Creating Interest in Computer Graphics by Teaching Game Development

Ashish Amresh[†1] and Pushpak Karnick[‡1]

[1]Arizona State University, Tempe AZ

Abstract

This paper describes our experiences in designing and implementing a junior/senior level undergraduate course in Game Programming at Arizona State University. We highlight the pedagogical methods employed during each of the three semesters that the class was offered. We compare our approach with established teaching methods and highlight the advantages of implementing our novel approach for teaching game development. We establish a strong connection between teaching game programming and computer graphics and show how an introductory game programming class can be an excellent way for getting junior and senior level undergraduate students excited about computer graphics.

Categories and Subject Descriptors (according to ACM CCS): K.3.1 [Computer Uses in Education]: Computer-assisted instruction (CAI) K.3.2 [Computer and Information Science Education]: Computer science education

1. Introduction

A recent study shows a sharp downturn in young people selecting a computing field for their careers [Pat05]. This is true for computer science, computer engineering, information technology, information systems, and software engineering. In computer science, the numbers of incoming freshmen fell by 60 percent between 2000 and 2004. The percentage of all college freshmen planning a major in computer science dropped to 1.4 percent in 2004, down from its peak of 3.4 percent in 1998 and below a trough of 1.6 percent in 1992–93 [Fos05]. To compound the problem, internal drop rates of 35 to 50 percent are common. Computer graphics (CG) faces graver problems as the number of computer science undergraduates enrolling in CG courses has traditionally been low.

This paper describes our experience in teaching a junior/senior undergraduate level course in the field of game programming. The objective of the course was to introduce basic concepts employed in the creation of modern computer games. Students had to create their own interactive application by integrating a variety of open source software, as well as implementing the concepts taught in class. The course was designed in a way that did not enforce prior knowledge of computer graphics, only a strong grasp of data structures was assumed. With an eye on the rapidly growing game industry, emphasis was placed not only on implementing original code, but also on looking "under the hood" of open source libraries to acclimatize students in working with a large codebase. We found that this approach was a novel way of getting our students motivated to learn more about various aspects of computer graphics in general, and game development in particular.

We state the contributions of our approach below:

1. Our hands-on method of teaching is a novel approach that overcomes some drawbacks of traditional teaching methods.
2. We show a strong correlation between our method and observed enrollment figures for computer graphics courses before and after our course was introduced.

The paper begins by discussing the motivating factors that led to the introduction of this course in the Computer Science and Engineering Department (CSE) at Arizona State University, Tempe (ASU). In section 2 we discuss current and past work done in the area of teaching game development and using it as a tool for teaching computer graphics.

† email:amresh@asu.edu
‡ email:pushpak@asu.edu

We explain our approach in detail in section 3. We compare our approach with existing teaching strategies in this section, and conclude in section 4, discussing future directions and a portability plan for our course.

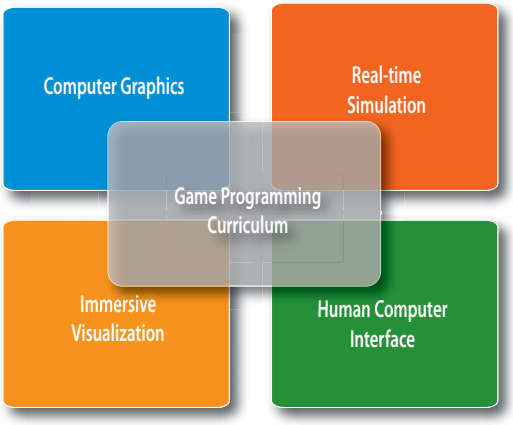

Figure 1: *Game Programming in the context of allied disciplines.*

1.1. Motivation

The course "*CSE494/598: Introduction to Game Programming*" was offered at ASU for the three semesters of Fall 2005, Spring and Fall 2006, as a junior/senior undergraduate level course in the CSE catalog. This course was introduced with the following goals:

1. **Addressing an impending need in the undergraduate CSE student community:** The CSE department wanted to make inroads into teaching game programming for addressing the needs of its undergraduates and also to keep a competitive academic edge over other universities.
2. **Generating Interest in Computer Graphics:** The CSE department, keen on increasing the undergraduate student intake for computer graphics, utilized this opportunity to increase general awareness about computer graphics research options at ASU via this course.
3. **Enhancing Student Enrollment:** Another goal of this endeavor was to study the student enrollment statistics before and after the inception of the course. This turned out to be a deciding factor in the expansion of the course into a comprehensive curriculum.
4. **Teaching Experience:** The authors, who were both doctoral students in 2005, wanted to leverage their past work experience in the game industry for designing and teaching an undergraduate course in the area of computer games.

2. Related Work in Gaming Curriculum Development

Game development is no longer considered an esoteric 'black art' of its early days. *Ludology*, the scientific study and analysis of games, has emerged at the forefront of efforts in laying out "*...an aesthetic approach to interactive systems* [SZ04]." There exists a substantial volume of work that explores the relevance of video games in the context of current pedagogical processes [Squ03, BS06, HS05]. Computer game programming, on the other hand, is steadily gaining acceptance as a mainstream academic track [PKR06]. Institutions like DigiPen (Redmond) [DIG], Guildhall (SMU) [GUI], UAT (Phoenix) [UAT] now offer specialized degrees/diplomas in game design and development. Similar efforts [PRK05, PKR06, BS06] have proven to be successful in enhancing enrollment for the Computer Science Departments. Tori et.al. [TJaLBN06] discuss teaching introductory computer graphics using Java3D^{TM}. Coleman et. al. [CKLW05] describe their successful initiative to introduce a "Game Concentration" within the traditional computer science curriculum.

While the above approaches have demonstrated that games can serve as a vehicle for teaching computer science, they are often *ad-hoc* in their implementation and involve an additional overhead in training/hiring faculty members in the process. Additionally, game programming, though an independent applied field in itself, overlaps strongly with the more established branches of Computer Graphics, Real-time Simulation, Human-Computer Interface, and Immersive Visualization (see figure 1). A curriculum design for gaming would certainly have to consider this overlap. Thus, a course that aims to teach game programming should consist of elements from each of the overlapping domains. This formed the basic principle of our course design that we describe in section 3.

2.1. Computer Graphics Instructional Models

Traditionally, introductory courses in computer graphics have been taught in a "bottom-up" fashion: starting from 2D scan-line algorithms and building slowly to graphical primitives and transformations , modeling, texturing and animation. This approach follows a classical textbook approach [FvDFH90] in computer graphics and places emphasis on explaining the basic building blocks of computer graphics. It covers fundamental algorithms in detail and expects the students to have a solid mathematical foundation. This approach is best suited for undergraduate students who are serious about learning the intricacies of modern graphics pipeline, but at the same time is limiting for the casual graphics enthusiast.

The essence of the above approach is that it teaches the basic mathematics and methodology of graphics engine design with lots of low-level details. Though best suited for a research oriented environment, it does not keep pace with the

developments in the game industry, which often happen at a more abstract level and where the prime emphasis is on rapid prototyping of demo applications. For the casual enthusiast, instant gratification can only be made available with the help of an encapsulated graphics API that enables rapid creation of interactive applications as a proof-of-concept implementation. A bottom-up approach is helpful in recognizing the intricacies of off-the-shelf graphics tools, but when the focus of the student is learning to use these tools, this approach is often an overkill. In such cases, the top-down approach of abstracting lower-level details works best to fulfil the students's requirements, see [Ang05]. Such abstractions can be implemented via high-level API libraries in Java/C++ that encapsulate the lower level functionality (game engines). Recent methods have also introduced CG curriculum that require little, or no, programming effort [Ove04].

2.2. Game Programming Instructional Models

Game programming has mostly adopted the "top-down" approach and applied a few variations to it in courses offered in various universities across the world [Ste, Ale]. Most of these courses use an established game engine as the underlying API, whether it be a commercial engine like Quake [QUA] or Unreal [UNR], or an educational version like Torque [TOR]. These courses require the respective universities to have a serious commitment towards advancing the gaming curriculum and invest in hardware/software required to maintain, run and teach these API's. The other approach is geared towards serious gaming and teaching game programming for use in places other than the game industry, for e.g. the military [aNPS], or casual gaming [Par07]. In this approach, custom API's are designed by the universities for teaching the required content. Bottom-up approach exists in the gaming realm as well in the form of courses that are involved in the design and development of gaming engines, tools and advanced API's [Pro, Eng].

3. Our Approach

Our goal was to develop a game development course based on a *student centric design* that aimed towards maximizing the student learning experience. It was a combination of the top-down and bottom-up strategies for maximizing student education both in terms of the theoretical background, and industry-aware curriculum with a "hands-on" model. We had the following goals in mind when we designed the course:

1. **Comprehensive:** The course should cover a typical game production pipeline – from game concept, to game design and eventual implementation.
2. **Modular:** The course should be modular, facilitating a rearrangement of subject topics based on student feedback and interest.
3. **Extensible:** The course syllabus should be designed such that it can serve as a one-semester introductory course, complemented with another one-semester advanced course in game development.
4. **Self Sufficient:** The course material should be self sufficient, alleviating the need, on the student's part, of satisfying often conflicting pre-requisite classes to enroll for the course.
5. **Industry Aware:** The course should provide some exposure to the practices in the game industry especially wrt. team dynamics and project planning.

Semester	Students enrolled	No CSE470	Percentage
Fall 2005	23	15	65.21%
Spring 2006	18	12	66.67%
Fall 2006	21	16	76.1%

Table 1: *Student enrollment statistics showing number of people registered and the number of students who had taken a computer graphics course*

This approach along with our instruction emphasis strategy allowed students without any computer graphics background to enroll in the course. As table 1 shows, more than 60% of our students had not taken the a course on introduction to CG (CSE470 at ASU).

3.1. Course Structure and Outline

We used the book "*Introduction to Game Development*" by Steve Rabin [Rab05] as our textbook for the three semesters that the course was taught at ASU. In addition, the following books were recommended as references:

1. *3D Game Engine Design*, by David Eberly [Ebe00].
2. *OpenGL Programming Guide* [SWND05], the "Red book."
3. *OpenGL Shading Language*, by Randi Rost [Ros06], the "Orange book."

3.1.1. Choice of the Textbook

"Introduction to Game Development [Rab05]", is a comprehensive introductory text by experts in the game industry. This text covers every major aspect of game development, from design to programming to visual arts and the business of gaming, and presents the material with a structure and outline that's appropriate for an introductory course on game development.

The book is modeled after the International Game Developers Association's (IGDA's) guidelines for a first-year game development curriculum [IGD], and covers four broad areas of study: general game studies, game programming, art/asset creation, and business/management. We used the programming areas covered in this book as our main course material.

3.2. Open Source Software and Tools

Our course was structured to be a hands-on interactive experience for the students. Each module was designed to be *application oriented*. To mimic the actual game production pipeline as closely as possible, students worked towards creating their own games in teams, defining the game concept, drawing up their game design and lastly, implementing that design with existing open source software tools and technologies. The primary goal of learning game development concepts was complemented with auxiliary exercises that consisted of documenting the conceptual framework for the games, creating tangible process documents (design document, test-cases, project schedule) and working with already existing codebase, first to gain sufficient understand of its workings, and then, modify it (if needed) to suit the requirements of their project. Our choice of the tools was influenced by the feature set provided by each tool, as well as the learning curve required to grasp a working knowledge of the same (limited to one semester). A brief description of the various software tools and technologies that were used follows:

- OSG (http:www.openscenegraph.org) is a popular open source graphics toolkit with a robust scene graph implementation and large user base. The OSG design model is very extensible and serves as an excellent starting point for developing games without the need for core computer graphics expertise. OSG is now a part of the official OpenGL SDK.
- ODE (http:www.ode.org) is a high-performance library implemented in C++, for simulating rigid body dynamics. ODE integrates easily with other C++ libraries and helps modularize the game design.
- Blender (http:www.blender.org) is a popular modeling tool for authoring game assets. Blender supports a wide range of file formats for input/output, as well as a gentle learning curve as compared to similar software available commercially.
- OpenGL Shader Designer (http:www.typhoonlabs.com) is a freely available IDE for developing and testing shaders in the OpenGL Shading Language (GLSL). Shader Designer helps test and debug most GLSL shaders without the time consuming effort needed in setting up a debugging environment. The web site provides short tutorials on using the tool and example shaders that demonstrate common real-time rendering techniques used in today's games. The IDE is now a part of the official OpenGL SDK.
- Cal3D (http:gna.org/projects/cal3d/) is a skeletal based 3d character animation library written in C++ that is independent of the underlying implementation of the graphics API. Cal3D files can be easily imported/exported from Blender via use of plugins written as Python scripts.
- Delta3D (http:www.delta3d.org) is a full-function game engine that is currently used for development of training and simulation applications. Delta3D integrates OSG, ODE, CAL3D and OpenAL seamlessly and enables rapid prototyping and development of games.

Figure 2 shows the software hierarchy among these tools.

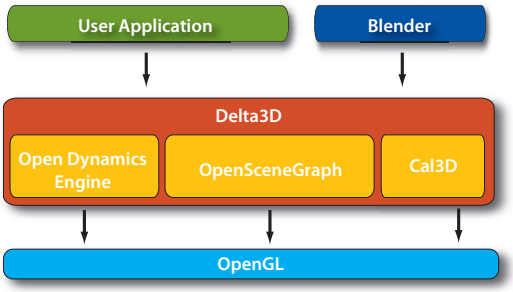

Figure 2: *Hierarchy diagram of the open-source software used during the course.*

3.3. Teaching Strategies

Game programming encompasses different disciplines in computer science and therefore poses a reasonable prerequisite challenge to the undergraduate student. The subject matter (game development) generates considerable interest itself, but students are often wary of their current skill sets and their ability to successfully complete such a course. For us, this was an important factor in determining which areas in computer science needed more teaching emphasis than others. We reviewed the undergraduate course curriculum and decided that the major requirement for taking this course would be an introductory course in data structures. We would have liked to have an introductory course in computer graphics as a requirement, but that would have not served well with our target audience as the graphics courses offered at ASU are not a part of the mandatory core course list for undergraduates. Consequently, a reasonable amount of instruction time was spent in teaching the basic graphics pipeline to bring such students up to speed with their peers. In contrast, areas like software design and architecture, artificial intelligence, software optimization and testing, systems and low-level programming, that constitute major building blocks for modern game development, were not discussed in as much detail as would be covered in a dedicated three-credit course for each area. We discussed the topics within each of these areas that were relevant in their connection to the game development pipeline.

Table 2 shows our course outline for the period of 16 weeks during the semester that was developed with the above observations in mind. The student programming projects were designed to be culminate into a simple game engine with incremental additions for every assignment (see figures 3 and 4). In addition, we hosted speakers from the industry (Microsoft Game Studios, General Dynamics,

Rainbow Studios), that provided the state-of-art information about development practices to the students. The feedback to these talks has been overwhelmingly positive.

3.4. Comparing Our Approach with Other Methods

The student experience by taking our class is compared with the other models of teaching graphics and games described in the previous section. Table 3 shows different action items that the students in our class are exposed to and how these action items are addressed by the other teaching models. Figure 5 shows the student enrollment statistics for the period that CSE 494/598 was offered at ASU. As can be clearly seen, the trend shows an increase over the three semesters that it was offered.

Figure 3: *Project1: Simple Interactive Application to load and manipulate models.*

Figure 4: *Project2: Improves the previous project by adding character animation and collision detection.*

3.5. Issues and Roadblocks

Game programming creates lot of interest in the undergraduates, though this may not always prove to be a positive aspect. Most of our classes got filled up within a week of the

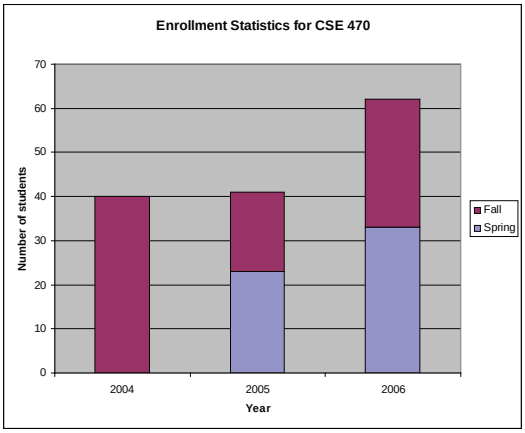

Figure 5: *Enrollment statistics for CSE 470 after our course was introduced. The data shows an overall increase in the number of students who enrolled for CSE 470.*

announcement and as such, many students were not able to register for this class, in spite of their enthusiasm. However, we also faced 20% drop out rate within the first three weeks as some students had enrolled into the class without anticipating the workload and commitment needed to successfully complete the class. This did not do justice to deserving and motivated students who were left out due to limited capacity. On the other hand, open source software had a few perils of its own that we had to overcome, especially with issues related to software set up, integration with other tools, lack of adequate software support and proper documentation, and constant debugging to find errors. We observe that though the software setup pipeline was streamlined over the duration of three semesters, novice students kept repeating the same mistakes during each semester (in spite of FAQs and TA support).

4. Conclusion and Future Work

Conclusion: We have presented a novel and effective approach to introduce game programming to senior undergraduate, and graduate students. Our experience for the past three semesters has proven the efficacy of our methods in popularizing computer graphics among the target audience. We are confident that efforts that involve teaching game programming with an interactive application development as the core focus propels a strong interest in learning more about computer graphics for the students. It also lays a good foundation for undergraduates who may be otherwise hesitant to enroll in a math oriented and technically challenging class in computer graphics.

Future Directions: The popularity of our course within the undergraduate student population has been the most en-

couraging feedback for our efforts. The overwhelming response to this course led CSE to develop a certificate program in game programming and design. This program will be launched in Fall 2008.

Acknowledgments

The authors thank the anonymous reviewers for their constructive comments on improving this paper. We are grateful to Dr. Gerald Farin, and Dr. Dianne Hansford (both ASU) for their constant encouragement and support.

References

[Ale] ALEXA M.: Game programming course 0433 l 370 , accessed april 10, 2007.

[Ang05] ANGEL E.: *Interactive Computer Graphics: A Top-Down Approach Using OpenGL*, 4 ed. Addison-Wesley, March 2005.

[aNPS] AT NAVAL POSTGRADUATE SCHOOL M. I.: http://www.movesinstitute.org/, accessed april 04, 2007.

[BS06] BAYLISS J. D., STROUT S.: Games as a "flavor" of cs1. In *Proc. of the 37th SIGCSE technical symposium on Computer science education* (2006), ACM Press, pp. 500–504.

[CKLW05] COLEMAN R., KREMBS M., LABOUSEUR A., WEIR J.: Game design & programming concentration within the computer science curriculum. *SIGCSE Bull. 37*, 1 (2005), 545–550.

[DIG] DIGIPEN: http://www.digipen.edu/ , accessed april 04, 2007.

[Ebe00] EBERLY D.: *3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics*, 1 ed. Morgan Kaufmann, San Francisco, USA, September 2000.

[Eng] ENGINE S. S. A. G.: http://larc.csci.unt.edu/sage/, accessed april 04, 2007.

[Fos05] FOSTER A.: Student interest in computer science plummets. *Chronicle of Higher Education 51*, 31 (May. 2005).

[FvDFH90] FOLEY J. D., VAN DAM A., FEINER S. K., HUGHES J. F.: *Computer graphics: principles and practice (2nd ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[GUI] GUIDHALL: http://guildhall.smu.edu/ , accessed april 04, 2007.

[HS05] HOETZLEIN R. C., SCHWARTZ D. I.: Gamex: a platform for incremental instruction in computer graphics and game design. In *ACM SIGGRAPH 2005 Educators program* (2005), ACM Press, p. 36.

[IGD] IGDA: Igda game curriculum framework, accessed january 06, 2007.

[Ove04] OVERMARS M.: Teaching computer science through game design. *Computer 37*, 4 (2004), 81–83.

[Par07] PARTRIDGE A.: *Creating Casual Games for Profit and Fun.*, 1 ed. Game Development Series. Charles River Media, February 2007.

[Pat05] PATTERSON D.: Restoring the popularity of computer science. *Commun. ACM 9* (Sept. 2005), 25–28.

[PKR06] PARBERRY I., KAZEMZADEH M. B., RODEN T.: The art and science of game programming. In *Proc. of the 37th SIGCSE technical symposium on Computer science education* (2006), ACM Press, pp. 510–514.

[PRK05] PARBERRY I., RODEN T., KAZEMZADEH M. B.: Experience with an industry-driven capstone course on game programming. In *Proc. of the 36th SIGCSE technical symposium on Computer science education* (2005), ACM Press, pp. 91–95.

[Pro] PROGRAMMING C. . G.: http://larc. csci. unt. edu/class. html , accessed april 04, 2007.

[QUA] QUAKE: id software, (http://www. idsoftware .com), accessed april 04, 2007.

[Rab05] RABIN S. (Ed.): *Introduction to Game Development*. Game Development Series. Charles River Media, Hingham, Massachusetts, 2005.

[Ros06] ROST R.: *OpenGL(R) Shading Language*, 2 ed. Addison-Wesley Professional, January 2006.

[Squ03] SQUIRE K.: Video games in education. *International Journal of Intelligent Simulations and Gaming 2*, 1 (February 2003), 49–62.

[Ste] STEWART K.: 3d game programming courses (http://www.stewart.cs.sdsu.edu/3dgame-prog), accessed april 17, 2007.

[SWND05] SHREINER D., WOO M., NEIDER J., DAVIS T.: *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R), Version 2*, 5 ed. Addison-Wesley Professional, August 2005.

[SZ04] SALEN K., ZIMMERMANN E.: *Rules of Play: Game Design Fundamentals*. MIT Press, 2004.

[TJaLBN06] TORI R., JOÃO LUIZ BERNARDES J., NAKAMURA R.: Teaching introductory computer graphics using java3d, games and customized software: a brazilian experience. In *ACM SIGGRAPH 2006 Educators program* (2006), ACM Press, p. 12.

[TOR] TORQUE: Torque game engine (http://www. garagegames. com), accessed april 04, 2007.

[UAT] UAT: University of advancing technology (http://www.uat.edu/) , accessed april 04, 2007.

[UNR] UNREAL: Unreal engine (http://www.unrealtechnology.com/html/technology/ue30.shtml), accessed april 04, 2007.

Week	Module Focus Area	Concepts Learnt	Software Tools and Source
1	Introduction	Video Games History, Role of Programmer in Game Industry	Textbook and online articles
2	Graphics Pipeline	Basic OpenGL Rendering pipeline	C++, GLUT
3	Interactive Application Development	Writing a simple application to (a) Load 3D model files (b) Position models by applying transforms (c) Basic user interface via keyboard and mouse (d) Selecting and highlighting models.	OSG
4	Game Architecture and Engine	Game Engine Design, Event-based Modeling Vs Polling, Implementation Issues	Game Design Literature
5	Game Math	Affine Transforms, Local and Global transforms, Mathematical Toolkit for Vector and Matrix Algebra	C++, OSG
6	Game Design	Design Principles, Case studies: “Good” and “Bad” designs	Game Design Literature
7	Game Animation	Animation Basics Keyframe Animation using Quaternions	Blender, Cal3D
8	Game Effects	Lighting Models, Texturing, Bump Mapping, Normal Mapping, Multitexturing, Particle Systems, Shadows	C++, Shader Designer
9	Asset Modeling	Mesh Modeling, Level-of-Detail, Texture Atlas generation	Blender
10	Game Physics	Rigid Body Dynamics, Collision Detection and Resolution, Particle Systems, Numerical Methods	OSG, ODE
11	Game AI	Pathfinding Algorithms, Search Strategies, Flocking and Swarming	C++, OSG, Delta3D
12	Game Shaders	OpenGL Hardware Pipeline, Implementing Shaders in GLSL	GLSL, Shader Designer, OSG
13	Group Game Specifications	Create game concept document and design document, plan and schedule a team project	Textbook
14	Game Production	Game Production Pipeline Team Dynamics, Asset Repository Management	Textbook
15	Game Optimization	Spatial Data Structures (Octrees, *Kd* trees), Visibility Solution using Portals	Online articles from Gamasutra.com and Gamedev.net
16	Group Demo	Presentation Skills	

Table 2: *Instruction breakdown on a weekly basis for our course.*

Action items for the student	Our method	Top-down Model	Bottom-up Model	Commercial Engine
Create a working game	✓	✓		✓
Implement concepts by writing new code	✓	✓	✓	
Modify existing code	✓			
Working in teams	✓	maybe		✓
Mimic a game production pipeline	✓			✓

Table 3: *Comparison of different teaching methods.*

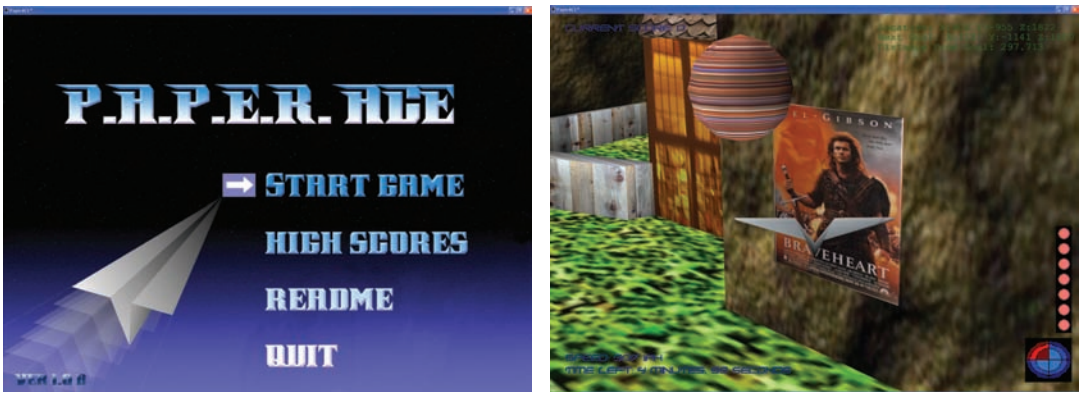

Figure 6: *Paper ACE: A physics based game which puts the user in the pilot's seat of a paper airplane. The game uses physical phenomena like wind, gravity etc. to present obstacles to the user.*

Figure 7: *Dodgeball: A character-animation centric game involving two three-player teams that play dodgeball. The user controls one player from a team, while the rest of players are driven by the AI engine. Explosion effects are modeled using sprites and particle systems.*

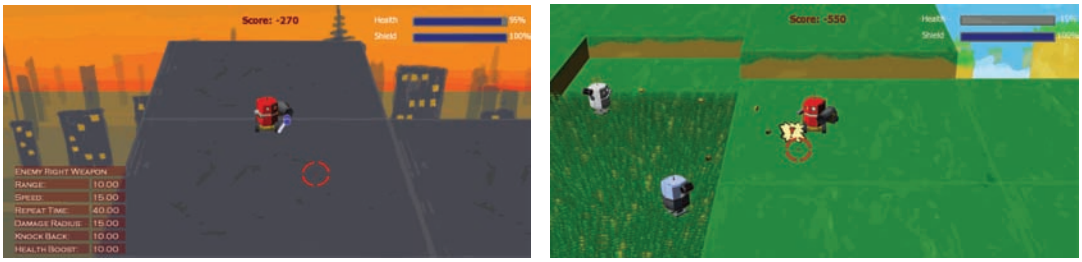

Figure 8: *J2Jazzment Day: A FPS game with heavy use of GLSL shaders for rendering the backdrop and the game characters.*