Volume xx (*200y*), *Number z, pp. 1–8*

eNVyMyCar: a multi-player car racing game for teaching Computer Graphics

F. Ganovelli and M. Corsini

Visual Computing Laboratory, ISTI-CNR, Italy †

Abstract

The development of a computer game is widely used as a way of conveying concepts regarding Computer Science. There are several reasons for this: it stimulates creativity, it provides an immediate sense of achievement (when the code works), it typically covers all the aspects of an introductory course, and it is easy to find ideas just by looking around and finding stimulation from one's environment and from fellow students. In this paper we present eNVy My Car (NVMC), a framework for the collaborative/competitive development of a computer game, and report the experience of its use in two Computer Graphics courses held in 2007 . We developed a multi-player car racing game where the student's task is just to implement the rendering of the scene, while all the other aspects, communication and synchronization are implemented in the framework and are transparent to the developer. The innovative feature of our framework is that all on-line users can see the views produced by their fellow students. This motivates students to improve their work by comparing it with other students and picking up ideas from them. It also gives students an opportunity to show off to their classmates.

Categories and Subject Descriptors (according to ACM CCS): K.3.2 [Computer and Information Science Education]: Computer Science Education

1. Introduction

Introductory Computer Graphics courses form part of many Engineering and Computer Science course programs. Computer Graphics, in its broader sense, includes a large number of sub-fields such as geometric modeling, rendering techniques, design, computer animation and computational photography, just to cite a few. Normally these are covered as part of specialized courses (for example Stanford University offers 13 different courses closely related to CG, ranging from *Mathematical Methods for Graphics* to *Advanced Geometric Algorithms*). In this paper we will particularly focus on introductory Computer Graphics courses.

Normally, an introductory Computer Graphics course gives an overview of the field and then focuses on the rasterization-based rendering pipeline of modern graphics hardware. At the end of this type of course the students should be able to develop an interactive 3D application involving geometrical manipulation and providing a more or less sophisticated rendering, possibly including non-local lighting effects such as shadows, ambient occlusion, reflection of the environment on the objects' surface and so on. CG concepts translate quite naturally to practical exercises that can be carried out with a computer, especially when an API such as OpenGL of DirectX takes care of the underlying details. It is a common practice to organize all the exercises in a single effort to implement some kind of graphical application. For this purpose, we propose *eNVyMyCar*, a multi-player car racing game framework specifically designed for learning Computer Graphics concepts.

The choice of using a software framework for teaching Computer Graphics is based on several recent teaching experiences in the field of Computer Sciences. For example, computer games have been used to teach object-oriented programming [CC07, CC05] and pattern design [Ges07] and Computer Graphics itself [HS05]. The Cannibal Experience [BDHB08] game engine was specifically designed at the

† name.surname@isti.cnr.it

Delft University to be used by students to learn many aspects of game development. ETH setup a game programming laboratory [STG08] with the specific aim of providing to the students in-depth understanding of Visual Computing concepts and teamworking skills. Many other studies shown the beneficial effects of teaching computing in context such as Xu et al. [XBK08] where robots and games are used to drive motivation and retention.

The eNVyMyCar framework is a car racing game implemented with a client-server architecture, where the student's task is simply to implement the rendering part alone. The framework is designed so that the student does not need to take care of the networking issues or of the physics (although such aspects might be included in a more specialized course). They are only required to understand a few very simple C++ classes describing the scene and render it interactively. The description of the scene is minimal and concerns only the parts that can physically influence the race, i.e. the streets, the buildings, and the trees (others can be easily added). They don't look to concern themselves with how things look. This then gives total freedom on how to represent their car, the terrain, the sky etc.

In this sense it is quite straightforward to see that a racing game is a perfect scenario for progressively mapping CG concepts to code, for example the geometric transformations (the front wheels that rotate and steer), the use of impostors (the billboards for the trees), and the environment mapping (the dynamic reflection on the car). However, freeing the students of non CG problems was not the only reason for eNVyMyCar. We also wanted them to be able to share knowledge and discuss problems in the same platform, and possibly to write modular code that it could be moved from one client to another (instead of the thousands-lines long *main* bodies).

Section 2 will give a detailed description of the eNVyMyCar framework. In Section3 we show how several Computer Graphics techniques can be fitted into the project to prove how this approach can be a very useful means for teaching both basic and advanced Computer Graphics topics. The results of the use of eNVyMyCar as educational tool are reported in Section 4 and the conclusions are outlined in Section 6.

2. eNVyMyCar: the framework

NVMC is a car racing multiplayer game realized with a single server-multiple client architecture.

The world represented by NVMC is made of a static part and a dynamic part. The static part consists of the circuit, the trees, the buildings and it is entirely stored by both the server and the clients. The dynamic part consists of the *state* of the cars (their position, orientation and speed) and the position of the sun, hereafter the *state of the race*.

The simulation of the race is run on the server, which updates the state of the race and broadcasts it to all the connected clients.

A client corresponds to a player of the race. It receives the state of the race from the server, renders the scene and control the player's car by sending messages to the server (such as. INCREASE_SPEED, STEER_LEFT, BRAKE etc.) which are processed and accounted for in the simulation. The communication is asynchronous, meaning that messages are sent independently from each client and from the server.

This may simply sound like a classic client-server scheme for a multiplayer game, with commands sent from the clients to the server and state of the system broadcasted from the server to all the clients. The novelty of NVMC is that there is another kind of data that the client may send, which is a snapshot of the view provided to the player. The snapshots follow the same path as the commands, except that they do not influence the simulation of the race but are simply rebounced to all the other clients.

In this manner, while the client is running the developer may see also snapshots from the other connected clients. So this is where the "the envy factor" comes into play. If a student see snapshots of other clients they feel envious and are stimulated to improve their own work. Of course it is not really envy but the curiosity and desire to obtain a visually pleasant result that motivated students to implement new features influenced by each other ideas. This is very different from comparing the students' work at the exam or at fixed milestones. It is more like forming a team where each student can develop their own version.

Often course projects are assigned to small groups of people but then often the individual contribution of students of the same group to the project must be figured out with an oral examination, while little feedback is given during the development of the project. Furthermore, students of the same group are typically in charge of different aspects of the project so they may specialize too much in one topic and lack insight into others (for example one student may learn everything about normal mapping but never attempt to instance a Vertex Buffer Object and so on). With NVMC every single student is in charge of the whole project. They can exchange ideas, tricks and code snippets, (as long as each student is able to explain clearly every line that appears in their code) and eventually everyone will have tackled all the difficulties of the development.

The instructor may be connected to the server with their own client and see how the projects are going. Note that the upload of a snapshot is done upon client request and not automatically. The developer may decide to code the uploading of a snapshot at fixed intervals of time or, as all the students did, associate the event with a key. This mechanism may also be used by the teacher to provide suggestions to the class,

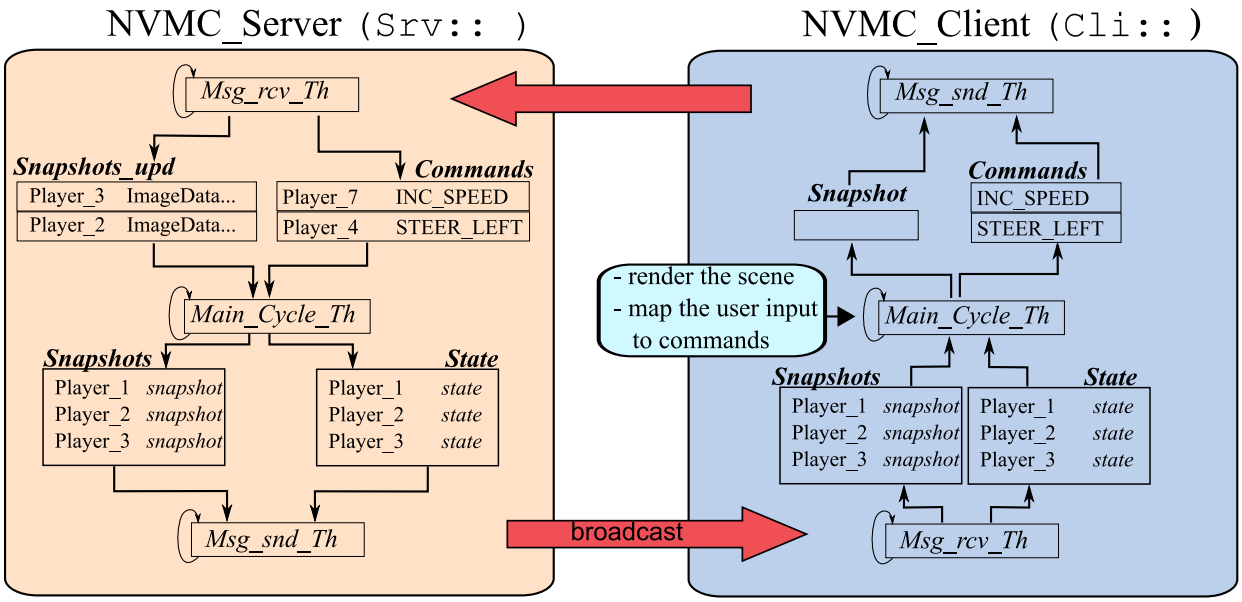

Figure 1: *Software architecture of the NVMC framework.*

by implementing their own client (supposedly better than all those of the students) and uploading snapshots.

2.1. Implementation

Figure 1 shows the structure of NVMC. The boxes with returning arrows represent process threads, the boxes named in bold are queues and the arrows are directed as the information flow.

`Cli::Main_Cycle_Th` is the main thread of the client and it is responsible for rendering the scene, for writing commands to be sent to the server in the **`Cli::Commands`** queue and for saving a copy of the current view in the `Cli::Snapshot` memory area. The thread `Cli::Msg_snd_Th` reads the commands from the queue **`Cli::Commands`** and transmits them to the server. If the commands is SEND_SNAPSHOT then the snapshot is read and sent to the server. On the server side, the thread `Srv::Msg_rcv_Th` receives all the communication from the clients and stores commands in the local **`Commands`** queue, where each entry is a couple *(player, command)* and snapshots in the **`Srv::Snapshots`** area where the most recent snapshot received from each client is stored. The `Srv::Main_Cycle_Th` is responsible for running the simulation of the race, which consists of updating the position of each car, and saving the state of the race in **`Srv::State`**. In addition, it updates the **`Srv::Snapshots`** area with the snapshot received in **`Srv::Snapshots_upd`**. `Srv::Msg_snd_Th` continuously broadcasts the state of the race **`Srv::State`** and the updating of the snapshot (when necessary) to all the clients.

Back in the client side, the thread **`Cli::Msg_rcv_th`** receives the messages from the server and copy them to the **`Cli::Snapshots`** and **`Cli::State`** areas, where they will be read from `Cli::Main_Cycle_Th`. The NVMC framework also provides a standalone mode in which the server threads are launched within the same process (the client). This is handy when the student works at home and does not want to launch a separate process.

2.2. Interfaces towards the developer

The students' goal was to implement their own rendering engine for the game without necessarily knowing the underlying architecture, so NVMC provides a very simple software interface. Basically the developer only needs to know the definition of a few classes: *Circuit*, *Car*, *Building* etc., to be able to draw them, and three functions:

- **Command(**command_name**)** to issue a command to the server
- **UpdateScene()**, which is called prior the rendering cycle and updates all the dynamic data structures
- **DrawScreenshots()**, which is called at the end of the rendering cycle and draws the other clients' screenshot (if there is one).

All the code is written in ANSI C++ using QT [qtl] for the multithreading and networking aspects and the VCGLib [vcg] (a header only library) for loading and rendering geometric models and performing simple matrix computations.

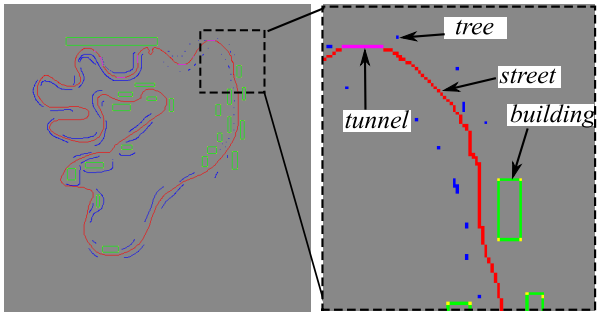

Figure 2: *A simple example of circuit encoded in a bitmap image.*

Along with the framework we provided two "hello world" clients for which we used SDL [sdl] and glut [glu] respectively, to handle user commands through mouse/keyboard and windowing. However these libraries do not have to do with the NVMC framework itself and other libraries could equally well be used.

2.3. Creating a circuit

Along with the framework for playing the game, we also provided the students with a simple program to create new circuits. Instead of using general tools for modelling, such as Blender or Google SketchUp and then convert them to our data structure, we decided to use RGB images to code a scene and therefore to write a simple converter from a RGB image to NVC format. Although obvious limitations arise using this encoding, it was more than enough for our needs and a simple image editor (i.e. Microsoft PaintBrush) was sufficient to create a new circuit. Figure 2 shows an example of an image coding a scenario.

3. How Exercises fit into the project: from a black screen to a working client.

It does not take long to describe the NVMC framework to students, since it is a matter of showing a few simple c++ classes. Depending on the students' background additional work may be needed. For example, it may be that most student have an object-oriented programming background but little or no knowledge of C++. In fact, in most OOP courses Java, rather than C++, is used to illustrate computer science principles. Hence, sometimes two or three lectures are needed to fill in any specific gaps.

We stress that the development of the project may start at the same time as the theory since, as shown below, the basic concepts and related exercises naturally map onto the task for implementing a working client.

3.1. Basic CG exercises

The minimal expected result from an introductory course on Computer Graphics is that the students become familiar with the theory and implementation of geometric transformations, lighting and texturing. Below we show how these concepts can be deployed to client functionalities.

Geometrical transformations: One of the first problems that a student encounters is how to handle basic geometric transformations correctly. Such transformations are necessary to visualize the car during its movement, to place the elements of the scene, and to manage camera movements. Obviously, all the students have to deal with this step. In addition, students can use car models composed by several parts and moving such parts in order to produce a more realistic animation of the cars, meaning, for example, that they need to compose roto-translation matrices to make the wheels roll and steer.

Lighting: Firstly, the basic Phong illumination model that OpenGL provides is used to implement very basic lighting: typically and ambient light plus a directional light corresponding to the sun (which is part of the state). It is common to try to do something more, for example to put lamps along the street implemented with a positional light and an emissive material, or use the spotlight to implement headlamps (which also involves some more linear transformation). This example is particularly useful to understand how much phong lighting is dependent on triangulation (the terrain is flat and typically tessellated with few polygons, therefore students are not happy about how the headlamps light the street).

Texturing: The first approach consists in applying textures to the buildings, the terrain, the road and the car to ensure a minimal visual richness. Typically the student (wisely) decides to use one tileable texture for the terrain and another one for the street. The faćades of the buildings need to be textured with special care to keep the appearance consistent with the scale of the scene (100 meter large windows are not acceptable).

3.2. Advanced CG exercises

To be able to implement the basic functionalities of the client is the minimum required to reach a sufficient score in the assessment. At this point the students are encouraged to improve their clients by adding new functionalities or enhancing the existent ones. Thus, a (not obligatory) list of choices is given to the students, each with a brief explanation and some references to further documentation.

Some students opt for simple techniques concentrating their effort in creativity while others try to implement more complex techniques. Below we report their preferences:

Billboarding: Billboarding is one of the image-based rendering technique shown to the students during the course. One of the typical uses of billboarding is of course for

trees. Most students replicate such example to show the trees in theirs scenarios or to implement lens flares. Some students use billboarding in other way such as to add streetlamp or to render the interior of the vehicle with screen-aligned billboarding (see Figure 4 top-left)

Projective texture mapping As previously stated, students are generally not happy with headlamps implemented with per-vertex lighting, whereas with projective texture mapping they obtain a much more satisfactory effect. They also need to use multitexturing and blending, i.e. to become more confident with texture mapping, to achieve a satisfying result. (see Figure 4 top-right)

Skybox Generally students are not satisfied with the look of the scenery until they see a sky over the car and a landscape around the main road. Most students exploit cube mapping to render a skybox and its reflection on the car.

Dynamic cube mapping Reflection of the whole environment on the car is accomplished by Dynamic Cube Mapping (see Figure 4 top-left).

Lighting Models: Students who want to make practice with vertex or pixel shaders are advised to implement one of the lighting models (e.g. Cook-Torrance, Oren-Nayar, Minnaert) they have seen in class.

Shadow mapping: Shadows add realism to the rendering and provide a professional look to the final rendered scenery. Our course did not deal with other shadowing techniques than shadow mapping such as volume shadows or soft shadows.

Accumulation buffer: Students use accumulation buffer to implement some interesting effects such as motion blur or depth of field. The use of accumulation buffer to implement such effects is advised during the course as a simple alternative to the implementation with shaders.

Particle Systems: Particle systems can be used in several ways in a car race simulation. The particles could simulate dust when the car accelerate or fire when the cars crash with something (since the collision detection is not implemented in the framework this effect is usually enabled/disabled by the users). Another effect that students can add to their client with a particle system is an atmospheric effect such as rain or snow. Particle systems was not among the recommended choices, simply because they are not part of the course. Nonetheless, its dynamic nature attracted at least one student in two out of the three courses where NVMC was used.

4. Results

So far the NVMC framework has been used in three Courses: the Computer Graphics Course of the University of Siena held in 2008 (6 students) and in 2007 (10 students) and the University of Ferrara's Advanced Computer Graphics Course of the (14 students) held in 2007. It is important to underline that the choice of NVMC is not mandatory, i.e. students could choose to develop an NVMC client, to do another project chosen from a list, or not to a project at all (in this case a penalty to the final evaluation is applied). Nevertheless, 28 out of 30 students choose to use eNVyMyCar for their project.

In order to evaluate the effectiveness of the eNVyMyCar framework as a tool to learn Computer Graphics after the examination the students have been interviewed informally and almost all of them enjoyed the project and found the framework a useful learning tool. Apart from the positive interviews another factor that demonstrated the effectiveness of the framework was the good results obtained by several students. Here, we show some screenshots of the developed clients. About 60% of students implemented more than the basic features necessary to reach a sufficient score (good camera handling/standard lighting/texture mapping/skybox). For example some use shaders to implement complex lighting models. Others inserted particle systems to produce dust, fire, or other similar effects. Others improved the look of theirs car with dynamic reflections. A few used procedural techniques for some of the elements in the scene. Most used billboarding to render trees and projective texturing to draw headlights. In terms of the artistic aspects only very few students contributed with their own graphics (e.g. creating original textures) or developed interesting ideas from a visual point of view. This is probably related to the fact that the majority of the students had a technical rather than artistic background.

On the downside, although the framework has been designed to be very easy to use and the student has provided with an "hello world" fully working client, it would have helped them to have a detailed technical manual about the framework and a FAQ. This would have speeded up client development and help to avoid some troubleshooting, particularly for those students who tried to modify the framework a bit in order to fit their needs. Despite this, as just stated, all the students are able to finished theirs project within 2-3 weeks with good results.

We found one shortage in the students' background which usually is not optimal. In fact, even if the pre-requisite is the OOP programming, Java programmers could report some initial difficulty with the use of STL libraries and other minor aspects of the framework.

5. Discussion

eNVyMyCar is not a complete game: collision detection among cars and scene elements, scores and power-ups, and other typical features of game engines are not considered in the current version. The main reason for this is that they were not necessary for our purposes. Furthermore, we did not want to include difficulties *unrelated* to CG. Collision detection for example, could be integrated in NVMC in a course specializing in game physics. Similarly, Level of Detail representations, procedural modeling or real time global illumination techniques could be integrated in a more ad-

Figure 3: *Some clients developed by the students. (Top-Left) Dynamic reflections on the car. (Top-Right) Projective textures for headlights and tunnel lamps faked with textures. (Bottom-Left) Motion blur and lens flare. (Bottom-Right) Billboarding for showing trees.*

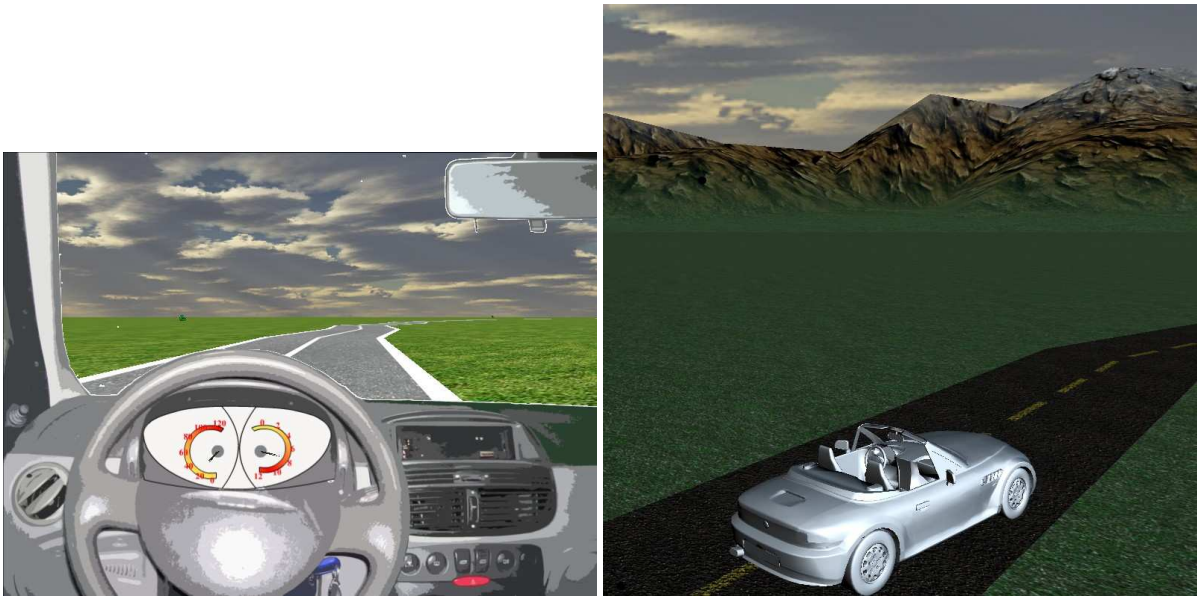

Figure 4: *More clients: (Left) Screen aligned billboarding for the view from the interior of the car (Right) The mountains are procedurally generated as a heightfield and rendered with a displacement map assigning a height-dependent texture (grass, mountain, snow).*

Figure 5: *Two players connected during the race. Note that each client provides its own view of the scenario, including the appearance of the cars.*

vanced course where these techniques would be explained in detail.

Finally, our framework can be tailored for use as an educational tool in other courses. For example, the framework can be easily reused for an Artificial Intelligence course, simply binding the commands to control the cars to an AI engine instead of the keyboard.

6. Conclusions

In this paper we have presented eNVyMyCar which is a framework to support the teaching of Computer Graphics at basic and advanced level. The use of the eNVyMyCar framework provides substantial advantages to the students: strong motivation given by the "computer games effect", personalized learning and retention.

At the time of this writing, eNVyMyCar has so far been used as part of four CG courses. Although a rigorous study about the effectiveness of the framework has not been conducted yet, student feedback was very encouraging. In addition, the framework can be easily modified for use in other Computer Science courses such as Artificial Intelligence, Games Physics and Advanced Geometric Modeling.

The future of this project is essentially to extend the use of eNVyMyCar to further CG classes, to gain more experience and finally to write a concise textbook.

The eNVyMyCar project is hosted by Source Forge and can be found at the URL: http://envymycar.sourceforge.net.

Acknowledgment

We wish to thank all the students of our courses for their feedback and their enthusiasm. Like many other researchers in Italy, we teach University courses as freelancers, so we also thank our affiliating institution, the National Research Council, for allowing us to take some time off our regular activities.

References

[BDHB08] BOERS J., DOBBE J., HUIJSER R., BIDARRA R.: From a Light CG Framework to a strong Cannibal Experience. Cunningham S., Kjelldahl L., (Eds.), Eurographics Association, pp. 15–19.

[CC05] CLAYPOOL K., CLAYPOOL M.: Teaching software engineering through game design. In *ITiCSE '05: Procs. of the 10th annual SIGCSE conf. on Innovation and technology in comp. sc.education* (New York, NY, USA, 2005), ACM Press, pp. 123–127.

[CC07] CHEN W. K., CHENG Y. C.: Teaching object-oriented programming laboratory with computer game programming. *IEEE Transactions on Education 50*, 3 (August 2007), 197–203.

[Ges07] GESTWICKI P. V.: Computer games as motivation for design patterns. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education* (2007), ACM Press, pp. 233–237.

[glu] Glut - the opengl utility toolkit. More info on: `http://www.opengl.org/resources/libraries/glut/`.

[HS05] HOETZLEIN R. C., SCHWARTZ D. I.: Gamex: a platform for incremental instruction in computer graphics and game design. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Educators program* (2005), ACM Press, p. 36.

[qtl] Qt `http://www.qtsoftware.com/`.

[sdl] Simple direct media library. `http://www.libsdl.org/`.

[STG08] SUMNER R. W., THUEREY N., GROSS M.: The eth game programming laboratory: a capstone for computer science and visual computing. In *GDCSE '08: Proceedings of the 3rd international conference on Game development in computer science education* (New York, NY, USA, 2008), ACM, pp. 46–50.

[vcg] Visualization and computer graphics library. `http://vcg.sourceforge.net/`.

[XBK08] XU D., BLANK D., KUMAR D.: Games, robots, and robot games: complementary contexts for introductory computing education. In *GDCSE '08: Proceedings of the 3rd international conference on Game development in computer science education* (New York, NY, USA, 2008), ACM, pp. 66–70.