Teaching Introductory Computer Graphics Using Java 3D, Games and Customized Software: a Brazilian Experience

Romero Tori*
Universidade de São Paulo
Centro Universitário SENAC

João Luiz Bernardes Jr.†
Universidade de São Paulo
Centro Universitário SENAC

Ricardo Nakamura‡
Universidade de São Paulo

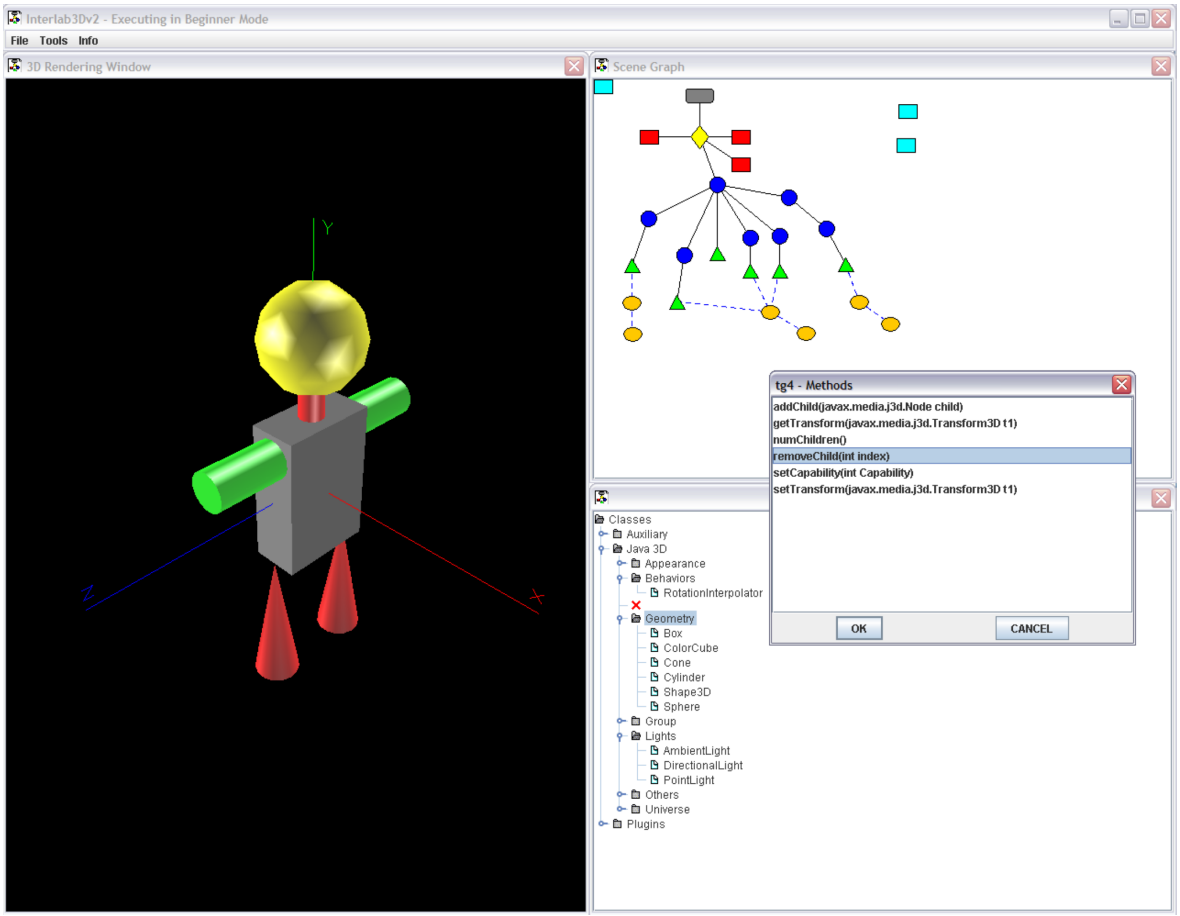

Figure 1: Interlab3D, a customized tool to support the learning of scene graph concepts and the Java 3D API

1 Introduction

This paper presents the authors' experience and results on organizing and teaching introductory Computer Graphics (CG) courses in Brazilian universities. The CG course described here uses a top-down approach, such as the one popularized by Angel [1997], where students manipulate 3D structures from the beginning in laboratory classes. Besides its top-down approach, this course adopts Java as its predominant programming language, Java 3D as the graphics API and the development of a 3D game as the course's main project. The laboratory classes are supported by software tools, two of them having been developed by the authors as customized and open-source applications: "Interlab3D" [Interlab 2005b], an interactive tool to learn and practice Java 3D and scene graph concepts, and "enJine" [Nakamura et. al. 2003b], a didactic game engine built over the Java 3D API.

The introductory CG Course described here requires a term project. Some years ago it was noticed that the number of projects dealing with game development issues were increasing with each course edition. Eventually, it was decided to fix the development of a simple 3D game as the project's theme. More recently a mandatory engine was also introduced, the specially developed Java 3D "enJine", so it became easier to support students, to control project level of difficulty, and to open the engine "black box" to the students. Using a game engine the students are able to build more complex and compelling games without needing to take their focus away from CG.

* email: tori@acm.org
† email: joao.bernardes@poli.usp.br
‡ email: ricardo.nakamura@poli.usp.br

This CG Course is not targeted at game development curricula (it is taught as an introductory CG Course to both computer science and computer engineering students). However, given market tendencies in the last decade, especially the growing availability of real-time 3D graphics hardware using local illumination models, as well as the marked interest students have shown in 3D games and their development, not only the content of laboratory classes, but also the theory presented has evolved to focus more heavily on real-time graphics, instead of realistic rendering. Electronic game technology and algorithms, particularly, have been having a growing influence on the course's conception, culminating in the term project. This approach has shown good results, demonstrated by positive feedback from the students, student performance and involvement, generally good course evaluations and excellent term projects developed in a relatively short time. The authors, therefore, believe that the approach presented here is an attractive alternative for teaching introductory CG in engineering and computer science courses as well as for the game development programs that are only recently being implemented in Brazil.

This paper is organized as follows: the first section consists of this introduction, while the second discusses related work. The third section describes the CG course in more detail, discussing both the adopted methodology and the course's structure. Next, the term project, consisting in the development of a 3D electronic game, is discussed. A description of the software tools used in the course, especially the customized ones, follows. The last two sections present final thoughts and conclusions.

2 Related Work

Many authors have discussed game technology applied to the learning process. Robertson and Good [2005] point out that a large percentage of children and young adults nowadays spends considerable time playing electronic games, a fact that cannot be left unnoticed by educators. Although it is not necessarily true that interest in playing games translates into interest in programming these applications, several authors such as Sweedyk & Keller [2005], Claypool and Claypool [2005] and Sweedyk et al. [2005] discuss how students can be motivated by game programming projects. Game programming is of great interest to students today, particularly for those attending computer science or computer engineering undergraduate programs. Recently, game programming has been used to teach, among others, basic programming skills [Moser 1997], software engineering [Sweedyk and Keller 2005], computer graphics [Hoetzlein and Schwartz 2005] and operating systems concepts [Hill et al. 2003]. Those works in some way support the decision of the authors of this paper in introducing game project, game engine and other related subjects into the computer graphics course design presented here.

Sung and Shirley [2003] have compared top-down and bottom-up approaches to teaching introductory computer graphics, and report limited success in an implemented top-down course. They also argue that top-down is a well-suited approach for mature students. In comparison with Angel [1997] methodology, those authors' course differs in the choice of a complex "popular graphic application", as the "top" software to be decomposed during course development, and in the focus on API independence. The approach described in section 3 of this paper is slightly more adherent to Angel's than to Sung and Shirley's method, as it starts with a simple 3D application and focuses on one specific API (Java 3D, in this case, instead of OpenGL as proposed by Angel). The authors of this paper believe that, as a top application, using a "popular graphic application" is less important than providing a fast learning curve, and that once a student has mastered the main concepts regarding API programming he can easily adapt to any other similar tool. So, the authors considered that for an introductory course didactic and usability aspects are more important than API independence.

Cunningham [2000] argues that "students experience a course very largely through the course projects" and points out the importance of shaping projects with real meaning for the students. He also defends a programming approach, using some graphics API, for a first computer graphics course. The decision for a game project development, using Java 3D API, as the core of the introductory course presented in this paper, is therefore in accordance with those arguments.

Coston and Edwards [2005] propose using "commercial-off-the-shelf 3D computer graphics software", like 3D Studio Max, in order to "enhance" computer graphics learning. The idea is to motivate beginning students to learn complex computer graphics concepts via "active learning". One of the benefits of that approach, besides learning by doing, is that students quickly grasp complex 3D graphics concepts and terminology. The course presented in this paper, in a similar way but in a smaller scale, implements that approach in the first part of its syllabus.

On the other hand Wolfe et al. [2000] argue that a "programming" approach using graphics API is not sufficient to cover all of the important concepts in an introductory course, such as global illumination. They also propose the development of customized didactic tools to help teach such concepts. But an introductory course usually has limited amount of time and some important subject will always be omitted. And a "programming" approach has, besides other advantages pointed out here and in other papers, the advantage of more recall provided by its "learning by doing" nature.

Some other related works are those from Nodelman [2003], exploring the link between theory and practice, Shultz [2006], proposing a "pedagogic 3D graphics API", and Wilkens [2001], defending a "multi-API" approach.

3 Course Methodology and Structure

This section will first discuss a few principles that guide the entire course and then present the actual course structure. The course was designed to require approximately 60 hours in class plus 40 hours of extra-class work. Most students are in the second half of a computer science or engineering course and have a good knowledge of programming, object orientation and software engineering. Most have no prior experience in computer graphics, however. Students need to develop a final graduation project and as this course takes place near the end of the graduation program, many choose to expand the practical project proposed in this course as their final project.

Given the relatively short course time in class, course design had to opt for certain subjects to focus on, to the detriment of others, as follows. The first option was to focus from the beginning on 3D computer graphics, instead of 2D. This decision was taken for several reasons, chiefly the growth and great availability of 3D

hardware and applications, students' marked interest in them, and the fact that it was judged simpler for the student to grasp high level modeling concepts before going into the technical details of the graphics pipeline and algorithms.

The course is taught with both lecture classes, to present the theory content, and laboratory classes where students have a hands-on contact with many of the concepts that will be seen in theory. All laboratory activities are developed in groups of 2 or 3 students. The laboratory and theory classes are alternated (students usually have both classes every week). Most concepts are manipulated first in the laboratory, with no more than an operational knowledge needed, and only later seen and discussed in the theory classes in more depth. In the authors' experience teaching such a course, it has been found that this approach aids in the comprehension and retention of theory concepts later, since the students already have a practical experience with the subject to fall back on during the exposition of the theory.

In the theory classes it is possible to give some overview of realistic rendering and low-level concepts, although these subjects due time constraints were not emphasized in the course design. In laboratory classes there is the need to specialize, focusing on one end of this spectrum. The higher-level end was chosen because it allows better visual results in less time, increasing student motivation and because in the Brazilian professional CG market those concepts are more frequently required. The impact of this decision on the laboratory classes can be seen in the following paragraphs. For a possible expansion of this course, the lower-level concepts could be worked within the practice classes as well and this is discussed at more length in section 6.

As discussed in the introduction and the section about related works, electronic games have influenced all aspects of the course, including theory classes, so that even within their generalist and introductory approach, real-time rendering is seen in more depth than realistic rendering. For the laboratory classes this is much more accentuated. Practical activities emphasize mostly game development related topics (involving CG, of course). The exception is ray tracing, explored in the first laboratory classes when using a modeler in an end-user context.

Both the laboratory and the theory classes attempt to follow a top-down scheme in the presentation of course content. Top-down, in this paper, is understood as starting with more general, higher-level concepts, which are closer to a user's experience and make use of several "lower-level concepts", and then gradually progressing through those lower-level concepts. The authors agree with Sung and Shirley [2003], who argue that this approach is better suited for teaching mature adults than going up from foundational algorithms such as rasterization to reach 3D images with lighting effects only by the end of the course.

Table 1 illustrates the course structure. Each row groups topics thematically and does not necessarily correspond to a single class, therefore this structure can be applied to courses with slightly different durations and adapted according to student background. There is a correspondence, however, between the columns describing theory and practice classes. The duration of practical activities described in each cell of the right column in Table 1 corresponds approximately to the duration of the presentation of the theory topics to the left of that cell. Also shown are the topics addressed in practice that will later be discussed in theory classes and the assignments given related to the project's development.

While the theory topics listed in Table 1 are self-explanatory, the practical activities require a better description. The first project-related activity is given even before the first laboratory class, during the first class in the course. The term project and its requirements are presented and the students have to prepare a game proposal, not quite as formal as a specification yet. These proposals are analyzed and, if necessary, modified by the teacher, as will be discussed in the next section. All games are required to have static as well as dynamic 3D elements.

<table>
<tr><th>Theory</th><th>Practice</th></tr>
<tr><td>Introduction to course and CG: origins, classifications, and devices.

Assignment: game proposal</td><td rowspan="2">Use of a 3D modeler

Concepts: Solid modeling, curves and surfaces, polygon mesh, colors, transparency, textures, light sources, camera, shading and ray-tracing

Assignment: Modeling the game's static components</td></tr>
<tr><td>Optics, human vision, color systems</td></tr>
<tr><td>Solid and Scene Representation and Modeling</td><td rowspan="3">Introduction to scene graphs

Concepts: Scene graphs, geometric transformations, animation

Assignment: Building the game's dynamic components (usually characters)</td></tr>
<tr><td>Graphics Pipeline</td></tr>
<tr><td>Local and global illumination; texture, light, bump and normal mapping</td></tr>
<tr><td>Geometric transformations, projection etc.</td><td rowspan="4">Game programming with an engine

Concepts: 2D overlay, camera manipulation, user interaction, polygon data structures

Assignment: Game completion</td></tr>
<tr><td>Curves and surfaces</td></tr>
<tr><td>Rasterization, anti-aliasing</td></tr>
<tr><td>Advanced topics (animation, VR, AR, visualization...)</td></tr>
</table>

Table 1: Course Structure

Then the students start working with a 3D modeling software [Eastman 2005] in the laboratory classes. Although the students have a programmer profile and are not 3D artists or designers, this activity is useful to give them an end-user's perception of many of the topics that will be explored in the course. While most students have had prior contact with 3D games, very few have used other 3D applications. Since the students already know what game they will develop by the time they have learned the basics of using the modeling software, they use the modeler to create the static 3D elements of the game. This is usually the game environment and character components. The assignment includes not only the creation of these elements but also their presentation as a static image with good visual impact. This forces students to tinker with optic properties, lighting and camera properties. The modeler used in the course allows configuration of several ray-tracing options and students are also encouraged to manipulate them to get the best results. This allows some practical contact with global illumination models that is not usually found in courses focusing on real-time rendering, such as those that use only OpenGL for practical assignments.

During the third project activity students use customized software that supports the learning of scene-graph concepts, particularly the Java 3D scene graph [INTERLAB 2005b]. In this stage they visually build a scene-graph that represents the dynamic elements of their game (usually the game's main character). This activity also demands intensive use of geometric transformations, since it is the only way to change an object's position in the scene. Finally, students build the code for their game using a didactic game engine [INTERLAB 2005a] that allows them to focus on CG concepts. Graphical response to user input and polygon data structures are some of the topics seen here.

Student evaluation is a continuous process. At the beginning of each theory class a quick quiz is applied. This not only reflects on student dedication but works as an important feedback to the teacher. Every laboratory class also has an assignment that is graded before the end of the next class. A final exam and the project's presentation are also part of the evaluation.

4 Game Development as a CG Term Project

The previous sections have discussed how today a large public spends long periods of time playing electronic games, and students of computer science or engineering, in the authors' experience, are generally part of this public. Related work showing how the use of such games can increase motivation in a course and how to use them has also been presented. Games are important applications in themselves, generating considerable revenue, helping propel the low-cost and real-time graphics hardware industry and being used in several applications besides entertainment, from scientific visualization of data to treatment of phobias. The CG course described here, however, did not have game development as a term project from its first version. This section will begin with a description of the project's evolution as a way to bring up several topics of interest.

In this course's first versions, the term project could be any "Interactive 3D application". While most students chose games, several other applications were presented, such as a facial expression simulator and a 3D chat environment. The projects varied greatly not only in theme but also in complexity and in which concepts were used most intensely in their implementation. Since projects were so different among themselves, it was also difficult to plan how to integrate the project in the laboratory classes. As a result, the students first had to complete assignments not related to the development of the project during half of the course and had only the second half to work on it.

Imposing 3D game development as the project's theme brought greater uniformity to this activity. This decision was supported by the prevalence of game development projects observed along the years. This decision simplified teacher interaction with the students during project specification to ensure less variation in complexity. It also simplified the choice of requirements to guarantee that all projects dealt with CG topics chosen by the teacher to complement the laboratory experiences. The games still varied greatly, however, and a few problems remained. First, students sometimes dedicated only the minimum required effort to CG and then spent considerable time and effort working with other subjects, such as artificial intelligence. With the variation among games it was still complicated to fit the project development and the other laboratory activities together. And students often opted to use different tools (such as engines for specific tasks) in the development, complicating project guidance and evaluation.

A radical imposition was then attempted. All projects would involve the development of the same game, designed by the teacher, with only a few variations possible in art or game rule details. While this allowed the integration of game development and practical activities from the beginning of the course (as will be exemplified in the next paragraphs), eliminated variations in complexity and allowed the use of uniform tools, it created two new problems. First, student motivation was somewhat lower and the great ideas that sometimes popped up in the projects with no restrictions could no longer be explored. Second, since all students faced almost exactly the same problems during a long term project, many were satisfied to copy the solutions from their colleagues, which happened much less often with a greater project variation.

These experiences illustrate the problem of how much to restrict the projects. More open assignments allow the students to focus on the area they are most interested in and may benefit most from [Kjelldahl 2005], especially with this course happening later in the graduation program. They also sometimes result in interesting, creative projects that positively surprise the teacher. On the other hand, this open approach considerably complicates the integration of the project with other activities, as well as project guidance and evaluation. Students also sometimes spend too much time deciding what to do when assignments have little restriction and are left with less time for implementation. If their proposal, in that case, is not completely accepted, this generates considerable frustration. Assignments that are too restricted, however, reduce motivation and stimulate cheating. In the current course version, a compromise solution is used.

Students are free to design any game they wish, but it must conform to a few rigid parameters. First, it must have a static 3D element, usually the environment, of some complexity and that has actual, not just esthetical, influence in the game. There must also be at least one element, usually the game's main character that is dynamic, meaning it is not a rigid body but has relative movement between its components. The game must not have a single fixed camera. The camera either follows the character somehow, or the player can choose between several cameras. While programming the game, the students must manipulate polygonal data structures directly. These last two parameters are used to help in the fixation of the related topics and could be discarded, replaced or complemented by similar requirements, depending on which concepts the teacher intends to work with in this stage of the project. Deviation from these parameters is only occasionally accepted if students intend to expand this work later as their final graduation project, and it must be strongly justified.

While rigid, these parameters allow a considerable variety of games to be developed. Environments can be forests, the rooms and furniture in a house, race tracks, a space scenario etc. Characters can be humanoids, animals or robots with moving parts, cars with moving wheels and steering wheels, helicopters etc. Manipulating polygonal data structures is usually accomplished through procedural generation of some environment elements. With these parameters, the resulting game projects are still different enough so that students feel motivated by feeling they "own" the idea and simply copying solutions between projects becomes more complicated and not as compelling. The parameters reduce the time students spend to create a proposal, allow the teacher to direct the project to work with desired concepts and allow the project to be better integrated with course activities, as shown in Table 1 and explained in more detail in the following paragraphs.

The first step in the project is presenting the requirements to the students, letting them choose their groups and demanding a game proposal as soon as possible. While this proposal does not need to be very formal, it must clearly state what the students intend to do and what is the project's scope. In the author's experience, most students do not get project complexity right in this first proposal. Many want to make, in a few months, a game that would take a professional studio a couple years to finish. Some want to do as little as they think is possible. This is the reason why these proposals are delivered quickly. The teacher analyzes each of them with the students and proposes changes when necessary, remaining as faithful to the original idea as possible, so the complexity of the projects is more uniform and they are feasible but still challenging. This analysis is also useful to see how well each proposal conforms to the requirements. Once the proposal has been modified and accepted, students start working on a game specification. While the teacher does not immediately analyze this specification, it is useful for the students. To prepare the specification they sketch the game's architecture and estimate initial values for the games numerical parameters, such as how an action should be scored. These activities usually take up a lot of time if they are left to be done only after implementation starts.

After the proposal is accepted, even before the specification is finished, game implementation is initiated and integrated into the course's practical activities as show in Table 1. Using the modeling software, after completing its tutorial, they start building game environment and other static elements. Then they visually build a scene graph, using a didactic software tool designed for that, to represent the game's character and other dynamic elements. Finally, using a didactic engine and the results from these previous assignments, they finish building the game.

Interaction between different groups is encouraged up to a certain point. While discussion of problems and solutions is healthy, when these solutions are simply copied from another project it becomes a problem. This is usually minimized, however, by the variations between proposed games and software architectures. A certain competition mood usually develops in laboratory classes with a few groups of students trying to develop the "coolest" game. This competition, however, has always been healthy, because students know there is little advantage in "winning" aside from personal satisfaction, since all the "contenders" usually end up developing works that are more than good enough to get the maximum grade in the project evaluation.

Students must deliver a project presentation by the end of the course. Figure 2 shows some students term projects screenshots. Project evaluation is based on the resulting software, this presentation and project documentation. This documentation includes the specification elaborated at the beginning of the course and a report detailing project development: how and why it deviated from the original specification, what were the major technical issues in the project, which solutions were chosen and a general discussion of the project. This documentation is not evaluated as rigidly as it would be in a software engineering course, but since most students have already acquired that knowledge, they tend to put it to good use anyway. While originality and good playability in the developed game is rewarded, one thing that is not usually strongly considered in the evaluation is artistic achievement. As said earlier, most students are not of an artistic profile and evaluating them in that aspect usually results in frustration.

5 Software Tools

As previously discussed, the course involves theory classes and laboratory activities. Referring back to Table 1, the tools adopted for the laboratory activities are: Art Of Illusion, a third party, free 3D modeler written in Java, Interlab3D, a custom tool to learn scene graph concepts and the Java 3D API, and enJine, a game engine also developed at INTERLAB/USP. This section presents details about each of them.

Art Of Illusion is a 3D modeling package available under the GNU General Public License [Eastman 2005]. The availability as Free Software was one of our selection criteria for tools used in the course, to allow students unlimited access. Art Of Illusion has features comparable to commercial products, including tools for complex object modeling and a ray-tracer with advanced capabilities. In the course, this tool is used for a practical introduction to computer graphics, allowing students to have some experience with concepts that will be discussed in later classes. It is also employed as a modeling tool for the game project.

Java is the programming language currently adopted for the practical activities in the course, along with Java 3D, its 3D graphics API, which is based on a scene graph model. The choice of a higher level API such as Java 3D is coherent with the course methodology and goals, as well as with considerations presented by Cunningham [2000] regarding introductory Computer Graphics courses. This decision has in turn influenced the creation and development of both Interlab3D and enJine.

Interlab3D was created at Interlab/USP as a tool to help students learn the Java 3D API. It was designed to expose the API classes and methods through an interactive interface, to avoid the overhead of writing initialization blocks and learning the details of the API in the beginning. It is available under the GNU General Public License. [INTERLAB 2005b].

Figure 1 shows the interface of Interlab3D, which is split into four independent windows whose size and layout can be freely reorganized. The top window contains the application title and menu. The large window on the left presents the 3D scene as rendered by the Java 3D API. The two windows on the right are the scene graph view and the class list.

The class list window presents the Java classes available to the student. Most of these are Java 3D classes, but the window can also include helper classes and extensions. When students select one of these classes from the list, they are presented with a list of constructors to create a new instance of that class. They must assign a valid Java identifier to the instance, which is then added to the scene graph view. The scene graph view contains icons that represent all the objects that have been created by the student. By selecting an object in this window, it is possible to call any of its public methods. In this way, students can interactively manipulate the objects and create 3D scenes. Later, students can generate Java source code from the scene they have edited.

The current version of Interlab3D allows its extension through the addition of new classes that are made available to the students. This is done through configuration files that may be manually edited or automatically generated by an existing helper application, without the need to recompile or generate new versions of the software. With this feature it is possible, for instance, to create higher-level abstractions that hide the Java 3D specifics to teach only API-independent scene graph concepts.

When the user executes Interlab3D, he must choose between two operating modes: beginner or expert. In the beginner mode, many helper classes are available, Java 3D capabilities are automatically enabled and the scene graph is already set up with the essential objects for a basic scene. In expert mode, only the Java 3D classes are made available, but the user can generate Java source code from the scene graph. This separation of features is meant to follow the transition of laboratory activities, from interactive object manipulation to direct programming.

The enJine software [Nakamura et al. 2003b] was initially developed as a didactic game engine, based on the results of research by Nakamura et al. [2003a] related to the use of commercial game engines in an educational context. As the other tools adopted for the course, it is available under the GNU General Public License [INTERLAB 2005a].

EnJine consists of a set of Java classes that can be used to represent the elements of a computer game. It also has packages to provide 3D graphics rendering, sound playing, input handling and collision testing. Its specification also includes support for networked games. From its origin as a tool for educational games, enJine was designed to better support action/adventure games, although it is possible to use it for other game styles.

Given the availability of many game engines in the Internet, it is important to understand the reasons for adopting this custom-made tool in the course. First of all, enJine was developed as a didactic tool, so many of its design choices favor source code clarity and correctness over extreme optimization. As the course uses Java, it was desirable to have a game engine in the same programming language. However, when the development of enJine began, the authors did not find any Java game engine available as free software.

The current version of the enJine is the result of a revision of the overall architecture of the project. In this revision, the initial specification of the engine was adapted to better suit its use in the context of a computer graphics course. The implementation of this version was focused initially on providing a stable core for single player games. For this reason, this version does not have support for networked games yet. The main features of enJine are:

- A set of Java classes to allow flexible representation of game elements such as characters, objects and scenarios;
- Basic infrastructure for a game application, including the capability for complex interactions between game objects through a message exchange mechanism;
- 3D graphics rendering through the Java 3D API, so that students can apply their experience in previous laboratory activities;
- Support for basic sound output and user input (currently only through keyboard, but extensible to other devices).

Future versions of enJine will contain improvements such as networked game support and other options for collision detection algorithms.

Since students have access to the entire source code of enJine, it is expected that they modify or extend it to suit their projects, and even contribute to the improvement of that tool.

6 Further Considerations

The methodology, experience and tools described in this paper are mostly the result of a recent effort, started approximately 6 years ago, in reformulating a traditional introductory CG course that has been evolved by one of the authors, in cooperation with other colleagues, since 1985. This reformulation effort involved at least three teachers, three different institutions, ten instances of the course, and literally hundreds of students with varying backgrounds. Each time the course was taught it was also modified based on student performance and feedback, technological changes and discussions among the teachers involved. Formal evaluations with the students were not performed in every course instance. In the past few courses, the number of these modifications has continuously fallen, leading the authors to believe that the methodology is converging to the one best suited to the needs of the teachers involved.

The matter of how much to restrict the term project is one example of a compromise solution reached during course evolution, that seems to combine the advantages of an open and a restricted project as well as possible, as discussed in section 4.

One issue that is still not considered well resolved by the authors is the current prevalence of high level and real-time rendering subjects, particularly in practical classes. Future work in this methodology involves a course expansion, possibly including programming at a lower level, for instance to directly manipulate the frame buffer and shaders. Yet another possibility is the development of a ray-tracer by the students as an assignment. Given the limited course time and the fact that the reasons to choose the current focus are chiefly student motivation and local market requirements, this expansion is tied to an expansion in course duration as well or the creation of an advanced optional course. The authors are also working in an open and distance education version of this course.

One final point is that, no matter how much focus is given to CG, which is of interest to this course, game development always touches on several other major areas of computer science (Ferreira, 2002), which leads to the question of whether a better alternative would be to use a game development project in practical classes not only for CG courses, but also in a project integrating several other courses, such as human-computer interface, networking and software engineering.

7 Conclusion

Computer graphics challenges teachers every time they are offering a course on the subject. Nowadays most of the computer science or engineering students enter College already having some familiarity with graphics and game skills, be it as a casual or even as experienced users. The increasing pervasiveness of computer-generated images and the “special effects overload” that has been seen in most societies have contributed to create misconceptions and false expectations. Another issue refers to the amount of mathematical, programming and multidisciplinary skills required for studying computer graphics, which could cause frustration to students expecting a more glamorous course. So, motivation, renovation, didactics and “real meaning learning” should be some of the issues an introductory computer graphics teacher should be worrying about. The large, although far from complete, roll of related works presented in this paper is an evidence of the abundance of discussion, ideas and proposals trying to address those issues.

In this paper the authors tried to bring some new experiences and results to this discussion, by presenting syllabus, methodology and tools used in an introductory computer graphics course for computer science and engineering programs. Additionally this paper has presented two customized didactic software developed by the authors for this course but that can be applied in other pedagogical contexts.

Acknowledgements

The authors would like to thank Escola Politécnica da Universidade de São Paulo and Centro Universitário Senac for their support, Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), for supporting undergraduate researchers via its PIBIC Program, Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), for hosting the projects "Interlab 3D" and "enJine" in its "Incubadora Virtual de Conteúdos". The authors would also like to thank all their former students and the researchers from Interactive Technology Laboratory (Interlab) that directly or indirectly participated in the development of "Interlab 3D" and "enJine" software tools.

References

ANGEL, E. 1997. *Interactive Computer Graphics: A Top-Down Approach Using OpenGL*. Addison Wesley.

CLAYPOOL, K. and Claypool, M. 2005. Teaching Software Engineering through Game Design. In *Proceedings of the Innovation and Technology in Computer Science Education (ITiCSE) Conference 2005*, ACM Press, New York, 123-127.

COSTON, J., and EDWARDS, A. 2005. Actively Learning Computer Graphics, *Journal of Computing Sciences in Colleges 20*, 4, 221-226.

CUNNINGHAM, S. 2000. Powers of 10: the Case for Changing the First Course in Computer Graphics. In *Proceedings of the Innovation and Technology in Computer Science Education (ITiCSE) Conference 2000*, ACM Press, New York. 46-49.

EASTMAN, P. 2005. *Art Of Illusion*. http://www.artofillusion.org

FERREIRA, A. P. L.; TORI, R.; BATTAIOLA, A. L.; and ELIAS, N. 2002. Game Technology as an Educational Tool. In *Informatics Curricula, Teaching Methods and Best Practice-IFIP Working Group 3.2 Informatics and ICT in Higher Education ICTEM 2002*, IFIP, 165-172.

HILL, J. et al. 2003. Puzzles and games: addressing different learning styles in teaching operating systems concepts. In In *Proceedings of the Innovation and Technology in Computer Science Education (ITiCSE) Conference 2003*, ACM Press, New York, 182-186.

HOETZLEIN, R., and SCHWARTZ, D. 2005. GameX: A Platform for Incremental Instruction in Computer Graphics and Game Design. In *Proceedings of ACM SIGGRAPH 2005 Educator´s Program*, ACM Press, New York.

INTERLAB. 2005a. *EnJine: Java Multiuser Game Engine*. http://enjine.incubadora.fapesp.br

INTERLAB. 2005b. *Interlab3D*. http://interlab3d.incubadora.fapesp.br

KJELLDAHL, L. 2005. Deepening Assignments. In *Proceedings of ACM SIGGRAPH 2005 Educator´s Program*, ACM Press, New York.

MOSER, R. 1997. A Fantasy Adventure Game as a Learning Environment, In *Proceedings of the Innovation and Technology in Computer Science Education (ITiCSE) Conference 2005*, ACM Press, New York, 114-116.

MUKUNDAN, R. 1999. Teaching Computer Graphics Using Java. In *Working group reports from ITiCSE on Innovation and technology in computer science education*, ACM Press, New York. 66-69.

NAKAMURA, R. et al. 2003a. A Practical Study on the Usage of a Commercial Game Engine for the development of Educational Games. In *Proceedings of 2nd Games and Digital Entertainment Workshop*, Brazilian Computer Society.

NAKAMURA, R. et al. 2003b. Development of a Game Engine Using Java. In *Proceedings of 2nd Games and Digital Entertainment Workshop*, Brazilian Computer Society.

NODELMAN, V. 2003. Learning Computer Graphics by Programming: Linking Theory and Practice. In *Proceedings of the Innovation and Technology in Computer Science Education (ITiCSE) Conference 2003*, ACM Press, New York, 261-261.

ROBERTSON, J. and GOOD, J. 2005. Story Creation in Virtual Game Worlds, *Communications of the ACM, 48*, 1, 61-65.

SHULTZ, G. 2006. Integrating 3D Graphics into Early CS Courses, *Journal of Computing Sciences in Colleges 21*, 3, 169-178.

SUNG, K. and SHIRLEY, P. 2003. A top-down approach to teaching introductory computer graphics. In *Proceedings of ACM SIGGRAPH 2003Educator´s Program*, ACM Press, New York., 1-4.

SWEEDYK, E. and KELLER, R. 2005. Fun and Games: A New Software Engineering Course. In *Proceedings of the Innovation and Technology in Computer Science Education (ITiCSE) Conference 2005*, ACM Press, New York, 138-142.

SWEEDYK, E., DE LAET, M., SLATTERY, M. C., and Kuffner, J. 2005. Computer Games and CS Education: Why and How. In *Proceedings of the Innovation and Technology in Computer Science Education (ITiCSE) Conference 2005*, ACM Press, New York, 256-257.

WILKENS, L. 2001. A multi-API Course in Computer Graphics. In *Proceedings of the Sixth Annual CCSC Northeastern Conference on the Journal of Computing in Small Colleges,* Consortium for Computing Sciences in Colleges, USA, 66-73.

WOLFE, R., LOWTHER, J. L., and SHENE, C. 2000. Rendering + Modeling + Animation + Postprocessing = Computer Graphics, *ACM SIGGRAPH Computer Graphics 34*, 4, 15-18.

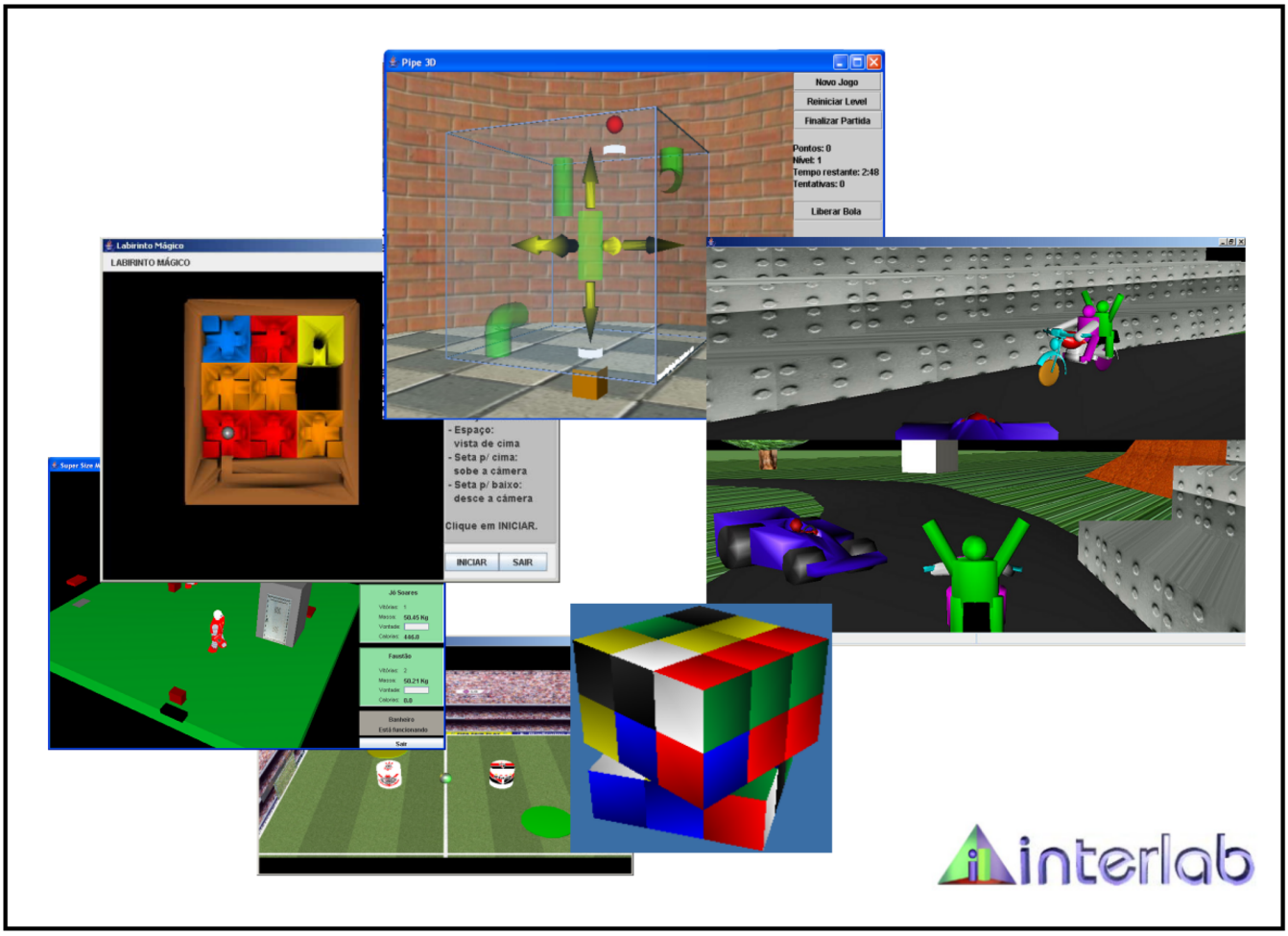

Figure 2 – Screenshots of some students´ projects