By students, for students: a production-quality multimedia library and its application to game-based teaching

Till Bay
Chair of Software Engineering
ETH Zurich
8092 Zurich, Switzerland
+41 44 632 76 33
till.bay@inf.ethz.ch

Michela Pedroni
Chair of Software Engineering
ETH Zurich
8092 Zurich, Switzerland
+41 44 632 76 84
michela.pedroni@inf.ethz.ch

Bertrand Meyer
Chair of Software Engineering
ETH Zurich
8092 Zurich, Switzerland
+41 44 632 04 10
bertrand.meyer@inf.ethz.ch

ABSTRACT

The attractive idea of using game development for teaching programming can only meet student expectations and modern software engineering requirements if it uses advanced multimedia technology, at the level of the best commercial solutions. In implementing novel pedagogical techniques, we have developed a powerful multimedia library, with major contributions from students, and used it to offer games as course projects. More than 150 games have been developed, many of very high quality, and publicly available for everyone's enjoyment. This experience combines advanced software development with student participation, strong O-O software engineering principles, and the excitement of one of the coolest areas of technology.

Categories and Subject Descriptors

K.3.2 [Computers and Education]: Computer and Information Science Education – *curriculum*. D.2.2, D.2.13 [Software Engineering]: Design tools and techniques, Reusable software. H.5.4, H.5.5 [Information Systems]: Hypertext/Hypermedia, Sound and music computing — *systems*.

General Terms

Theory, Software engineering.

Keywords

Curriculum design, knowledge modeling, course comparison. object-oriented programming, game programming.

1. GAMES IN THE CURRICULUM

Today's students are increasingly game-savvy; computer games are among the top three positive factors that make them consider computer science as a major [4]. CS education should not only acknowledge this phenomenon but capitalize on it. This can only succeed if we meet the quality expectations of students used to increasingly powerful graphics and multimedia.

Starting with the very first CS course, introductory programming, our teaching fundamentally relies on games, especially game projects, to introduce software concepts and let students apply them to large examples. As a key component of this strategy we have developed a state-of-the-art multimedia library, EiffelMedia [3]. Started in 2004 as a single student project to support the city simulation software Traffic used in our introductory course ([15] and fig. 1), the library has progressively expanded into a major software project, now reaching 500,00 lines of code, but still maintained by students. We use it extensively in our courses, both introductory and intermediary. We feel that the library fills the needs of game programming in an educational environment, a claim backed by more than 150 student-implemented games to date, some of industrial grade and all publicly available (like EiffelMedia itself) in both source and binary.

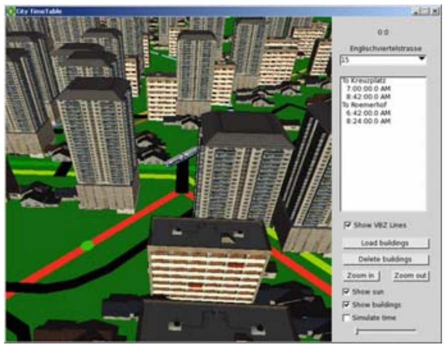


Figure 1: City3D Traffic example application

Section 2 is an overview of the library. Section 3 explains how we use it for teaching. Section 4 presents a few student games. Section 5 describes our open-source development process involving students. Section 6 presents an evaluation and outlook.

2. KEY FRAMEWORKS

EiffelMedia consists of a number of frameworks, some of which will now be presented. Each description provides an overview and a screenshot of an example application, making it clear in which part of game programming the framework can help.

2.1 2D graphics

The core framework is for displaying two-dimensional image data. EiffelMedia supports most image file formats and has a pixel manipulation API, as well as built-in alpha channels that allow adjusting opacity or even making portions of the screen transparent. Loading images alone is not enough for gaming; the framework also supports displaying so-called *sprites*: time-based sequences of flip-through images used for example to animate the movement of a player within a game.

To handle and display text, font support is available. This includes not only bitmap fonts, which require generating one bitmap image per font size (fig. 2) but also TrueType fonts, which remove this limitation and support seamless scaling, anti-aliasing and transparency.

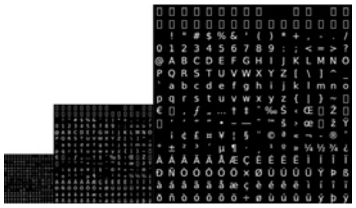


Figure 2: Bitmapfont bitmaps for three different font sizes

A collection of geometric figures including polygons, circles and lines completes the 2D API. It is possible to perform drawing operations without loading the data from image files. All rendering of geometric figures is anti-aliased. The object model for the graphical artifacts allows combining them in a hierarchical way and manipulating entire sets at one time. The standard viewport into which EiffelMedia renders is an entire program window, but it is possible to switch to full screen or render into a widget of EiffelVision [5], the platform-independent widget toolkit of the Eiffel [6] platform. To partition the application into different logical parts, EiffelMedia provides scenes. In the context of games, these scenes can be understood as levels.

Soon after the first jump'n'run games had been developed, users of the library asked for a physics engine that would provide a general solution to the problem of applying forces to the movement of graphical objects. The resulting framework goes further than this goal; since different games may require different kinds of physics, it actually provides a scriptable physics engine, which can be configured for different force directions, different force fields and varying forces. The project that contributed this engine also showed how difficult it was to generalize game developers' needs; as a result, physics manipulation for the 3D part of the library is left to individual developers. The project leaders propose a sample 3D physics engine as inspiration, but no generic implementation as for the 2D case.

Figure 3: "HelloWorld" example application

Fig. 3 shows the small sample application of the 2D framework: a simple Hello world consisting of a displayed bitmap image. Figure 4 is from an application that demonstrates the use of the 2D drawing primitives and a zooming viewport.

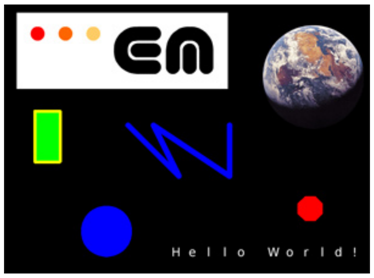


Figure 4: Drawable example application

2.2 Sound

An image is worth a thousand words, but in gaming the mood of the game is also conveyed by the soundtrack. Sound was only added in the fourth version, more than a year after the first public release. Until then, it had been the most urgently requested addition on the forums. It is interesting to see how much more fun a game with good sound is.

Game developers can choose from a wide range of possibilities to add sound. As with image formats, the library supports the most popular binary audio file formats. The API for playback supports mixing different tracks, adjusting sound volume, and grabbing sound data for use as input for visual sound renderings. A mini-framework is also available to manage sound filters. As with the 2D-physics engine, a generalized framework with a simple interface is available for the more common sound needs of a game (playback of a background tune with short mixed-in sound effects underlining actions happening in the game).

A small and easily understandable example shows the basic use of the sound API; its code is just

```
if Audio_subsystem.is_enabled then
                -- Open mixer with frequency 44.1 kHz
    Audio_subsystem.mixer.open
                -- Initialize with one single file
    create player.make_with_file ("wish_you_were_here.ogg")
    player.set_repeat (true)
    player.play
end
```

A full application — an Audio Mixer, fig. 5 — is also available, demonstrating the full range of possibilities.

Figure 5: Audio Mixer example application

2.3 Collision detection

Collisions are one of the primary sources of interactions between game objects. For collision detection it soon became clear that it's not enough to use bounding boxes based on outer edges of colliding objects. In the *TinyToys* student game (fig. 7), for example, cars race each other on a track that contains obstacles which cannot possibly be approximated by bounding boxes.

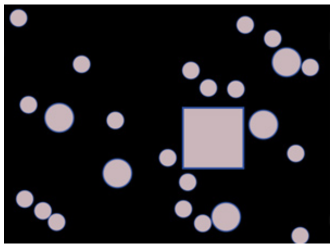


Figure 6: Collision detection example application

Figure 7: TinyToys, a racing game

The immediate enhancement was to switch to polygon-based collision detection. It still requires manual definition of the polygons for all cases involving concave polygons, but this is supported by the polygon editor illustrated in figure 8.

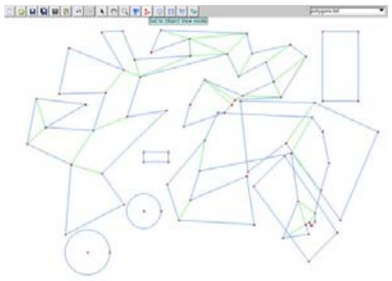


Figure 8: Polygon Editor

2.4 3D graphics

With Direct3D [9] and OpenGL [14], the industry provides powerful back-ends for rendering three-dimensional worlds. EiffelMedia is platform-independent and therefore builds its 3D support on OpenGL. The EiffelMedia 3D framework supports loading 3D polygon mesh models from different vendors, and texturing and lighting these models. At the moment, collision detection in 3D is managed using intersecting spheres; this appears sufficient.

Manipulating objects in 3D-space requires suitable and ready-to-use data structures, such as vectors and quaternions, all with implementations of cameras and rendering pipes that handle the fast display of complex structures. The library provides a number of 3D primitives including spheres, boxes, triangles and cylinders, complemented by such advanced features as procedural textures and support for the OpenGL Shading Language (GLSL) [1].

Using the object-oriented 3D framework has enabled student projects to implement games such as *Conquer* (figure 9) or *AntWorld* (figure 10), which although all each implemented in a few weeks reach a level of 3D quality that withstands the comparison with commercial games.

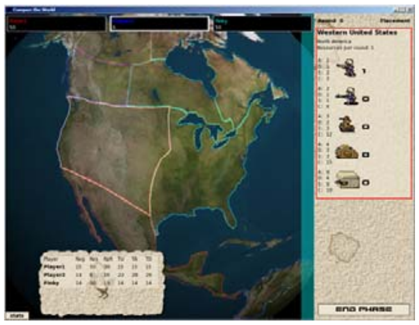


Figure 9: Conquer, a world conquering game similar to Risk

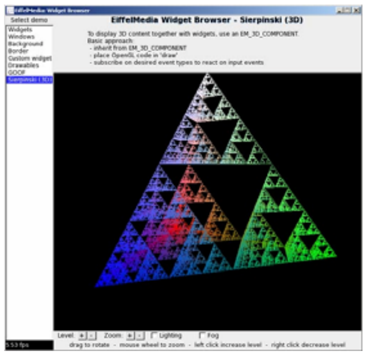
Figure 10: Day and night in AntWorld

2.5 Networking and high-score tracking

Just like sound, the ability to interact with other players adds a key dimension to today's games. EiffelMedia's networking framework illustrates our view of a ready-to-use infrastructure, letting students and other game authors go in little time from *localhost* to the full World-Wide Web. Starting with TCP and UDP support, it moves on to sample implementations of HTTP and other protocols useful in network gaming. Again the API is simple and contains a generalized multiplayer example.

EiffelMedia adds to this a high-score-publishing framework, to share the outcome of gaming sessions. One of the requirements was to be general enough to support various game modes. Ranking in games can vary from very simple lists, where points are listed in decreasing order, to games where players can form clans or teams; the framework supports this full spectrum.

2.6 Widget toolkit



EiffelMedia Widget Browser - Sierpinski (3D)

To display 3D content together with widgets, use an EM_3D_COMPONENT.
Basic approach:
- inherit from EM_3D_COMPONENT
- place OpenGL code in 'draw'
- subscribe on desired event types to react on input events

drag to rotate - mouse wheel to zoom - left click increase level - right click decrease level

Figure 11: Widget Browser example application, showing 3D Sierpinsky's pyramid in OpenGL widget

One of the reasons for EiffelMedia's feature richness is that the development process allowed immediate integration of user feedback. As an example, when the creators of *Conquer* (figure 9) encountered the need for buttons and other widgets, they implemented what is now the Widget Toolkit, themable and well integrated with the rest of the library. The API is inspired by existing toolkits such as EiffelVision2 [5] and native ones such as Cocoa [2] , GTk [8] and GDI [10].

The toolkit includes a large set of themable widgets—everything from the radio button to the split-screen widget; some elements such as a Rich-Text widget, not needed so far, remain to be added. Fig. 11 shows the widget example application with a navigation panel where different widgets can be investigated. The selected widget illustrates the component for 3D object rendering.

2.7 Other features

Game programming requires many other facilities, which are available in EiffelMedia but cannot be fully described in the scope of this article. We'll list only a few. EiffelMedia supports user interaction through a wide variety of input devices: keyboard, mouse, joystick, CD-ROM, DVD. It includes a framework for serializing object structures (e.g. for save-games); a level management mechanism; a resource-loading framework; an error reporting facility; video playback for intro and outro game sequences.

3. USES IN THE CLASSROOM

Our use of games is part of a comprehensive approach to teaching introductory programming, based on novel concepts of “inverted curriculum” and “outside-in” strategies and supported by a textbook in progress [12]. The approach emphasizes strong software engineering principles, including some formal aspects, to prepare our students for the IT world of the future; but it is also deeply rooted in the practice of software development and gives a key role to active student participation. One of its characteristics is that it presents students, right from the start of the first-semester introductory course, with considerable amounts of software, both existing libraries and some, like Traffic, developed specifically for this purpose. This pursues several objectives: providing large amounts of inspiration (since programming is learned in part by imitation anyway, better offer *good* software for imitation); showing the benefits of abstraction, information hiding and Design by Contract; enabling students to develop their own exciting applications.

In this last goal the results have exceeded all our expectations. We started asking students to produce games as final project for the 2nd-year Software Architecture course in 2005; the results were so impressive that we gave a similar assignment for the *first-semester* introductory programming course the next year. In groups of 2 to 4, students produced games, in each case over a period of only 6 to 8 weeks at the end of the semester.

We encourage students to choose partners of equal proficiency and let them freely choose the game to produce[1]. This creates a motivational boost and results in a mastery effect that proves to be highly beneficial for the learning process. We chose to trust the creativity and good sense of the students, and experience showed this to have been the right decision. Many expended considerable effort, far more than required; in fact, because of the specific rules at ETH (where student assessment for the first two years is based solely on an end-of-year exam), the projects are not even graded.

We have been repeatedly amazed by the seriousness of the work, including (as we of course request) design documents and user documentation. Along with the fun, students understood that these were serious software engineering endeavors. Some projects are professionally delivered, with such trappings as an installation (.msi) package. In the end, more than 150 different running games have been developed over these two courses; several dozen are available for public enjoyment at games.ethz.ch [7].

Particularly gratifying is that surprisingly good games were created not only by the whiz-kids but also by average students. We believe that the attraction of games combined with relentless emphasis of the best software engineering principles is one of the best teaching vehicles, and profits both the students who were programming-savvy before their studies and those who are new to the field but strong in other areas, for example mathematics.

4. EXAMPLE GAMES

The following are three examples of student-created games, available from the site.

4.1 X Adventure Engine

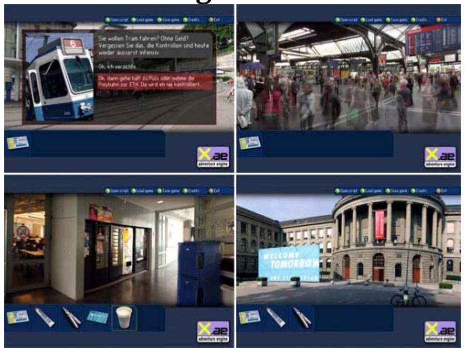


Figure 12: X Adventure Engine

X Adventure Engine (XAE) was the first game to blew everybody's mind, in the first “Software Architecture” project. The four students wrote not just a game but a *game engine* that can load adventure games described in XML. The viewer application loads the game files and renders panorama images on a cylindrical stage where the user navigates, picks up objects and talks to people. It supports animation, as well as loading and saving of playing status; the credits section shows amazing 3D animations. XAE comes with an example game entitled “*Tomorrow is another day*”. The game's goal game is to decode an encrypted USB stick to find an algorithm that converts frequent-customer points from a well-known Zurich supermarket into credit points for your studies. In the process you may query fellow students (from that year) as well as some ETH professors, appearing in effigy throughout the game, for help according to their expertise, e.g. cryptographic algorithms. There are many more creative ideas, all invented (as well as the story) by the students, who created all the images and convinced students and professors to appear in the game. Playing the game through takes approximately an hour.

[1] As ETH was celebrating its 150th anniversary in 2005, the rule, broad and not strictly enforced, was that games have some connection to ETH, Switzerland, the year 1855 or the number 150.

4.2 Ebouncy

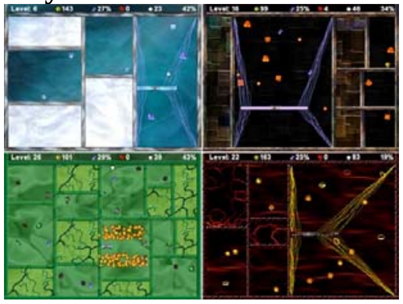


Figure 13: Different levels in Ebouncy

Ebouncy has been responsible for many students spending time gaming rather than studying. The plot is simple: a laser splits the game area into two parts, with hostile objects bouncing around to make this difficult. The higher the (unlimited) level, the more hostile objects. Power-ups, extra-lifes and other candy can make life easier. Enhancing the game are a cool soundtrack and the connection to a high-score server. The game is visually very attractive, with considerable attention devoted to graphics effects.

4.3 AntWorld

AntWorld (fig. 10), based on the latest version of the library, features the most advanced 3D graphics and never fails to impress with its graphic sophistication. The game idea is simple: the gamer controls an ant population that tries to spread on a planet. The students came up with mouse-driven 3D navigation and implemented different perspectives for looking at the world.

5. DEVELOPMENT PROCESS

The library is maintained as an open-source project with the help of numerous students. The most important factor in its development has been students first using it for their game projects, then contributing. They quickly noticed whatever was missing for their games; after the courses, many came back and naturally started to add the missing elements. This has led to a fast and effective development process. The result is feature richness—mini-frameworks have come to life that the project owners would never have imagined—and the reconciliation of different visions of what the ideal multimedia library should be.

Such a process also has risks. The software grows organically and may veer into directions that need to be abandoned later. The key to success has been to enforce thoroughly on everyone involved—students and assistants— a small set of critical design principles.

5.1 Design principles

Not only does EiffelMedia provide a completely object-oriented multimedia API, with extensive use of genericity, polymorphism and dynamic binding, it also relies on a strong O-O approach for its own design, including Eiffel-specific techniques, notably Design by Contract [11]. This yield a library that is easy to understand and to maintain. Another principle is that every part of the API must provide, from the start, a very simple example showcasing its use. Library and examples evolve together: a student who improves the audio API must update all the related examples. We encourage every team also to provide a larger application—such as the mixer of fig. 5 for the sound API—demonstrating the full power of their API. We think that this idea of a small and a big demo to drive the process is of interest as a general principle of library design.

Another principle (initially neglected, which caused considerable later work) is: no use of any copyrighted material. It's by imposing this rule strictly, not only on the library but on all applications built with it, and by checking its observance, that we were able to publish all the projects on the Web [7].

5.2 Metrics

EiffelMedia and the EiffelMedia community have grown substantially since the library was started two years ago. Some of the relevant figures are: one permanent (but part-time) developer (the first author, who was the original student behind the first version); 30 developers in total over two years of development; 2000 posts on mailing list; 12,300 CVS check-ins; 1350 classes; 500,000 lines of code; more than 150 applications.

5.3 Awards

While having teams of students develop the next version of the library, we participated in two iterations of a programming contest, the Eiffel Struggle [13], organized by an independent consortium; EiffelMedia won the 15th price in 2004 and 2nd price in 2005. The prize money is used for release parties. This contest has been a strong motivational factor for the students..

6. OUTLOOK

The EiffelMedia experience shows that it is possible to develop production-grade software in a university context. It also demonstrates that it pays to give good tools to students and encourage their creativity, and that coolness can go very well with teaching the sternest software engineering principles.

Fostering contributions by students who formerly experienced the library as clients has boosted the development of EiffelMedia to a full-fledged multimedia library. This idea of letting everybody suggest new functionality and add to future versions can work in other programming projects, and not just for education. Whether for students or for professional programmers, it's hard to think of a better motivation.

7. REFERENCES

[1] 3Dlabs GLSL, at developer.3dlabs.com.

[2] Apple Cocoa, at developer.apple.com/documentation/Cocoa.

[3] Till Bay: *EiffelMedia 0.9.0*, at eiffelmedia.origo.ethz.ch.

[4] L. Carter: *Why students with an apparent aptitude for computer science don't choose to major in computer science*, in SIGCSE '06, New York, ACM Press, 2006, pp. 27–31.

[5] Eiffel Software, *EiffelVision 2 library*, at www.eiffel.com/libraries/vision2.html.

[6] Eiffel Software, www.eiffel.com.

[7] ETH Zurich, various students: *EiffelMedia Games*, numerous games, 2003-2006, at games.ethz.ch.

[8] Gtk, at www.gtk.org.

[9] Microsoft Direct 3D, www.microsoft.com/windows/directx.

[10] Microsoft GDI, at msdn.microsoft.com/library/default.asp?url=/library/en-us/gdicpp/GDIPlus/aboutGDIPlus.asp.

[11] Bertrand Meyer: *Object-Oriented Software Construction*, 2nd edition, Prentice Hall, 1997.

[12] Bertrand Meyer: *Touch of Class: Learning how to Program Well Using Object Technology and Design by Contract*, to appear; draft and supporting material at se.ethz.ch/touch.

[13] NICE Consortium: *Eiffel Struggle*, at http://www.eiffel-nice.org/eiffelstruggle/2005/results.html.

[14] OpenGL, at www.opengl.org.

[15] Michela Pedroni: *Traffic v. 3.0.0*, at traffic.origo.ethz.ch.