EUROGRAPHICS 2008 / G. Drettakis and R. Scopigno
(Guest Editors)

Volume 27 (2008), Number 3

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

SUBMISSION ID: 1008

Abstract

The development of a computer game is widely used as a means to convey Computer Sciences concepts. There are several reasons for that: it is stimulates creativity, it provides an immediate sense of achievement when the code works, it typically covers all the aspects of an introductory course, it is easy to find ideas just looking around.
In this paper we present ***NVMC*** *(EnVy My Car), a framework for collaborative/competitive development of a computer game and report the experience in using it in two computer graphics courses held in the year 2007 by the authors. We developed a multiplayer car racing game where the student is only asked to implement the rendering of the scene, while all the other aspects, communication and synchronization are implemented in the framework and transparent to the developer. The novelty of our framework is that all the clients on-line are able to see the views provided by the other clients, which serves to motivate the students to improve their work by comparing it with the other clients, as a means to pick up ideas from the others and finally to show off with their classmates.*

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

1. Introduction

Nowadays, introductory Computer Graphics courses are present in the majority of Engineering and Computer Science programs. Computer Graphics, intended in its broader sense, includes a large number of sub-fields (geometric modeling, rendering techniques, design, computer animation, 3D photography, to cite a few) which are normally treated in dedicated courses (for example the Stanford University offers 13 different courses closely related to CG, ranging from *Mathematical Methods for Graphics* to *Advanced Geometric Algorithms*).

For what concerns introductory courses, they typically give an overview of the field and then focus on the rasterization based rendering pipeline. At the end for the course the students should be able to project a 3D application involving geometrical manipulation and providing a more or less sophisticated rendering, possibly including non local illumination 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 take cares of the underlying details. Therefore, after the lessons on geometric transformation the student can be asked to implement a virtual manipulator (like a trackball) without necessarily knowing much about lighting and rasterization, and so on.

It is a common pratique to organize all the exercises in a single effort to implement some kind of application. Previous work how a computer game can be used to teach also other fields of Computer Science: object-oriented programming [CC07, CC05], pattern design [Ges07] and CG itself [HS05] to cite a few.

We implemented a multiplayer game called *EnVyMyCar*, a car racing game, where the visualization of the entire scene is a placeholder and must be developed by the student. The framework is designed so that the student does not need to take care of the networking issues or with the physics (although this might be included in a more specialized course) but only to understand the very simple C++ classes describing the scene and render it interactively. The description of the scene is minimal and only concerns the parts that can physically influence the race, i.e. the streets, the buildings, the tree (others can be easily added) and not at all with the appearance of things, which give to everyone total freedom

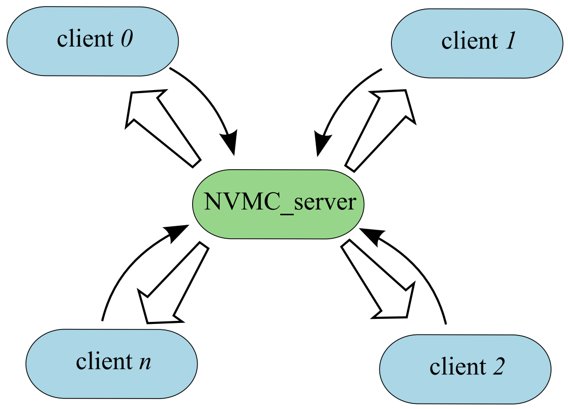

Figure 1: *Scheme of the NVMC framework.*

on how to represent their car, the terrain, the sky etc. It is quite straightforward to see that a racing game is a perfect scenario for progressively mapping CG concepts to code: the geometric transformations (the front wheels that rotate and steer), the use of impostors (the billboards for the trees), the environment mapping (the dynamic reflection on the car) etc. Alleviating the student from non CG problems was not the only reason for EnVyMyCar. We also wanted them to share knowledge and discuss problems in the same platform and possibly write code so modular that is could be moved from one project to another (instead of the thousands-lines long *main* body). This is also what we ironically termed "the envy factor": to see that your classmate makes billboards more convincing than yours.

In the next Section we give a detailed description of the EnvyMyCar framework. In the third Section we explain how several Computer Graphics techniques can be fit into the project to demonstrate that this kind of approach can be a very useful training for both basic and advanced Computer Graphics topics. The results of the use of the framework as an educational tool are reported in Section 4 and the conclusions are outlined in Section 5.

2. EnvyMyCar: the framework

NVMC is a car racing multiplayer game realized with a single server - multiple clients architecture shown in Figure 1. The scene is composed by a static part (the circuit, the trees, the buildings etc.) which is permanently stored both by the clients and by the server and a dynamic part, i.e. the players' cars and their state. We refer to the *state* of a car to indicate its position and orientation, its speed and other information that can be contained in few bytes.

The state of the race (i.e. the collection of the states of the single cars) is updated by the server, which runs the race simulation, and broadcasted to all the connected clients. The clients may send messages to the server (i.e. INCREASE_SPEED, STEER_LEFT, BRAKE) which are processed and accounted in the simulation. The communication is asynchronous, meaning that messages are sent independently from each client and from the server.

So far, this is 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 just broadcasted to all the other clients. A function *ShowSnapshots* is implemented in the client side which takes care of showing the snapshots of all the other clients.

In other words, every time a client is launched, the developer may see also snapshots from the other connected clients. This is what we ironically named “the envy factor”, alluding to the fact that seeing rendering effects on another clients that one was not able or did not think to implement may cause a feeling of envy and a wish to improve her own client. Of course it was not envy but the curiosity and the wish to obtain a visually pleasant result what moved the students to implement new features following each other ideas. Note that this is very different from comparing the students' work at the exam or at fixed milestones. On the contrary, it is like forming a developers team where each student may develop her own version.

There are other ways to organize projects in teams: students may gather in groups of 2 or 3 and develop a project but then often happens that the contribution to the students of the same group to their project must be figured out at oral examination, while little feedback is given during the development of the project. Consequentially, 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 of insights on others (for example knowing everything about normal mapping but never trying 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, trick and code snippets, (as long as each one is able to explain clearly every line that appears in her code) but eventually everyone will have faced all the difficulties of the development.
The teacher may be connected to the server with her 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 everyone did, to associate the event to a key. This mechanism may also be used by the teacher to provide suggestions to the class, by implementing her own client (supposedly better than all the others) and uploading snapshots.

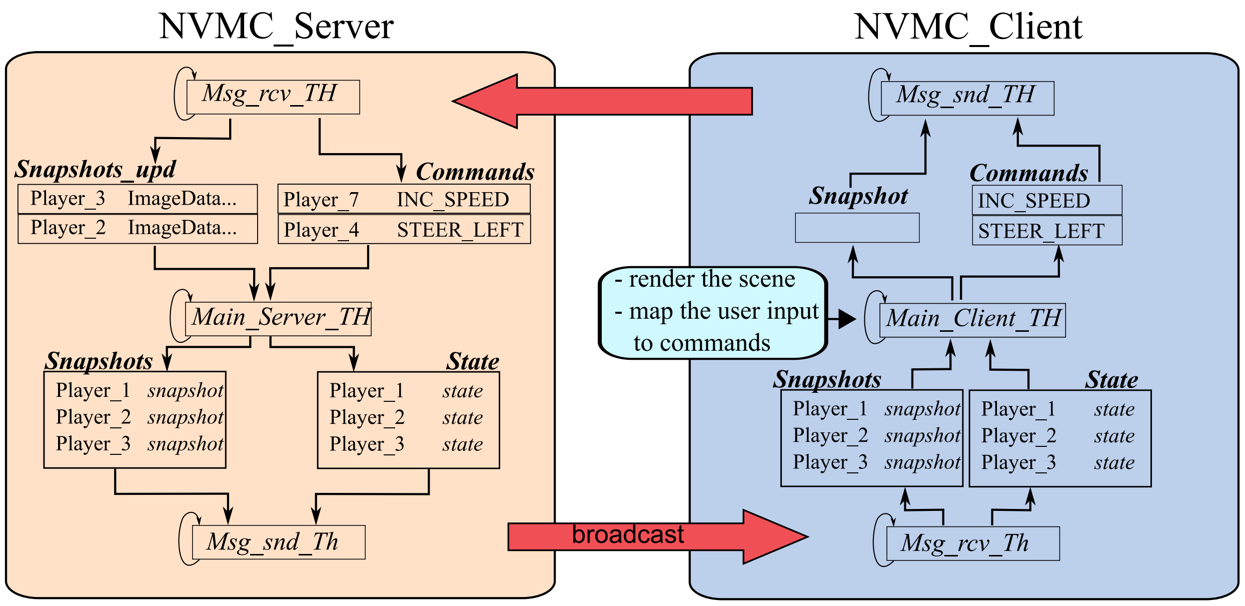

Figure 2: *Software architecture of the NVMC framework.*

2.1. Implementation

Figure 2 shows the structure of NVMC. The boxes with returning arrow represent process threads, the boxes named in bold are queues and the arrows are directed as the information flow.

Main_Client_Th is the main thread of the client and it is responsible for rendering the scene, writing commands to be sent to the server in the ***Commands*** queue or saving a copy of the current view in the *snapshot* memory area. The thread *Msg_snd_Th* reads the commands from the queue ***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 *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 ***Snapshots*** area where the most recent snapshot received from each client is stored. The *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 ***State***. Furthermore, it updates the ***Snapshots*** area with the snapshot received in ***Snapshots_upd*** . *Msg_snd_Th* continuously broadcasts the state of the race ***State*** to all the clients and the updating of the snapshot when necessary. Back in the client side, the thread ***Msg_rcv_th*** receives the messages from the server and copy them to the ***Snapshots_c*** and ***State_c*** area, where they Will be read from *Main_Server_Th*.

The NVMC framework also provides a standalone mode in which the server threads are launched within the same process (the client). This can be vary handy when the student works at home and does not want to launch a separate process.

2.2. Interfaces towards the developer

The goal of the students was to implement their own rendering engine for the game without necessarily to know all the underlying architecture, so we defined a simple software interface. Basically the developer only needs to know the definition of few classes: *Circuit*, *Car*, *Building* etc., to be able to draw them, and to use two functions: *Command(command_name)* to issue a command to the server, *UpdateScene()* which update all the dynamic data structures.

All the code is written in ANSI c++ using PTypes [Mel], a GPL library supporting multithreading and friendly wrap towards IP communication. Whe found in SDL [sdl], a practical choice to handle user commands through mouse/keyboard and windowing, althought SDL does not have to do with the NVMC framework and others libraries could be equally used (for example **QT** of **glut**).

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 (just to pick two examples) and then perform the conversion 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,

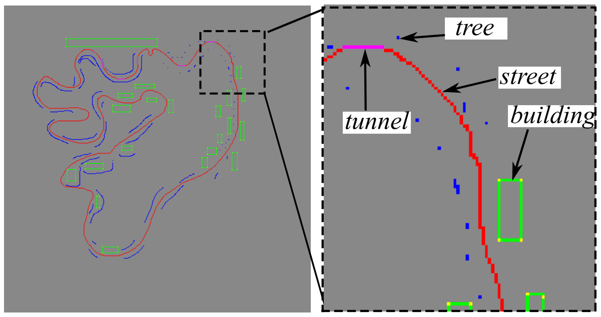

Figure 3: *A simple example of circuit encoded in a bitmap image.*

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 3 shows an example of an image coding a scenario.

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

The time to describe the framework and its libraries could vary but typically it needs few hours. In fact, the students need to know only the information to develop the client and not a detailed description of the entire framework and its associated libraries.

Depending on the students' previous examination some additional lessons need to be provided. For example, it could happen that most student have object-oriented programming background but few or no knowledge of C++. In fact, in most OOP courses Java is used to illustrate principles instead of C++. Hence, sometimes two-three lessons to integrate specific lacks are necessary. After this preliminary lessons the students start to develop their clients at the laboratory and at home. The development is done in parallel during the course. In this way the theoretic concepts can be put in practice during its learning. The students are assisted in any development problems that can encounter during the hours of practice in the laboratory.

One of the strong points of the EnvyMyCar is that during the development of the client the students deal with a lot of different Computer Graphics aspects in a natural way. In other words to develop a client a student has to solve a series of CG exercises before to reach an acceptable result.

3.1. Basic CG exercises for basic NVMC functionalities

NVMC, and more in general a car racing game, provided a natural mapping from the first steps in practical CG to the very basic functionalities of the game.

Geometrical transformations: One of the first problems that a student encounter is handling basic geometric transformation correctly. Such transformation 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. Moreover, he student can use car models composed by several parts and moving such parts in order to produce a more realistic animation of the cars. This is a more "advanced" use of geometrical transformations.

Lighting: About lighting, at the beginning, the common practice is to setup the basic Phong illumination model that OpenGL provides.

Texturing: Concerning texturing, in practice, all the students apply textures to buildings, terrain and road 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 needs to be textured with special care to keep the appearance consisted with the scale of the scene (100 meters large windows are not accepted).

What describes until now are the minimum features required to reach a sufficient score in the evaluation. At this point the students are encouraged to get a more high score by adding other features to their client. In this phase students are encouraged to be creative and found by themselves what to implement on the basis of what explained during the course. Some students opt for simple techniques moving its effort in creativity while others try to implement techniques more complex from an implementation viewpoint despite the final look of the client. Just to give an idea we report some examples implemented by our students specifying the techniques used, and for what purpose the student use it:

Billboarding: Billboarding is one of the image-based rendering technique shown to the student during the course. One of the typical example is the rendering of trees as oriented texture to replace geometry. Most students reimplement such example to have trees in theirs scenery 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.

Projective texture mapping One of the things that some students would like to see is the headlamp of the car that illuminates the road. To achieve this goal they have to deal with projective texture mapping and blending.

Cube mapping: A lot of students are not satisfied of the look of its scenery until they do not see a sky over the car and a landscape around the main road. Most students envisaged cube mapping to render a skybox.

Lighting Models: Student that wants to make practice with vertex or pixel shaders are suggested to implement a lighting model (e.g. Cook-Torrance, Oren-Nayar, Minnaert). One of the exercise during shaders lesson consist of show how to implement a per-pixel version of the standard Phong model.

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 impact with something (since the collision detection is not implemented in the framework this effect is usually enabled/disabled by the users). Other effects the student can add to their client with a particle system is atmospheric effect such as rain or snow.

Dynamic cube mapping One very pleasant thing is the addition of dynamic reflection to the car. This can be accomplished by rendering several time the scene on the face on the appropriate in order to create a dynamic cube map that can be used to render car's reflection.

Shadow mapping: Shadows add realism to the rendering and provide a professional look to the final rendered scenery. Our course not cope 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 suggested during the course as a simple alternative to the implementation with shaders.

Some screenshots of the rendering techniques implemented are shown and described in the Results section.

4. Results

The EnvyMyCar framework has been used in two Courses for now. The first one is the Computer Graphics Course of the University of Siena (10 alumni). The other one is the Advanced Computer Graphics Course of the University of Ferrara (14 alumni). For this second class this project is facultative, the student could also choose to do another project chosen from a list or not implement a project at all, in this last case a penalty on the final evaluation is applied. Nevertheless, 12 students choose to develop an EnvyMyCar client. In order to understand the effective utility of the EnvyMyCar framework as a tool to learn Computer Graphics after the examination the students have been interviewed (informally) in order to collect their opinions. Most of them enjoy the project and find it an useful learning tool. It has been noted that, although the few things of the framework to know are quite trivial and the students where provided with a first "hello world" fully working client still a written tutorial and a FAQ could have been useful, for example to those who missed some lessons.

Apart the satisfactory interview another factor that underline the effectiveness of the framework is the good results obtained by several students. Here, we show some screenshots of the developed clients, but it is interesting to analyze that about the 70% of the students have implemented more than the basic features necessary to reach a sufficient score (good camera handling/standard lighting/texture mapping/skybox). For example some of them use shaders to implement complex lighting models. Others insert particle systems to produce dust, fire, or other similar effects. Others improve the look of theirs car with dynamic reflections. Few of them generate procedurally some of the scene elements. Most deal with billboarding to render trees and projective texturing to draw headlights. Concerning the artwork aspect only very few students have contributed with own graphics.

5. Conclusion

In this paper a framework to support teaching of Computer Graphics at basic and advanced level has been presented. The effectiveness of the proposed framework as a learning tool has been demonstrate by the practical use in University courses and by the students' feedback. The use of the EnvyMyCar framework provides two basic advantages: strong motivation given by the "computer games effect" and personalized learning.

The only drawback concerns the student background that 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.

NVMC has been developed quite in a hurry in the first days of the course and it still lacking some important features, such as collision detection among cars. However, the most important improvement will be to write of a extended tutorial and manual for next year students and possibly incorporate the exercises as part of the tutorial, as a first step towards a concise textbook.

References

[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.

[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.

[Mel] Melikyan H.: C++ portable types library. More info on: `http://www.melikyan.com/ptypes/`.

[sdl] Simple directmedia library. More info on: `http://www.libsdl.org/`.

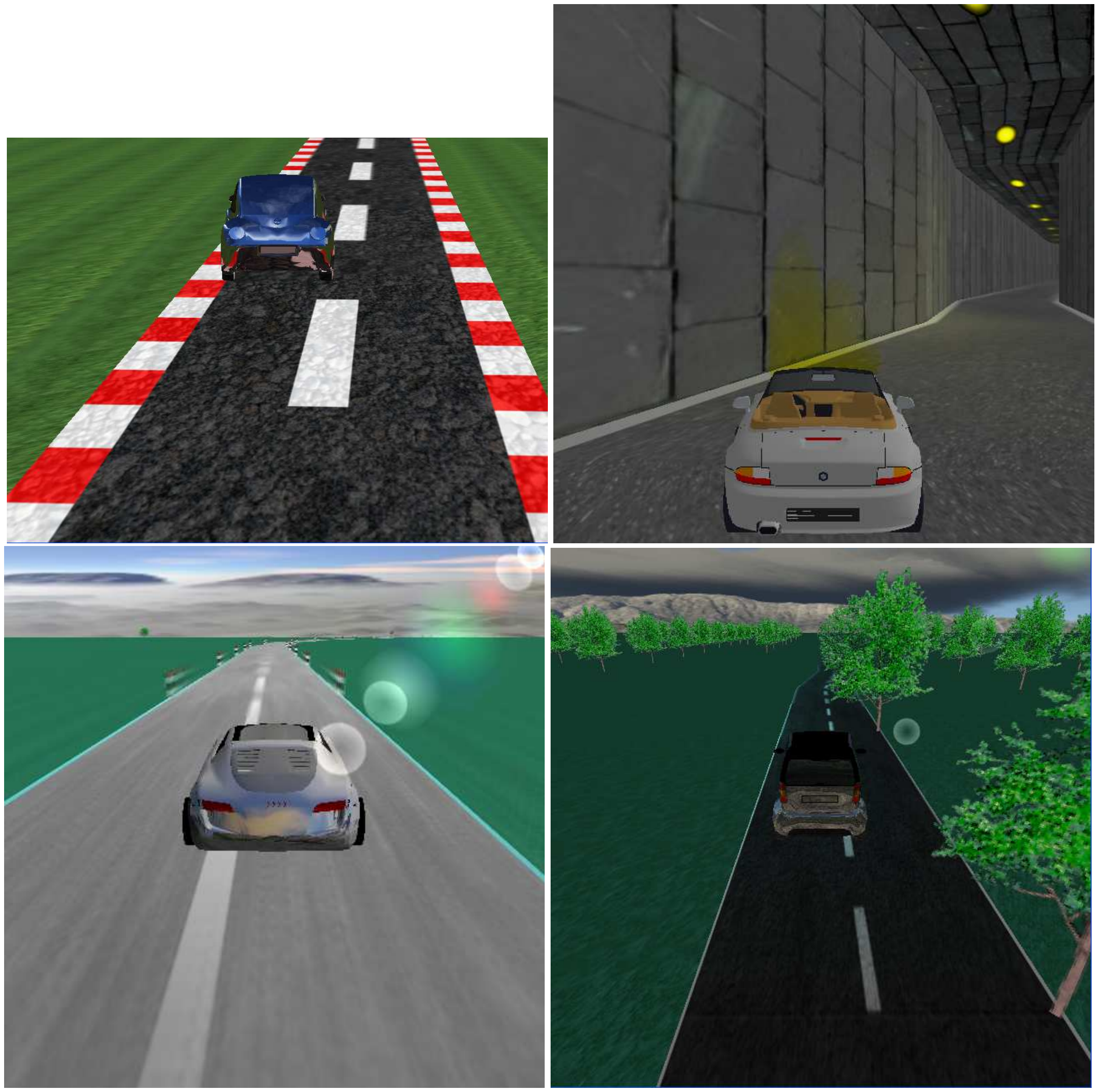

Figure 4: *Some clients developed by the students. (Top-Left) Dynamic reflections on the car. (Top-Right) Projective texture for headlights and tunnel lamps fakes with textures. (Bottom-Left) Motion blur and lens flare. (Bottom-Right) Billboarding for showing trees.*

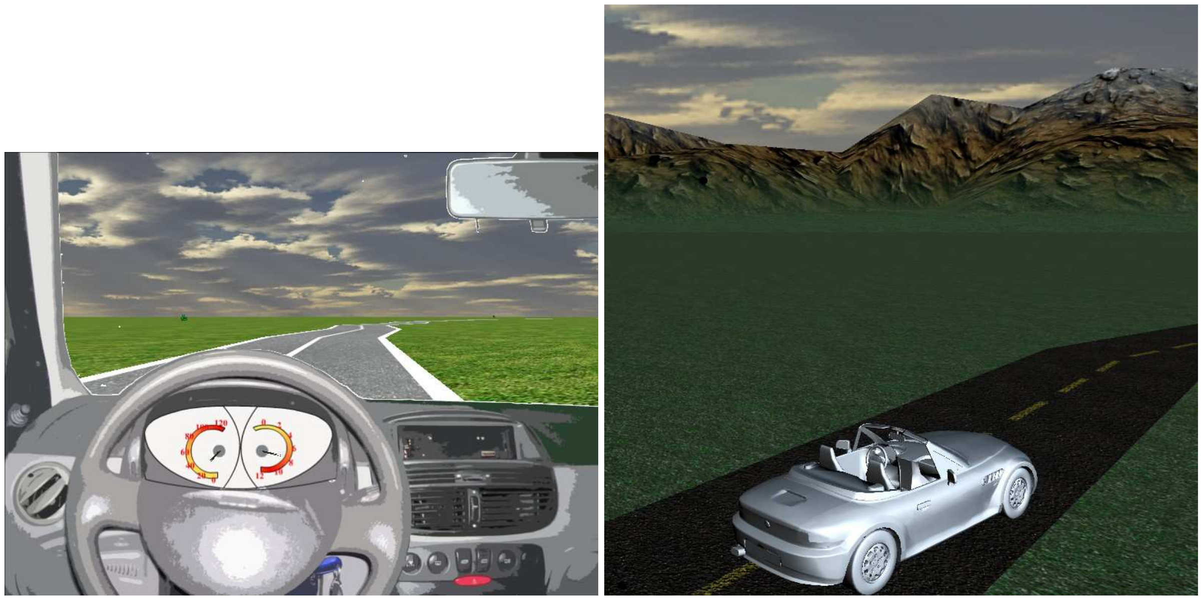

Figure 5: *More clients: (Left) Screen aligned billboarding for the view from the interior of the car (Right) The mountains are procedurally generated as a heightfield and rendered with a displacement map assigning a height-dependent texture (grass, mountain, snow).*

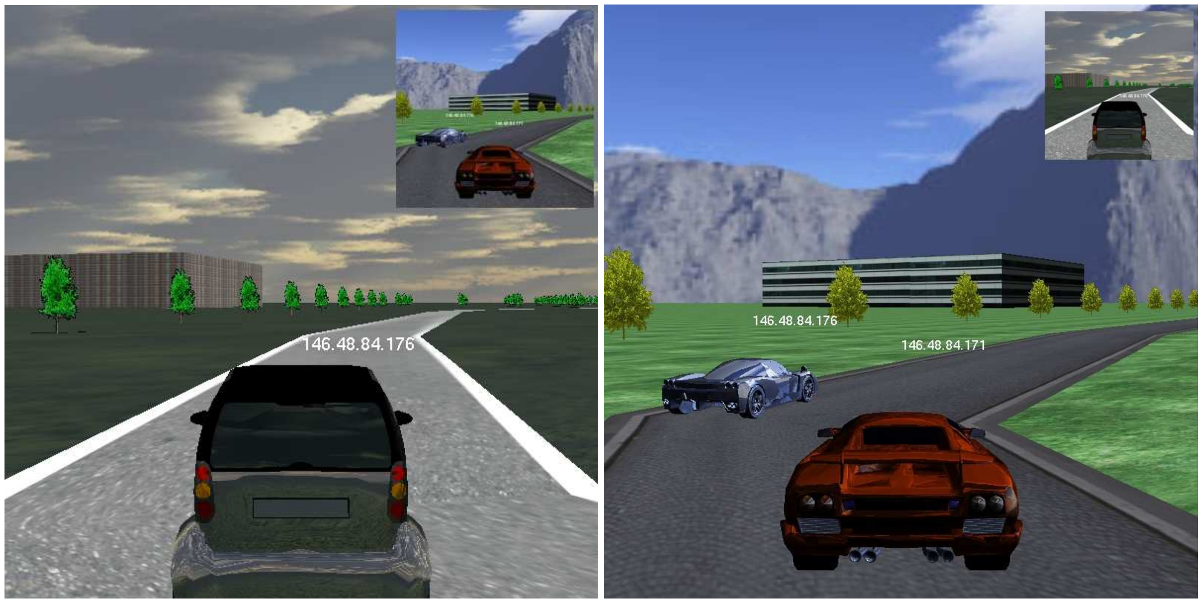

Figure 6: *Two player connected during the race. Note that each client provides its own view of the scenario, including the appearance of the cars.*