Experiencing Aspects of Games Programming in an Introductory Computer Graphics Class [*]

Kelvin Sung
Computing & Software Sys.
Univ. of Washington, Bothell
Bothell, Washington
ksung@u.washington.edu

Peter Shirley
School of Computing
Univ. of Utah
Salt Lake City, Utah
shirley@cs.utah.edu

Becky Reed Rosenberg
Teaching and Learning Center
Univ. of Washington, Bothell
Bothell, Washington
breed@uwb.edu

ABSTRACT

Our computer graphics (CG) programming class uses games development as a means to help students understand CG concepts. Many students mistakenly thought this CG class was a games programming class. We present a simple framework for discussing games programming classes. Based on the framework, the paper describes our efforts in integrating competencies associated with games programming into our CG programming class. Our results show that the resulting class maintains the integrity of the original CG class while allowing students to develop projects with more interesting games features.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education – *computer science education*

General Terms

Design, Experimentation

Keywords

Graphics Programming, Games Programming, Curriculum

1. INTRODUCTION

We have designed and implemented a *top-down* approach to teaching computer graphics (CG) where we study fundamental concepts in the context of designing and implementing moderately complex CG applications [22]. To show real-world applicability, we have built many "concept demonstration applications" (CDAs) to present CG concepts. Our top-down implementation has been well received by the CG education community [24, 2]. We have been refining the CDAs [20] and are currently funded by the National Science Foundation (NSF) to formally study the effectiveness of our CG class [21].

Since adopting the top-down approach, we have observed a steady enrollment increase in the CG class. Many of the students in that class chose to develop interactive game-type applications as their final project and have attracted considerable attentions from their peers. After graduation, some of these students have pursued careers in games development. For example, some of our graduates took jobs at games companies/studios including Crankypants, Electronic Arts, Flying Labs, Amaze Entertainment, Griptonite, Handheldgames, Her Interactive, Humongous Entertainments, Wild Tangent. As a result, students in our department started referring to the CG class as "*the games programming class.*" Ordinarily, we would not be overly concerned with how students refer to any particular class. However, associated with the name reference seem to be students' expectations. Students started to direct computer games specific questions toward this class such as "*which games engine would be studied.*"

Our introductory CG class is not designed to be and does not contain the necessary intellectual contents [13] to be a games programming class. We have no interest in changing our CG class into a games programming class. In addition, our funding obligations [20] and the on-going assessment project [21] dictate that we must maintain the basic philosophical approach: learning fundamental CG concepts in the context of designing and implementing moderately complex CG applications. Instead of confronting students to re-align their expectations, encouraged by the recent successes (e.g., [4, 3, 15]), we investigated and exploited the possibilities of maintaining the academic integrity of an introductory CG class while addressing students' intellectual curiosity about games programming.

In the next section, we present a simple framework that helps categorize existing games programming classes. Base on this framework, Section 3 analyzes the content of our CG class and identifies the missing competencies that are associated with games programming. Section 4 describes our efforts in integrating some of the topics from games programming into our CG class. Section 5 concludes the paper with a discussion of our results.

2. BACKGROUND

To integrate aspects of "games programming" into our CG class, we must understand existing games programming

[*]This work is supported in part by the National Science Foundation grant DUE-0442420.

classes. When we examine recent publications that relate to "games" and "programming" we observe three general categories of classes listed below.

1. *Games development classes.* These are entire curricula (e.g., [6]), individual classes (e.g., [18, 16]), or capstone projects (e.g., [14, 19]) designed specifically to develop new games as an end product. When evaluated against the curriculum framework proposed by the IDGA education committee [13], we see that these classes cover all the major *core topic* areas. Students in these classes must be concerned with all aspects of a *real game production* including entertainment value, visual quality, audio effects, physics simulations, and real-time performance.

2. *Games programming classes.* These are classes (e.g., [25, Kuffner's CMU course]) designed specifically to study technical aspects and issues involved in building games. For example, topics covered may include event loops, path planning algorithms, terrain representation, etc. These classes typically do not require building an end product and the topics covered are general and typically can be applied to different domains. These classes concentrate on covering the *game programming* topic area in the IDGA curriculum framework.

3. *Games development client.* These are existing CS classes that creatively integrate games into their existing curriculum. Typically, games are used as programming assignments (e.g., [1, 10, 11, 23, 25, 3, 15]) or to teach abstract concepts (e.g. [4, 11, 12, 5, 9, 17]), or as an example application area to teach the concepts involved in an entire topic area (e.g., [8, 7]). These are *traditional* CS classes that exist independent from games programming. These classes are actually *clients* of games development where they *use* games development as a vehicle to deliver specific abstract concepts. After these classes, students are expected to understand the abstract concepts, and not about games development.

Our CG class emphasizes the understanding of basic concepts and implementing of these concepts in interactive applications. Because of the "interactivity" and "graphics" components, it is trivial and natural to build "games themes" into the programming assignments and we have. All of the programming assignments in our CG class involve user control of graphical objects that move and interact [22, 24]. These assignments use "games programming" as a vehicle for delivering CG concepts. Competencies in games development are not part of the consideration in the design of the assignments. In this way, our original top-down CG class belongs to the third category: a *games development client.*

3. DESIGN CONSIDERATIONS

Based on informal discussions, it is clear that our students are interested in acquiring competencies associated with both *games programming* and *games development.* In the long term, as an academic department we must determine the feasibility of creating new classes to support our students' needs. In the short term, we challenged ourselves with two related questions:

> *Is it possible to integrate meaningful aspects of games programming and development into our CG class without sacrificing the intellectual integrity of the course? If so, how?*

To begin answering these questions, it is helpful to compare the topic areas covered by interactive CG programming classes (e.g., [24]) to those covered by games programming classes (e.g., [18]):

- *Common to both:* software architecture and programming framework. Both of these classes require students to understand and practice event-driven programming based on Model-View-Controller architecture framework.

- *In-depth coverage in CG:* transformations, geometric modeling, scene graphs, animations, viewing, projection, illumination, shading, rendering buffers, rendering effects, texture mappings [2]. These are "*standard*" topics in CG classes. It is interesting that even though not covered at the conceptual level, games programming classes typically use the results of these topics to communicate to the user.

- *Additional essential topics for games programming:* Newtonian physics, scripting architecture, sprite animation, object resource managements, audio programming, in-game artificial intelligence (AI), networking, persistent game state, file format. These are technical topics usually not covered in a "*typical*" CG class.

The above comparison points out that to systematically expose students to the competencies associated with games programming, we must at least touch on topics in the last category. Our challenge then is to find ways to cover or touch on these additional topic areas *without* sacrificing the core topic areas of a CG class.

We should point out that missing from the above discussions are important *qualitative* issues that must be addressed by games development classes. For example, degrees of fun, social impact, general aesthetic appeal, etc. These are issues we dare not even begin to consider in a CG class.

4. IMPLEMENTATION

We have two simple principles guiding our efforts in integrating games programming topics into our CG class: (1) This is a CG class and CG learning will be the only factor governing the scheduling the presentation of the materials. (2) We will *not* dedicate lecture time to cover topics specific to games programming.

The above seems to imply we are looking for a *free ride* to cover extra topics in games programming. Quite on the contrary, the strict guidelines imply careful planning and gradual integration. Over the past couple of years, based on different strategies, we have managed to integrate some aspects of games programming into our CG class.

4.1 Physics and In-Game AI

A major characteristic of our top-down approach to teaching CG is in the CDAs (Concept Demonstration Applications). The CDAs are custom built, Model-View-Controller based, event driven interactive graphics programs that demonstrate foundational CG concepts. We have modified appropriate CDAs to include simple physics computation in event service routines. In other CDAs, we have maintained persistent application state (game state) with build-in randomness based on users' input (rudimentary AI). Below are some examples of our modifications. For the CDAs demonstrating timer event services, we modified the service routine to include a free falling circle. For the CDAs demonstrating extents of graphical primitives, we modified the collision detection routines to compute and react to perfectly elastic collisions when primitive bounds are violated. For the CDAs demonstrating multi-level scene graphs, we modified the main timer service routine to *shoot* circles towards one of the leaf nodes; allow student interactive control over the position of the leaf node; and alter the shooting rate and direction based on the leaf node position (rudimentary AI). For many other CDAs, we modified their implementation to demonstrate other physical effects, e.g., friction, deceleration, left/right turning of primitives, "home-in" automatic target seeking, etc.

We use these CDAs to demonstrate/discuss CG concepts as before, except that we briefly mention the "*extra*" games programming features. In this way, we do not dedicate lecture time discussing physics or in-game AI and yet students have the opportunity to examine source code of simple examples of these topics. From a games programming point of view, the CDAs seem to scatter important topics into unrelated examples that are covered according to unrelated CG concepts throughout the academic quarter. For example, it would be awfully difficult to try to learn physics simulations, or build a physics engine based on our CDAs. However, from the CG point of view, the CDAs present CG topics in a coherent manner. Presently, there are more than 100 CDAs publicly accessible on our course web sites at:

http://courses.washington.edu/css450
http://courses.washington.edu/css451

4.2 DirectInput and Audio Libraries

Input mechanisms are discussed in our course as part of event services and user interactions. However, due to time constraints, we do not use a specific input API. In particular, we do not cover keyboard input because the CG topic coverage does not demand it. Audio is an even more remote topic. These topics are not covered in our CG class and there are no associated CDAs for these topics.

We understand the importance of keyboard input and sound effects in games. To address this knowledge gap, we engage students in independent projects outside of classrooms. Through these projects students investigate relevant APIs, and develop detailed tutorials demonstrating how to integrate such libraries into applications. We have collected tutorials for working with the *DirectInput* library, as well as the *Bass* and *FMOD* audio libraries. These tutorials have become standard resources on our course web sites. Students are expected to read these and integrate keyboard input and audio effects into their projects.

4.3 Sprite Animation

As part of texture mapping, we do mention that cycling through appropriate texture maps during subsequent refresh cycles can create interesting "animation" effects. However, from a CG perspective, there is no real need to go into the details of programming with sprites. On the other hand, we do demand students program with masks and alpha blending. As a result, more motivated students often create interesting animations in their projects by cycling through alpha-blended texture maps.

4.4 Assignments and Projects

Since the above topics are not covered in lectures, we rely on programming assignments to assist in organizing students' exposure to these topics. Physics simulation and AI requirements are integrated into all programming assignments, and successive ones build upon previous results and increase in complexity. For example, the first programming assignment is designed to practice event driven programming. As part of the assignment, students must simulate projectile motion under gravity where the user has control over the world bounds, initial velocity, and gravitational acceleration. The next assignment builds on this and requires students to implement collision between moving and stationary objects. Eventually, students must program the collision of multiple objects in motion. In the final project for the course, students must synthesize their knowledge by building a game-like application that includes all of the CG *and* games programming topics described above.

4.5 Discussions

With the above integration efforts, we have managed to expose students to *some* interesting topics in games programming that are not usually covered in CG classes. There are many important aspects of games programming that are still absent from our current course materials. Some of these topics e.g., file format, and object resource management, are potential candidates for future integration into relevant CDAs. For other topics that require tight integration and dedicated software infrastructure support, e.g., scripting, our minimalist "*on-the-side*" type approach may simply not work.

The above integration efforts involve work done mostly *before* the offering of the CG class. During the course we dedicate minimal lecture time on this material, and rely heavily on student initiative. Students are expected to understand simple linear algebra implementation, and learn new APIs. With this extra effort, students can expect *limited* experience and competencies in the related aspects of games programming. Without dedicated lecture time one cannot expect in-depth understanding.

5. RESULTS

We begin experimenting with the top-down approach to teaching CG in 2000. By 2002, we had a somewhat stable version of courseware material. That was about the time we began noticing "*games programming*" expectations. The games programming related materials were introduced over the past two years as part of an on-going assessment project [20, 21]. Throughout these times, we used the most up-to-date materials when we teach. Since the materials evolved gradually and students from different years were exposed to slightly different versions, it is difficult to draw subtle *before/after* observations.

Fortunately, also as part of the on-going assessment project, we monitor student learning with pre/post-tests, and continuously polling students on self-reflections. In the pre-test, we assess student's prerequisite (e.g., programming skills) and CG (e.g., terminologies, concepts) knowledge. During the course, we ask students to reflect on what is easy, diffi-

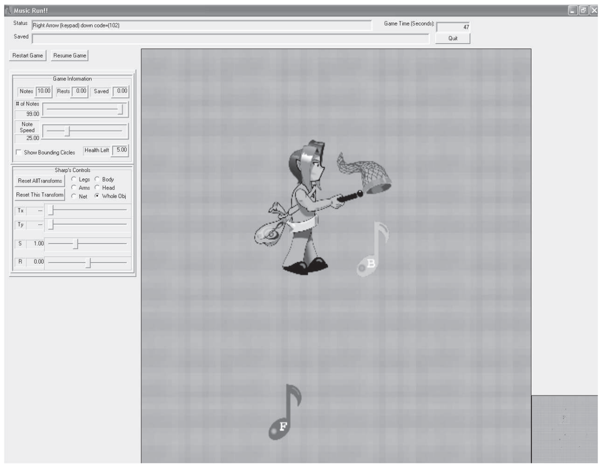

Figure 1: *Music Run* (Fall 2002).

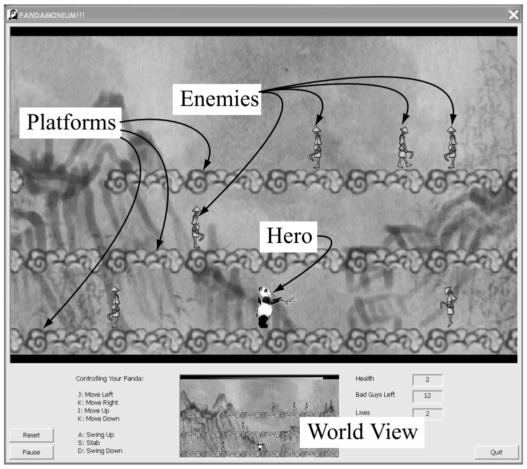

Figure 2: *Pandamonium* (Fall 2004).

cult, or helped them learn, etc. The post test evaluates the students' grasp on CG concepts. In addition, we also collect the "typical" student course evaluation forms. With these information, we evaluate our work in the following three ways.

1. Effects on learning CG. One concern with our work is that the additional information would divert students' attention and thus affect their learning of CG concepts. Fortunately, this concern is unfounded. The post-test results showed remarkably little variations over the years.

2. Comments from students. Except during the initial years (2000/2001) when the CDAs were under major developmental changes, student feedback from our CG class has been along a similar theme: a lot of work, and the course is challenging and fun. It is interesting that even as the CDAs are refined to include extra complexities; students are required to self-learn additional APIs; and the assignments are biased increasing towards full-blown games, the types and distribution of student comments do not show significant changes. Examples of student comments include, "*took up all my free time*," "*challenging class*," and "*had too much fun with projects*,". These types of comments were distributed in similar fashions over the years. The numeric rating for the class has also remain somewhat constant at around 4.5/5.0. Based on student feedback, it is as though nothing much has happened.

3. Quality of student projects. Figure 1 shows *Music Run*, a final project from Fall 2002. This was one of the projects we referred to when we describe the top-down approach to teaching CG in [24]. Figure 2 shows *Pandamonium*, a final project from Fall 2004[1]. The two projects have similar technical CG contents. For example, both contain two user-manipulable views of the world; both have a multi-level scene graph defined *hero* object that is under user's control; and in both cases, the hero object can interact with other elements in the environment, etc. However, the project from 2004 do *appear* busier, with a more interesting environment. For example, the hero can *jump*, with gravitation effects, to different platform levels to battle with the enemies. Finally, and very importantly, the project from 2004 provides a more pleasant game play experience with interesting audio effects and familiar keyboard controls. These characteristics are generally true that student projects from latter years appear busier and support more pleasant game play experiences.

[1]Due to a sabbatical interruption, this CG class was not offered in 2005.

Over the past few years we have systematically increased the complexities of the course materials without dedicating lecture time to cover the extra materials. At the same time, we have increased students' workload and demanded extra features in their assignments. It is interesting that, from the feedback, students' opinion of the class did not show any changes. One may conclude that the class was too easy before the changes, but we beg to differ. We believe there are two independent factors at play. First, the extra workload are related to *games programming* and are *fun* for the students. They probably did not mind the extra learning. Second, the refined CDAs are better documented and better organized. With the improved CDAs, students should be able to comprehend the concepts more efficiently and thus might not have spent significantly more time on this class.

6. ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation grant DUE-0442420. All opinions, findings, conclusions, and recommendations in this work are those of the authors and do not necessarily reflect the views of the National Science Foundation. Thanks to the reviewers for the excellent and critical suggestions. Steve Baer has redeveloped most of the CDAs in CSS450, the refinement effort is supported by the University of Washington, Bothell, Worthington Scholar Award. *Music Run* is developed by Aida Sakkal; and *Pandamonium* is developed by Mike Shaver.

7. REFERENCES

[1] J. C. Adams. Chance-it: an object-oriented capstone project for cs-1. In *SIGCSE '98: Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, pages 10–14, New York, NY, USA, 1998. ACM Press.

[2] E. Angel, S. Cunningham, P. Shirley, and K. Sung. Teaching computer graphics without raster-level algorithms. In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 266–267, New York, NY, USA, 2006. ACM Press.

[3] J. D. Bayliss and S. Strout. Games as a "flavor" of cs1. In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 500–504, New York, NY, USA, 2006. ACM Press.

[4] K. Becker. Teaching with games: the minesweeper and asteroids experience. *J. Comput. Small Coll.*, 17(2):23–33, 2001.

[5] M. C. Carlisle, T. A. Wilson, J. W. Humphries, and S. M. Hadfield. Raptor: a visual programming environment for teaching algorithmic problem solving. In *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education*, pages 176–180, New York, NY, USA, 2005. ACM Press.

[6] R. Coleman, M. Krembs, A. Labouseur, and J. Weir. Game design & programming concentration within the computer science curriculum. In *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education*, pages 545–550, New York, NY, USA, 2005. ACM Press.

[7] F. S. C. da Silva. Artificial intelligence for computer games, 2006. University of Sao Paulo (USP/SP), Microsoft Academic Alliance Repository Newsgroup, Object ID: 6210, *http://www.msdnaacr.net/curriculum/pfv.aspx?ID=6210*.

[8] F. S. C. da Silva. Software enginnering for computer games, 2006. University of Sao Paulo (USP/SP), Microsoft Academic Alliance Repository Newsgroup, Object ID: 6211, *http://www.msdnaacr.net/curriculum/pfv.aspx?ID=6211*.

[9] W. Dann, S. Cooper, and R. Pausch. *Learning to Program with Alice*. Prentice Hall, Upper Saddle River, NJ, 2006.

[10] N. Faltin. Designing courseware on algorithms for active learning with virtual board games. In *ITiCSE '99: Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education*, pages 135–138, New York, NY, USA, 1999. ACM Press.

[11] R. Giguette. Pre-games: games designed to introduce cs1 and cs2 programming assignments. In *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education*, pages 288–292, New York, NY, USA, 2003. ACM Press.

[12] S. Hansen. The game of set&#174;: an ideal example for introducing polymorphism and design patterns. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 110–114, New York, NY, 2004. ACM Press.

[13] IDGA. IGDA curriculum framework report version 2.3 beta, February 2003. International Game Developer's Association, *http://www.igda.org/academia*.

[14] R. M. Jones. Design and implementation of computer games: a capstone course for undergraduate computer science education. In *SIGCSE '00: Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*, pages 260–264, New York, NY, USA, 2000. ACM Press.

[15] M. C. Lewis and B. Massingill. Graphical game development in cs2: a flexible infrastructure for a semester long project. In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 505–509, New York, NY, USA, 2006. ACM Press.

[16] B. Maxim. Game design and implementation 1 and 2, 2006. Microsoft Academic Alliance Repository Newsgroup, Object ID: 6227, *http://www.msdnaacr.net/curriculum/pfv.aspx?ID=6227*.

[17] MUPPET. Multi-user programming pedagogy for enhancing traditional study, 2006. Rochester Institute of Technology, *http://muppets.rit.edu/muppetsweb/people/index.php*.

[18] I. Parberry, M. B. Kazemzadeh, and T. Roden. The art and science of game programming. In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 510–514, New York, NY, USA, 2006. ACM Press.

[19] I. Parberry, T. Roden, and M. B. Kazemzadeh. Experience with an industry-driven capstone course on game programming: extended abstract. In *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education*, pages 91–95, New York, NY, USA, 2005. ACM Press.

[20] K. Sung. Courseware infrastructure development for supporting proposals to the nsf. *Worthington Scholar Award, University of Washington, Bothell*, June 2004-2006.

[21] K. Sung. Essential concepts for building interactive computer graphics applications. *CCLI-EMD, NSF, DUE-0442420*, 2005-2007.

[22] K. Sung and P. Shirley. A top-down approach to teaching introductory computer graphics. *ACM SIGGRAPH 2003 Educator's Program*, July 2003. Conference CD/DVD-ROM Disc 1.

[23] K. Sung and P. Shirley. Teaching computer graphics programming to non-traditional returning adult students. *Extended Abstract in Eurographics/ACM SIGGRAPH Workshop on Computer Graphics Education 2004*, June 2004.

[24] K. Sung and P. Shirley. A top-down approach to teaching introductory computer graphics. *Computer and Graphics*, 28(3):383–391, June 2004. Invited full paper based on SIGGRAPH 2003 conference-paper.

[25] E. Sweedyk, M. deLaet, M. C. Slattery, and J. Kuffner. Computer games and cs education: why and how. In *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education*, pages 256–257, New York, NY, USA, 2005. ACM Press.