materials useful, and they enjoy learning the concepts in the context of a commercial product.

7. Conclusions

Our game-themed 2D CG class went through exciting evolution cycles [SSRR07], [SS04], was welcomed by students, and had led to rewarding work with students and colleagues [SPH*11]. However, it is also true that we were frustrated by the constraint of 2D space. Our attempts at expanding the custom infrastructure to support moderately complex 3D CG applications proved to be overwhelming for both the faculty and more importantly the students.

The described course was developed over multiple years before it was first offered in 2017. We have defined a clear guideline for designing a commercial product based introductory CG class, followed the guideline and designed a product independent course. Our course materials, both the CDPs and programming assignments, are effective tools and can be ported to any viable commercial product. Students enjoyed the class and appreciated the learning platform. Most importantly, although the class was delivered based on Unity3D, we have successfully educated practitioners of CG. We are encouraged and excited by these initial results.

However, we have also learned that there are many aspects of the course that can be improved. This improvement includes implementation details like the messy and potentially labor-intensive logistics in enforcing which commercial application specific functions students can use. Other improvements are typical continuous refinements of courseware materials, e.g., though effective, the lecture notes for the CDPs can be improved with more precise 3D illustrations. There are also longer-term issues without immediate solutions. For example, it would be greatly beneficial if there was a coherent textbook that effectively integrates concepts and implementations based on the commercial product.

The final and just as important perspective of a class outcome is from the viewpoint of the faculty. It is encouraging and rewarding that after the class, many students contacted the faculty and continued to engage in CG-related exploratory projects. Among these include results that are presented at international conferences including: collaboration in video games [FCS18], locomotion in Virtual Reality [AS18], and multi-view in AR applications [HS18].

Acknowledgements

Thank you to the students from both years for helping with the refinement process, Yusuf Pisan and Lesley Kalmin for providing feedback on the initial draft of the paper, and the reviewers, whose detailed comments significantly improved the quality of this paper.

References

[AB15] Ackermann P., Bach T.: Redesign of an Introductory Computer Graphics Course. In *EG 2015-Education Paper* (2015). 2

[ACM19] ACM: Curriculum Recommendations, Last Access: 3/2019. URL: https://www.acm.org/education/curricula{-}recommendations. 1

[ACSS06] Angel E., Cunningham S., Shirley P., Sung K.: Teaching computer graphics without raster-level algorithms. In *SIGCSE'06* (2006), pp. 266–267. 2

[AP10] Anderson E. F., Peters C. E.: No More Reinventing the Virtual Wheel: Middleware for Use in Computer Games and Interactive Computer Graphics Education. In *EG 2010-Education Paper* (2010), Kjelldahl L., Baronoski G., (Eds.). 2

[AS12] Angel E., Shreiner D.: *Interactive Computer Graphics - a Top-Down Approach using OpenGL*, 6th ed. Addison Wesley, 2012. 2, 4

[AS18] Albert J., Sung K.: User-centric classification of virtual reality locomotion. In *ACM VRST* (2018), pp. 127:1–127:2. 8

[BC11] Bailey M., Cunningham S.: *Graphics Shaders: Theory and Practice*, 2nd edition ed. A.K. Peters, 2011. 4

[BWF17] Balreira D. G., Walter M., Fellner D. W.: What we are teaching in Introduction to Computer Graphics. In *EG 2017-Education Paper* (2017). 1, 2

[CBBO88] Cunningham S., Brown J. R., Burton R. P., Ohlson M.: Varieties of computer graphics courses in computer science. In *SIGCSE'88* (1988), pp. 313–313. 2

[CHV*02] Carroll J., Howard S., Vetere F., Peck J., Murphy J.: Just What Do the Youth of Today Want? Technology Appropriation by Young People. In *HICSS'02* (2002), pp. 1777–1785. 1

[Cli15] Cliburn D. C.: Teaching Shader Programming Through Team-based Learning in a Computer Graphics Course. *J. Comput. Sci. Coll. 31*, 2 (2015), 11–17. 2

[CRC19] CRCS: The Cross Reality Collaboration Sandbox Group, Last Access: 3/2019. URL: http://depts.washington.edu/cssets/CRCS/. 6

[Cun07] Cunningham S.: *Computer Graphics: Programming in OpenGL for Visual Communication*. Prentice Hall, 2007. 1

[CXR18] Chen M., Xu Z., Rippin W.: On the Pedagogy of Teaching Introductory Computer Graphics without Rendering APIs. In *EG 2018-Education Paper* (2018). 2

[DP11] Dunn F., Parberry I.: *3D Math Primer for Graphics and Game Development*, second edition ed. CRC Press, 2011. 4

[FCS18] Fiebelkorn N., Clark B., Sung K.: Would Gamers Collaborate Given the Opportunity? In *FDG'18* (2018), pp. 47:1–47:4. 8

[HB03] Hearn D., Baker P.: *Computer Graphics with OpenGL*, third ed. Prentice-Hall, 2003. 2

[HCGW99] Hitchner L., Cunningham S., Grissom S., Wolfe R.: Computer graphics: the introductory course grows up. In *SIGCSE'99* (1999), pp. 341–342. 2

[HS18] Hitchcock A., Sung K.: Multi-view augmented reality with a drone. In *ACM VRST* (2018), pp. 108:1–108:2. 8

[Hu10] Hu H. H.: Teaching Introductory Computer Graphics via Ray Tracing. *J. Comput. Sci. Coll. 26*, 2 (2010), 30–38. 2

[IAS19] IASystem: Interpreting Reports, Last Access: 3/2019. URL: http://iasystem.org/wp-content/uploads/2015/05/IASystem-Interpreting-Reports.pdf. 7

[Lew12] Lewis R. R.: Coaster: Teaching Computer Graphics with a Comprehensive Project âĂŞ Work in Progress. *J. Comput. Sci. Coll. 28*, 1 (2012), 192–199. 2

[May19] Maya: Maya Animation System, Last Access: 3/2019. URL: https://www.autodesk.com/products/maya. 1

[MS16] Marschner S., Shirley P.: *Fundamentals of Computer Graphics*, 4th edition ed. CRC Press, 2016. 4

[SPH*11] Sung K., Panitz M., Hillyard C., Angotti R., Goldstein D., Nordlinger J.: Game-Themed Programming Assignment Modules: A Pathway for Gradual Integration of Gaming Context into Existing Introductory Programming Courses. *IEEE Transactions on Education 54*, 3 (2011), 416 –427. 7

[SS04] Sung K., Shirley P.: A Top-Down Approach to Teaching Introductory Computer Graphics. *Computer & Graphics 28*, 3 (2004), 383–391. 2, 4, 7

[SSB08] Sung K., Shirley P., Baer S.: *Essentials of Interactive Computer Graphics*. A.K. Peters, 2008. 4

[SSRR07] Sung K., Shirley P., Reed-Rosenberg R.: Experiencing aspects of games programming in an introductory computer graphics class. In *SIGCSE '07* (2007), pp. 249–253. 1, 2, 3, 7

[Sun19] Sung K.: 3D Computer Graphics, Last Access: 3/2019. URL: http://courses.washington.edu/css451. 5

[TYC*17] Tanaya M., Yang K., Christensen T., Li S., O'Keefe M., Fridley J., Sung K.: A Framework for analyzing AR/VR Collaborations. In *CIVEMSA'17)* (June 2017), pp. 111–116. 6

[Uni19] Unity3D: Unity: Game Development Tool, Last Access: 3/2019. URL: http://unity3d.com/. 1

[Vol19] Volkan: Volkan api, Last Access: 3/2019. URL: https://www.khronos.org/vulkan. 2

[Wer12] Werner M.: Graphics Programming on Android. *J. Comput. Sci. Coll. 27*, 6 (2012), 76–77. 1

[Wik19] Wikipedia contributors: List of 3d computer graphics software — Wikipedia, the free encyclopedia, 2019. [Online; accessed 7-March-2019]. URL: https://en.wikipedia.org/?title=List_of_3D_computer_graphics_software. 4

[Wol12] Wolff D.: How Do We Teach Graphics with OpenGL? *J. Comput. Sci. Coll. 28*, 1 (2012), 185–191. 2