# Mental Vision: a Computer Graphics teaching platform

Achille Peternier[1], Daniel Thalmann[1] and Frédéric Vexo[1]

Virtual Reality Laboratory (VRLab), École Polytechnique Fédérale de Lausanne (EPFL),
CH-1015 Switzerland

**Abstract.** We have developed a learning platform to simplify and improve teaching and practice of Computer Graphics for beginners and advanced students. Our goal is to offer a set of tools to help students having a better mental vision of the abstract notions introduced during the course. Our platform is composed by a pedagogical-oriented, intuitive and user-friendly graphic engine, offering a powerful amount of features with an extremely easy and comfortable interface, and a set of interactive and collaborative applications (called *modules*) to use during lessons and workshop sessions to present complex concepts in an easy and clear way. In this paper we expose how we have built our platform, what it offers and which role will play in our courses.

## 1 Introduction

Computer Graphics (CG) can be, like modern physics or advanced mathematics, a difficult and non intuitive topic to explain from a teaching point of view. It is extremely complex to illustrate spatial concepts or multi-dimensional geometric operations by only recurring to words or static images. The help of videos is also often limited and approximative because of their lack of interaction and the impossibility to modify in real-time the parameters of the algorithms.

During practical works or student projects, a concrete application of such concepts on micro computers requires knowledge related to specific hardware/software initialization and access, through complicated libraries and API which are time-expensive to get comfortable with, diverting student attention from practicing with CG. Such time expensive tasks are also unavoidable when students of advanced classes practice with Virtual Reality (VR) topics and need to resort to a graphic engine (or have to build one from the scratch) to apply and experience more high-level concepts like humanoid animation, inverse kinematics or virtual agents interacting in a 3D space. There is a good amount of open-source or free graphic engines and libraries, like Ogre (http://www.ogre3d.org) or Crystal Space (http://www.crystalspace3d.org), which offer an important set of functionalities. Unfortunately, such engines are still quite difficult to handle and require a good amount of learning time to start mastering them, so they are not well suited for a semester project. Furthermore, they are mainly oriented to game-development and targeted to developers used to CG programming (as pointed by Coleman and al. in [1]). Finally, if practical sessions are teamwork oriented, the lack of a unique, common and shared platform may lead to confusion and reduce or even obstruct cooperation between students and students and between students and assistants. The experience we gathered with master and undergraduate student projects in [2] showed us that offering original

and funny project topics was motivating for learners but the amount of different tools and libraries they used to build their solutions was a really nightmare for assistants and teachers. Finally, a lot of problems also occurred during public presentations of their works, because of the need to run the demos on different computers, often wrongly configured or non compatible with their code.

To avoid such kind of problems, we have created a set of software tools to improve the quality and comprehension of our CG and VR courses and to simplify and unify student works, offering them a simple but complete and robust graphic engine to use during the practical sessions and projects. To break the lack of dynamism and interactivity given by slides, images and videos during the teaching class, we have developed a set of applications featuring real-time and dynamic demonstrations of the presented topics. Our demonstrators allow students to directly act on parameters and algorithms, offering a dynamic cause-effect explication, unavailable through static methods. These demonstrators are a set of interactive compact applications (modules) which show in a clear and simplified way complex notions like sweeping surfaces, Bézier patches, camera clipping planes, etc. Some of them also feature collaborative support by sharing their interface on a network-based client-server architecture. These modules are extremely lightweight software which can seamless be distributed over internet, included in presentations or executed on handheld devices: in facts, thanks to their versatility, they can also be executed on laptops or Personal Digital Assistants (PDA), thus be brought and used directly during the lessons.

For the workshop sessions, to improve focus on the concepts instead of the corollary frameworks, we have developed a pedagogical-oriented generic graphic engine (named MVisio), with a learning curve of few minutes, allowing beginner students to immediately and efficiently deepen in their exercises without having to care about out of topic operations like hardware initialization, system configurations *et similia*, and offering to advanced learners a set of functionalities currently used in Virtual Reality related projects (like direct support for VR devices such as stereographic head-mounted displays, multi-screen environments, motion trackers, etc.). Finally, such an engine is extremely well known by assistants who can offer immediate and competent support to students experiencing problems during their projects done with it.

In this paper we show why we decided to create this platform, what other researchers did in the past in this domain and what are the new contributions brought by our work. We will present the details about the content and technical aspects of our solution, both for the creation of the modules and the design of the graphic engine. We conclude then with a report of the first experience we gathered by using our platform on student projects and public presentations.

## 2 Related work

Computer Science (CS) education is constantly evolving and requires both teachers, documentation and tools to be continuously updated. Computer Graphics, as a subset of CS, follow this rule as well. Different approaches have already been tested in the past to reinforce the support for CG courses and/or to offer development platforms for CG teaching purposes.

Towle and De Fanti first identified in 1978 with GAIN [3] the interest to offer interactive applications to use at home as an option to practical work sessions, done in cooperation with other students, and as a more intuitive way to learn than with just a manual or a workbook. We extended this idea with our modules, by integrating them directly with the class documentation and by adding multi-user interactivity to them. The benefits offered by multimedia contents for CG teaching purposes are also shown by Song and al. in [4]: interactive modules reduce learning time and improve use and diffusion of contents over the web, potentially targeting more people without additional costs. Many CG topics can also be taught by recurring to games: for example Hill and al. used in [5] puzzles and games to reinforce the learning objectives. Similarly, Becker in [6] used video-games as motivator for programming applications on Computer Science classes. We already had a positive feedback by offering gaming projects during the course of advanced Virtual Reality in [2]: with MVisio, we want to provide to the learners a tool allowing them to more benefit from this option, by giving them all the instruments they need in an easy and comfortable way.

About the creation of pedagogical oriented graphic engines, when 3D graphic accelerator cards for micro computer weren't available, Clevenger and al. developed a graphic engine to supply students with a learning platform (called TUGS) in [7]. Their goal was to offer a support to students to immediately render some images and to allow them to substitute parts of code of the TUGS engine with their own later in the semester, in order to have a full working platform since the beginning of the class. Their approach was particulary useful before the large introduction on personal computers of graphic APIs based on 3D hardware acceleration like OpenGL and DirectX, which substituted the expensive need to develop a custom rasterizer. Coleman and al. created Gedi [1], an open-source game engine for teaching videogame design and programming in C++. Based on the same principle to create a pedagogical engine, with MVisio we want to provide a more generic software that can be used not only for games but also for Computer Graphics and Virtual Reality applications, by adding for example support for VR devices like head-mounted displays or trackers.

Finally, all these lectures showed that interactive applications and specific graphic engines were generally an advantage and an improvement to the quality of the CG courses. This motivated us to deepen in this way by creating a set of tools utilizing all the hardware and software improvements of the last few years, as Owen did consequently to the evolution of 2D graphic hardware for micro-computer in [8], in order to offer an updated and modern platform for the teaching of Computer Graphics. In the next sections we will present all the details about our work.

## 3 Teaching platform

Our teaching platform is built by two separate entities: a set of compact applications, showing selected techniques and algorithms of Computer Graphics, and a pedagogical-oriented graphic engine to be used for student projects and practical works. We define our solution as a *platform* because we use the same tools during the whole class, from practical works to documentation, thus our framework covers the entire process of teaching, from the presentations to the student projects.

### 3.1 Pedagogical modules

Our pedagogical modules are compact applications which are used to illustrate in a plain and simple way complex Computer Graphics topics like sweeping surfaces or Bézier patches. Every module presents a single topic, in order to avoid confusion and simplifying interaction and comprehension. A module is typically composed by an intuitive interface allowing to dynamically modify the parameters of the exposed algorithms, in order to clarify how they work and change. A screen-shot of the related class slide is usually included as well.

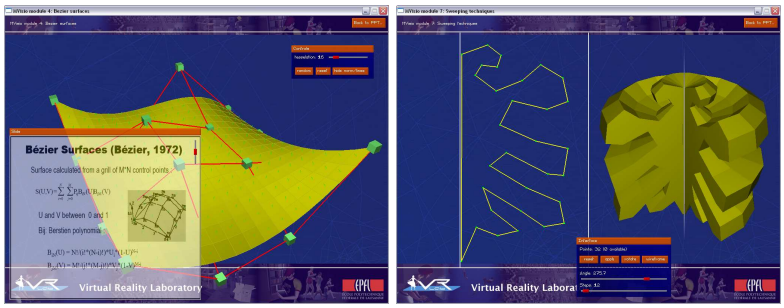


**Fig. 1.** Modules: Bézier surfaces (left) and sweeping surfaces (right)

The Poseidon project is a promotion done at our school to offer new personal computers at extremely interesting prices to our students (http://poseidon.epfl.ch). Thanks (also) to this project, an increasing amount of listeners brings a laptop PC or a PDA during the lessons. Consequently, we decided to introduce an interactive interface, allowing student computers to act like a remote controller enabling them to modify the parameters shown on the room wall display during the class lesson, offering them a way to directly interact with the teacher and other attendants. Typically the teacher computer works like a privileged module which can offer and receive control requests from client (student) PCs or PDAs. Of course, this option must be explicitly activated on the server-side to avoid request spam or unwanted modifications during teaching explanations. A common scenario suitable for this feature is either when students desire to ask something and can benefit of the interactive interface to better explain and share their doubts with the rest of the auditorium or when teachers ask students to solve a problem and then to show publicly their results.

**Technical overview** Modules are built on the top of the MVisio engine (see next section) which allows an high recycle of piece of code and data, minimizing sizes, system requirements and bandwidth when downloaded from the course homepage. Thanks to their compactness, modules are also suited to be directly included in presentations (PowerPoint) and to be started from the slide describing the technique they show: in facts, the goal of this software is to act as an interactive slide-show. Moreover, being

based on MVisio, modules can also be executed on several platforms (Windows family, Linux, MacOS) and devices (PC and PDA).

The shared interactive interface is based over a simple TCP/IP protocol. Fortunately, our class rooms are covered by a WiFi network allowing PCs and PDAs within the auditorium to be connected to the web. When started, modules look for a connection with a remote server acting as a broadcaster of messages. Two levels of users are recognized by the server: teachers and students. When launched, modules allow to specify a command-line password used to determine the rights of the connecting client. Teacher and student modules are exactly the same: features exposed by the interface are determined during the login. Typically teacher modules send a copy of every operation done to the server, which stores the information. On demand, client modules can synchronize their interface by a simple click. This allows students to play with the modules but still offering them to revert their clients to the state shown by the teacher on the wall display. Teacher modules can also allow student clients to act as a teacher client, in order to share the current state of their modules with the rest of the class. This operation requires an authorization and works in this way:

- the teacher enables remote controlling by clicking on a special button on his/her module interface. This button is available only on teacher modules when correctly logged to the server.
- the server sends an invitation message to every student connected client.
- students receive a pop-up window asking them if they want to accept the invitation.
- when a student accepts, pop-up windows on other clients disappear and only the client that accepted the invitation can now forward its modifications to the server.
- the student can now show his/her parameter modifications on the wall display and the other clients can synchronize with his/her parameters on demand (as if the student client were now a teacher client).
- after this, the teacher clicks on another button and gets back full control.

**Fig. 2.** Hermite interpolation and Kochanek-Bartels spline module: listeners can bring their PDAs during the class to directly play with modules and interact with the auditorium

A typical use of this option is for example when a student has a question and can benefit of this client-server architecture to explain more clearly his/her doubts with

a graphic support and without having to go to the blackboard. Because modules are available both on PCs and PDAs, it is more and more a common thing to see students bringing their notebooks or handheld during the lessons.

**Case of study: camera handling** In this section we show how our module about camera handling and clipping planes works. In concrete, this application splits the screen in two panels, one (right) showing the point of view of the camera, the other (left) displaying the full scene from a remote third-person viewpoint. A movable window displays both current camera model view and projection matrices. Some slides allow to modify the distances of the near/far plane and the angle of view. Modified parameters directly affect the rendering on the right panel. Both near/far planes are displayed as transparent yellow rectangles on the left window. By moving the cursor on the right panel, objects selected by the mouse start blinking: with a click on them, they become the current camera target. The camera can then be moved by clicking and dragging the mouse on the left panel: the camera still keeps its view locked in the current selected object on the right panel (fly-by). The scene loaded as example for this module is an high detailed pub interior. This scenario allows students to see how clipping planes and field of view parameters affect the rendering. They can freely move the camera and see how the model view and projection matrices change, tracking at the same time what visually and mathematically happens when they play with the parameters.

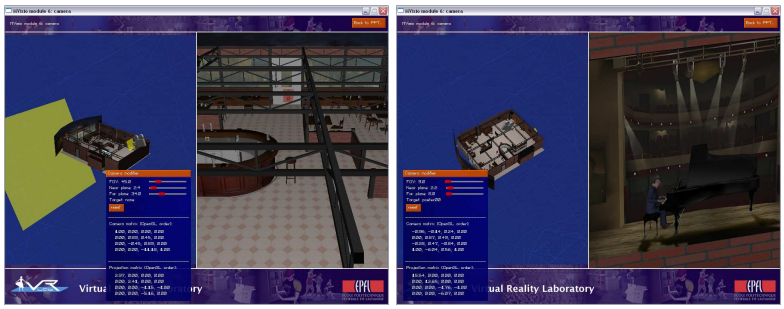

**Fig. 3.** Camera handling: third-person overview of the scene (left) and rendering from the current camera viewpoint targeted on a wall poster, as requested by the exercise (right)

As an exercise for students, we have added some posters on the pub's walls. Because of the complex architecture of the scene, such images are often hidden behind other objects and cannot be easily spotted. We asked our students to answer to some questions by retrieving the requested information by zooming and spotting the different posters. For this, students had to learn how to move the camera within the scene, to track objects by selecting them, to use the angle of view slider, to zoom in or out and to move the near/far planes to remove objects obstructing the camera. After this exercise, the students were able to show their results to other people thanks to the shared interface moderated by the teacher.

### 3.2 MVisio graphic engine

MVisio (Mental VISIOn) is a lightweight, stable and user-friendly 2D/3D graphic engine. The goals of MVisio are several: offering an extremely easy to use and an intuitive interface to 2D/3D graphics, being able to run on almost every available desktop PC or laptop by automatically adapting the rendering quality and settings, being also able to work on other devices like PDAs and CAVEs, being compact in sizes and system requirements and finally being fast and modern.

We developed MVisio in order to offer a common developing tool to advanced students for their Virtual Reality related practical works and projects. In the advanced classes, learners don't have to take care anymore about low-level operations such as rasterizing triangles or computing normals, but have to implement high-level applications like little videogames or virtual worlds (see [2] for a description of our old projects). What students need is a platform allowing them to directly work with textured models and characters, cameras and lights, offering the opportunity to efficiently animate models and dynamically modify the scene and the objects. Students also often need a 2D interface able to handle buttons, windows and text over a 3D rendered scene. MVisio brings all this functionalities in an extremely easy way, allowing students to deepen immediately into the lesson topics without wasting time learning things not related with the course goal. We adopted few years ago a C++ object-oriented class base code during our practical works: although our learners are usually IT engineers, a good amount of students was not used to C++ and spent more time learning it than using this language to solve the CG exercises we suggested. MVisio, even if strongly based on an object-oriented architecture, exposes a Java-like interface which requires a minimal knowledge of C++. Moreover, Java is a language largely taught in our school, thus the step to MVisio for not skilled C++ programmers is less uncomfortable. Finally, the included 2D GUI system is accessible, from a programming point of view, like a simplified version of the Java windowing interface.

**Technical overview** MVisio is built in C++ and uses OpenGL (http://www.opengl.org) on PC and OpenGL ES on PDA (http://wwww.khronos.org) as 3D rendering API. We used Microsoft Windows CE 4.2 based PocketPCs. On WinCE, we worked with two different versions of OpenGL ES: one in software mode distributed by Hybrid (http://www.hybrid.fi) and compatible with almost every existing PDA, and a second one working only on the Dell Axim x50v (http://www.dell.com), using an hardware acceleration chip released by PowerVR (http://www.pvrdev.com). We also used Simple DirectMedia Library (SDL, http://www.libsdl.org) to initialize windows and rendering contexts in order to improve code portability over different platforms. No other library like Glu or Glut have been used to reduce sizes and dependencies of the engine: the functions needed for perspective computing, mipmap generation or image loading have been directly included into MVisio and extremely optimized for speed (e. g. our mipmap generator is about five times faster than the Glu one). Actually, a release .dll only weights 132 KB (compiled under Visual .NET 2003).

The internal architecture of MVisio is extremely object-oriented based: this allowed us to further reduce code sizes by reusing long portions of code and to develop an

extremely simplified interface. MVisio entities like lights, cameras, fonts etc. are created in a Java-like way and don't need to be released once used: MVisio manages with that.

Complex tasks like scene-graph handling or mesh instancing (a same mesh displayed many times in the same scene with different parameters at several locations) are transparently and automatically performed by MVisio: the user simply declares which objects have to be rendered and announces them in a way similar to the OpenGL glBegin/glEnd methods. For every frame, the user can move, modify objects and then pass them to MVisio, which stores a reference to the entities in an internal list. Once all the objects announced, MVisio performs several operations on that list (like transparency sorting, clipping, HSR, ect.) and renders the final image.

One of the most interesting features for students is that we developed a plugin for 3D Studio Max which allows to export with just some clicks complex scenes, with textures, lights, hierarchical scene-graphs, animations, etc. All this information is transparently converted, stored on the right places to be loaded and accessed through MVisio with just few lines of code (in facts, you can load and rotate the pub scene in fig. 3 with only 7 lines of MVisio calls, included the engine initialization and release). We have a good amount of 3D models in our database: students can simply choose since the beginning of the semester the meshes they want and then modify and use them on their projects.

**Case of study: MVisio used on projects** We observed in [2] that proposing little games as projects for the course of Virtual Reality motivated students and produced a good feedback, both from the quality of the final products and the satisfaction of the participants. Creating video-games by using advanced VR interfaces was an amazing task for students and an excellent practice from a pedagogical point of view, too: they used data-gloves, head-mounted displays, motion capture and our haptic workstation, which are all complex devices to deal with. With the introduction of MVisio, we dramatically simplified all the aspects related to the rendering and handling of the 3D content, allowing learners to furthermore and immediately focus on other activities more related to Virtual Reality than to Computer Graphics.

For example, a student has recently developed for his master project a virtual juggler. The idea was to use the haptic workstation to simulate juggling with three balls using two hands. The haptic workstation provided hands and fingers tracking as well as force feedback on fingers and wrists. A head-mounted display with head-tracking was also used to provide an immersive environment. The student used MVisio as graphic engine to put the user in a virtual pub and to animate the balls. MVisio also managed the stereo rendering system required by the HMD. On a four months project, he spent less than a week working on the rendering part of his virtual juggler, investing the rest of the time on difficult topics like implementing a good physic system for simulating ball animations, force feedback and dealing with the complex use and calibration of the haptic workstation.

Benefits resulting from the device independency and portability of MVisio were largely used in another student project aiming at tracking on PocketPC the position of a real mini-blimp. The goal was to display on the PDA screen a 3D model of our school with the blimp correctly localized. Blimp position was gathered through a WiFi connection and a GPS system. At the beginning of this project, the PDA version of

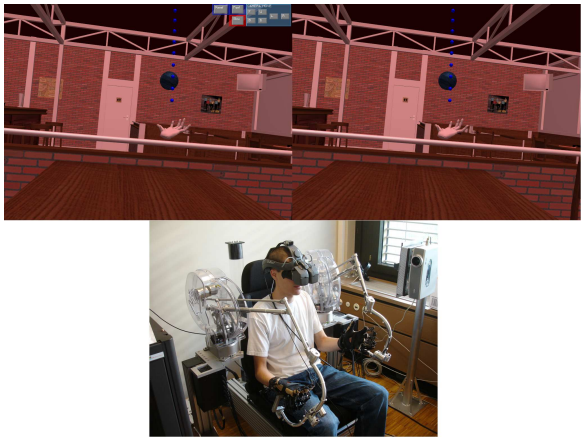

**Fig. 4.** HMD stereographic rendering (top); a user in the haptic workstation (bottom)

MVisio was in development and not available. Students worked on MVisio for PC and switched to the PocketPC just at the end of the project, in a matter of hours, without needing to modify their code or their models. These options have also been extremely useful on other student projects that worked on the PDA platform: even if learners did'nt have a PocketPC at home, they still could develop on PC and usually move their code, without any modification, on PDA. An overview of other projects achieved in our laboratory is available at [9].

As a direct confirmation of the versatility and user-friendliness of our engine, MVisio has also been largely used by assistants and member of our laboratory during its developing phase on several research projects (like in [10] to display steering lines toward oriented targets). Thanks to its auto-tuning system and stability, MVisio is a good choice as rendering engine for distributions and demonstrations.

## 4 Conclusions

After several months of work spent to design and develop both modules and the graphic engine, we are actually using them concretely on the ongoing course this semester. So far we tested our platform *in vitro*, on internal projects done by assistants and some pilot student projects. Modules have been introduced during public presentations but have been added to the class documentation only since October 2005. Even if we need to wait the end of the semester to obtain a more complete feedback about the usefulness and further improvements required, first student impressions are extremely positive, mainly about the good ratio between the features offered by MVisio and the time required to access them. The multi-device support of MVisio also spontaneously motivated some learners to create cross-device projects, supporting both PC and PDA. Some modules, like the ones about curve generation and sweeping surfaces, rose interest of teachers coming from other institutes as well, involved in mathematics and physics. We are also exploring the idea to bring modules on mobile phones: last generation mobile phones

have enough computational power to support basic 3D graphics. Despite of the good amount of students bringing their laptops or PDAs during the lessons, mobile phones are still more widespread than notebooks and PocketPCs.

In the near future, some more modules with advanced topics like skeletal animation, inverse kinematics and walking models will be added, as well as a set of simple tutorials about how to use MVisio. We will also insist on the development of MVisio for the CAVE (actually in an early stage), in order to allow students to practice with this device too, typically difficult to access because of its network and multi-computer based architecture.

## 5 Acknowledgements

Many thanks to the Centre de Recherche et d'Appui pour la Formation et ses Technologies (CRAFT, http://craft.epfl.ch) for supporting the Mental Vision project and to Renaud Ott for his precious contribution in helping us improving and debugging the graphic engine.

## References

1. Ron Coleman, Stefen Roebke, and Larissa Grayson. Gedi: a game engine for teaching videogame design and programming. *J. Comput. Small Coll.*, 21(2):72–82, 2005.
2. M. Gutierrez, D. Thalmann, and F. Vexo. Creating cyberworlds: Experiences in computer science education. In *Proc. International Conference on Cyberworlds*, 2004.
3. Tom Towle and Tom DeFanti. Gain: An interactive program for teaching interactive computer graphics programming. In *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 54–59, New York, NY, USA, 1978. ACM Press.
4. Wen-Chai Song, Shih-Ching Ou, and Song-Rong Shiau. Integrated computer graphics learning system in virtual environment - case study of bezier, b-spline, and nurbs algorithms. In *Proc. Fourth Internation Conference on Information Visualisation*, 2000.
5. John M. D. Hill, Clark K. Ray, Jean R. S. Blair, and Jr. Curtis A. Carver. Puzzles and games: addressing different learning styles in teaching operating systems concepts. In *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education*, pages 182–186, New York, NY, USA, 2003. ACM Press.
6. Katrin Becker. Teaching with games: the minesweeper and asteroids experience. *J. Comput. Small Coll.*, 17(2):23–33, 2001.
7. John Clevenger, Rick Chaddock, and Roger Bendig. Tugs a tool for teaching computer graphics. *SIGGRAPH Comput. Graph.*, 25(3):158–164, 1991.
8. G. S. Owen. Teaching introductory and advanced computer graphics using micro-computers. In *SIGCSE '89: Proceedings of the twentieth SIGCSE technical symposium on Computer science education*, pages 283–287, New York, NY, USA, 1989. ACM Press.
9. Virtual Reality Laboratory (VRLab EPFL). Student Projects Repository. http://vrlab.epfl.ch/public/students_projects.
10. R. Boulic. Proactive steering toward oriented targets. In *Eurographics Short presentation program (to appear)*, 2005.