REYKJAVÍK UNIVERSITY
HÁSKÓLINN Í REYKJAVÍK

CarDriver – Using Python and Panda3D to construct a Virtual Environment for Teaching Driving

Verena Henrich, Timo Reuter

RUTR-CS08003 — May 2008

Reykjavík University - School of Computer Science

Technical Report

ISSN 1670-5777

CarDriver – Using Python and Panda3D to construct a Virtual Environment for Teaching Driving

Verena Henrich*, Timo Reuter*

Technical Report RUTR-CS08003, May 2008

Abstract: *CarDriver* is an educational virtual environment for safer driving. It is a game where you are the driver of a car and have to care about the traffic rules. While playing, you may subconsciously become a safer driver. In order to encourage the players along, several design issues are considered, e.g. social presence and flow. CarDriver is implemented in Python and Panda3D. A simple data-driven creation of the world is presented, which affords an easy creation of different levels. The physical simulation, which is based on the *Open Dynamics Engine* [Smi08], gets particular attention, as it is the heart of CarDriver. There were heavy performance issues, which were analysed and dealt with. Our solutions for the performance issues are presented. The performance load of the physical simulation lead to the idea and implementation of a physical *Level of Detail*. We also explain the usage of *Finite State Machines* ([Tea08c], II-N) in CarDriver.

The source code is freely available under the GNU Public License (Version 3) [FSF08] on SourceForge [H08].

Keywords: Educational Virtual Environment in Panda3D; Python Open Dynamics Engine (ODE); Physical car simulation; Performance optimization in Panda3D; (Physical) Level of Detail; Networking in Panda3D; Finite State Machine.

(Útdráttur: næsta síða)

* Reykjavík University, Kringlan 1, IS-103 Reykjavík, Iceland. verenah08/timo08@ru.is

HÁSKÓLINN Í REYKJAVÍK
REYKJAVÍK UNIVERSITY

CarDriver – Smíði sýndarumhverfis til akturskennslu með Python og Panda3D

Verena Henrich, Timo Reuter

Tækniskýrsla RUTR-CS08003, Maí 2008

Útdráttur: *CarDriver* er sýndarumhverfi sem kennir fólki að aka varlegar. Þetta er leikur þar sem ökumaður bifreiðar verður að taka tillit til umferðareglna til að komast áfram. Með því að spila leikinn rétt, er möguleiki að undirmeðvitundin læri varlegri akstur. Nokkur tiltekin hönnunarmarkmið voru höfð að leiðarljósi til að hvetja leikmenn til lengri setu við leikinn, t.d. félagslegur þáttur og flæði. CarDriver leikurinn var skrifaður í Python og Panda3D. Einföld gagnadrifin aðferð til að búa til nýtt akstursumhverfi með skjótum hætti er kynnt í þessari skýrslu. Eðlisfræðivélin *Open Dynamics Engine* [Smi08] fær sérstaka umfjöllun þar sem nún gegnir veigamiklu hlutverki í CarDriver. Greina þurfti afköst þessarar vélar til hlýtar og tekist var á við ýmis vandamál sem stóðu góðum afköstum fyrir þrifum. Þessi skýrsla greinir meðal annars frá lausnum okkar á lélegri afkastagetu eðlisfræðivélarinnar. Hið mikla álag á hana leiddi til dæmis til hugmyndar og útfærslu sem við köllum eðlisfræðilegt *Level of Detail*. Í þessari skýrslu skýrum við einnig notkun endanlegra stöðuvéla ([Tea08c], II-N) í CarDriver.

Forritakóði CarDriver er aðgengilegur á SourceForge [H08] til frjálsar notkunar samkvæmt skilmálum þriðju útgáfu almenna GNU leyfisins [FSF08].

Lykilorð: Panda3D sýndarumhverfi til kennslu; Python Open Dynamics Engine (ODE); Bílhermir með eðlisfræðivél; Afkastabestun í Panda3D; Eðlisfræðilegt Level of Detail; Nettengt Panda3D; Endanleg stöðuvél.

(Abstract: previous page)

Contents

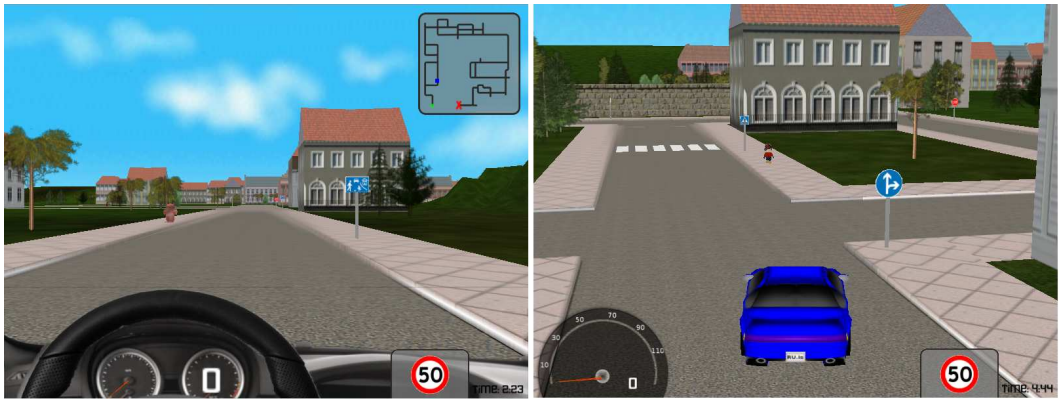

Figure 1: A 1st person (left) and 3rd person (right) view from CarDriver.

1 Introduction and Motivation

Nowadays, many people are driving a car or will apply for a driver's license. For the ones who want to apply for a driver's license, they do not know yet all the traffic rules. But also the other people already driving have problems regarding the traffic rules, because they become careless, or new rules could be defined which they are not aware of. The motivation for this project is to make people more sensible and prepare them for dangerous situations using a virtual environment. They should refresh their knowledge about traffic rules and get familiar with new circumstances. The main goal of this virtual environment game, we call CarDriver, is thereforesafer driving.

To achieve this goal the players of CarDriver drive a car from place A to another place B. On their way they have to care about traffic and traffic rules. If a rule is violated or another mistake is made, the game will give a corresponding remark. If the mistake is severe, the driver is stopped and a little embedded game, related to the mistake, needs to be solved. The purpose of these games is educational and the engaging 3D environment along with an entertaining challenge should provide a better retention of the traffic rules.

To make the virtual environment interesting there are several factors which need to be considered. First, the game needs to be as realistic as possible and the player should have the feeling of physical presence [GR03]. This is provided by a physics engine together with the ability to steer the car without restrictions. Second, the flow of the game needs to be as intuitive as possible to keep people playing. Therefore a graphical user interface provides indirect information about the actual situation. Third, it includes a network mode where people can play together. This can provide an element of competition as well as a feeling of social presence [GR03].

This paper starts with a related work section, where comparable Virtual Environments are introduced. The approach is described in several subsections. It begins with the description of the game flow and explains the creation of the world. The fundamental topic of the physics engine is covered in the section about the Open Dynamics Engine [Smi08]. The concept of Finite State Machines ([Tea08c], II-N) and their usage in CarDriver follows. As another main topic of this paper there is a section about performance tuning. Several possibilities to optimize the Virtual Environment are regarded and corresponding solutions given. The main solution how to achieve a better performance in CarDriver is presented as physical Level of Detail. Finally, the networking functionality is described.

2 Related Work

As a desktop game with an educational purpose in the traffic domain, there are surprisingly few comparable games. This approach is mainly comparable to the German software *3D-Fahrschule* [Bes08]. It also provides a virtual world where you can drive around. In this program, a guide will help to prepare for the driver's license. The guide is situated in an advice box. In this box the player gets information about where they should drive or if they have made a mistake. The system will count the mistakes done by the driver and if there are too many mistakes – all have a different weight – the game is stopped. Here, the flow is broken and a dialog box appears to ask whether the user will quit the actual game or restart at the last given advice.

Another game is *Taxi Driving School* [Gam08]. It is based on Flash and playable in a web browser. The game provides different lessons before a driver can apply for a final test. This approach uses 2D graphics in a top-down view. The driving is actually a bit difficult and the user gets no experience of sitting in the car. The game is also less educational. It should be considered more as a fun game.

A lot of other games in the area of car racing games use perspectives similar to CarDriver. The user is also in the situation of being the driver and the car is behaving mostly according to physical laws. These games have no educational purpose at all and in fact they may make the players' real world driving worse!

3 Approach

CarDriver is implemented in Panda3D [Tea08b] using Python [Fou08]. Furthermore the Open Dynamics Engine [Smi08] is used to simulate the physical behavior of the car in the world. The player of the game has the ability to drive from one place to another. This path is defined by drawing a picture of the roads. The prototype already supports mostly physically correct driving and integrates a time based change of weather, the onset of fog. The player can change the camera position to the first or third person view. It is also possible to see the car from the side.

This section gives an overview of all technologies which are used throughout the implementation of the virtual environment.

3.1 Flow and the GUI

To ensure that the flow in the game is not broken and the user enjoys playing, different things are implemented in the programme. When the game is started, an info screen in front of the main game tells the user about his options regarding the goal and the possible key strokes. This screen is faded out with an *Interval* to let the user know where he is.

The game starts in the first person view, as shown in the left screenshot of figure 1. A *Heads-Up Display (HUD)* shows a dashboard and a steering wheel. In the upper right corner is a map, where the player can find his own position and the finish line. There is a smaller HUD in the third person view, see right side of figure 1. Both HUDs provide status information about the actual situation. The steering wheel in the first person perspective moves to the left or right depending on the steering. Furthermore the actual speed is shown in both perspectives. On the lower right side the player finds information about the actual speed limit. If a mistake is made, like passing a forbidden entrace sign or a violation of the actual speed limit, the background begins to flash, warning the user to slow down. This gives the player the ability to correct his behavior. It avoids irritating player by not popping up an embedded game until he accidently drives too fast for 2 seconds. The flashing is made by a *Color Interpolator Interval*. It changes the color to red and back to the original white while the opacity is not touched.

If an embedded game is triggered because of violating the speed limit, a stop sign or a red traffic light, the main game stops and the embedded game is faded in. The background of the actual position in the main game in still visible to assure the player that he has not broken the game. All information screens are drawn in the same style to be easily recognized.

To give the user a better feeling for the car, the camera is connected to the car in the first person view. In the third person view the height is not influenced by the car to avoid a shackle effect when the car is rolling over a sidewalk.

3.2 Dynamic Creation of the World

The world is dynamically created from an image. Therefore, the *Python Image Library* [AB08] is used. The creation of the world uses different pixel patterns, which fall into three categories. The first series of patterns are represented in 3x3 blocks of pixels and only contain black pixels, as in figure 2 a) and b). These are used to build the different types of streets, e. g. figure 2 a) represents a straight road and b) a curve. The second series, the colored pixels within the 3x3 pixel blocks of the streets symbolize street elements such as signs, traffic lights, parking cars, or hydrants. Figure 2 c) shows a t-junction with one stop sign. Depending on the pixel position, a different position and orientation with respect to the street is set. Depending on the sign, the appropriate Panda3D collision solids are added, so that the corresponding rules can be applied when the user is passing by a sign. A traffic light has collision solids depending on its actual state. The third series, 3x3 pixel

blocks containing a cyan pixel in the middle, define houses. Depending on the two other neighbouring pixels the type of the house and its orientation is defined. An example is seen in figure 2 d).

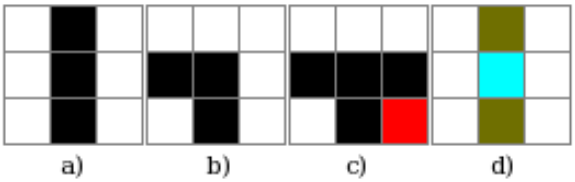

Figure 2: Pixel codes for world elements.

To achieve this functionality, a lot of preparing for the street elements and houses is needed. The main problem is to make them fit together. The objects were modelled to occupy exactly one or more squares. If an object takes more than one square, the upper most and left 3x3 block is used to define the whole object.

Some objects like trees are too small to be defined in the 3x3 pixel blocks. In this prototype they are set by an array which was created by driving around and planting trees. The type of trees was chosen randomly. The same thing applies to other objects which were not prepared to fit into the squares.

3.3 Open Dynamics Engine

One of the main aspects of this approach is the use of a physical engine. In this case the Open Dynamics Engine, (ODE) [Smi08] with its Python bindings, is used to simulate the physical movements of the car in the game. ODE is a very exact physics engine and a system can become unstable very quickly if it contains contradictional contraints [Sou08a].

In addition to the graphical representation in Panda3D, a new dynamics environment, where the physical simulation takes place, is defined. One of the constraints for a realistic behavior is the factor of gravitation which is set to -9.81. To use collision, there needs to be a special*physical space* environment. For the physical simulation two kinds of information is needed: *masses* and *geoms*. A *mass* is needed to know the density and weight of an object. The *geom* provides necessary information about the outer contraints of an object for the use of collision detection.

Due to the fact that there is a difference between how Panda3D and PyODE [Sou08b] represent the positioning and orienation of objects, a function is needed which converts between the two worlds.

The first step is to model the world physically. To prevent objects from just falling down, a plane geometry is created at the heigth 0.0. As described above, on each square of the world the appropriate geometries for the sidewalks, houses etc. built. All these objects are just geoms because they do not move and therefore do not need any masses. A mass would

just interfere with the physics simulation. Until now, all objects can be considered as stiff and unmovable objects which are necessary for collision detection.

The next step is to define all other objects like the traffic signs and the car. Due to performance problems there is a physical Level of Detail algorithm implemented, as described later.

The car needs to be modelled as it would be constructed in the real world to ensure a physically correct behavior. Therefore the car needs to have a chassis. This could be simplified by using a simple cube as its mass. This needs to be scaled to have the boundaries of a real car. After that, 4 wheel masses are defined as spheres. Both the car chassis and the wheels need geometry to be part of the collision simulation. Even though the geometry is defined exactly like the masses, it could be improved if the car shape is adjusted to fit the Panda3D shape. The use of spheres instead of cylinders for the wheels is recommended. If cylinders are used, the simulation can become unstable due to lack of precision. Next, a joint between the chassis and each wheel is required. ODE provides a joint called *hinge-2* ([Sou08a], Manual – Joint Types and Functions 1.3.5), see figure 3, which consists of two hinges in series with two different axes. The first axis allows the steering of the wheel. The other is used to rotate the wheel. When defining the joints, the positions of the objects to be connected must not interfere to avoid unwanted physical behavior. Furthermore the objects need to be oriented in the same direction. In this constellation the car is already ready to work but there are still some unwanted effects.

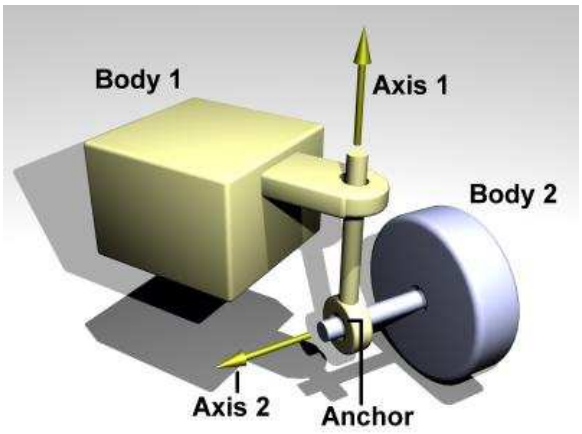

Figure 3: Hinge-2 [Sou08a].

The next step is to prevent the left/right movement of the rear wheels. This is done by setting the parameters for minimum and maximum angle of the hinge to 0.0 radians. For the front wheels the angle is also limited. In this case radians between 1.0 to -1.0 provide a good result. When the wheels are turning really fast and one attempts to turn at a corner, the wheels rotate off their proper constraints. This is caused by numerical errors. To fix it,

ODE provides a function to set the finite rotation mode ([Sou08a], FAQ 6.9), which needs to be enabled for the wheels.

Two factors are a must to ensure an appropriate physical behavior: *Constraint Force Mixing (CRM)* and *Error Reducing Parameter (ERP)* ([Sou08a], Manual – Concepts 7.2). The first parameter allow definition of the stiffness or hardness of an object. If it is set to 0.0 the object is totally hard and there can be no interference at all. If it is set to something positive the constraint can be violated. The factor should be set really low because the chassis should be very stiff. There is a warning about setting it to 0.0 because it is also good to reduce numerical errors if objects are defined more soft. The ERP defines the amount of errors reduced in each step of the physical simulation. The higher it is set, the more errors are fixed during the next step. The error can occur when one object, which is connected with joints to another, is moved without moving the connected object. In CarDriver this is the case when the front wheels are turning and moving and the the car chassis and the back wheels need to change position, too. Another reason why position does not match is when numerical errors occur. The higher the ERP is set, the more force will be applied to the object to put it back to its position. For the simulation of the car, a relatively high rate is used to prevent a drift between the wheels and the body.

To move and steer the car, there have to be *motors* ([Sou08a], FAQ 6.6) for each axis of the hinge-2 joint. There is no option to define a certain angle for an axis, it can only be influenced correctly by the motor. A motor needs a velocity and a force. A fixed force is set for both, the steering axis and the rotation of the wheel. In the prototype there is no simulation for the different gears of a car. If different gears would be used, there would be a mechanism for changing the force at a certain speed.

A Panda3D task, which is called every frame ([Tea08c], II-J), is created. Its purpose is to simulate one step of the physical system and calling the speeding up and steering function. The step size has to be well considered. If it is lower, the simulation is more accurate, which is actually a good idea for CarDriver. The disadvantage is the higher consumption of CPU power. For CarDriver a step size of 0.65 is chosen. It is exact enough to keep the system stable. The arrow keys are accepted to increase or decrease the velocity factors. For left and right a certain amount is added or substracted from the radian on every frame the key is pressed. For the velocity of the wheel turning the same thing applies for the forward arrow key. While pressing backwards the car lowers the speed with the effect of braking.

While the car is on the street, the four wheels are colliding with the road elements. In every step a new contact joint is created. This joint gets some parameters to keep the car on the street. To simulate soft wheels a bounciness of 0.1 is appropriate. There is also a slip factor, which is 0.0 in CarDriver. For a simulation on a slippery road it should be set higher. It is kept low to make driving not too difficult.

3.4 Finite State Machines in CarDriver

There are several Finite State Machines (FSM) ([Tea08c], II-N) used in CarDriver, for instance for the traffic lights and in the embedded games. The traffic lights each embody a Finite State Machine with the states *Green*, *Yellow*, *Red*, and *Yellow-Red* in that order.

A next state is always entered after a defined time interval by using the do-later task of Panda3D ([Tea08c], II-J-1).

There are mainly two kinds of embedded games in CarDriver: A *reaction test* and a *driver's license test*. The reaction test also represents a traffic light with the same states as mentioned for traffic lights above. The big difference in the FSMs is the state order. In the reaction test there is no defined order for the traffic light states in the reaction test; these occur randomly. The do-later task again is used for switching to the next state.

The driver's license test represents an FSM with a state for each question. There are different questionnaires for this game, thus the states are individually initialized for each new game and do not always present the same questions. The order of states is just the order of questions. A next state is entered if a question is answered.

3.5 Performance Tuning

CarDriver uses several Panda3D tasks ([Tea08c], II-J-1), for instance for the physical simulation as described above. Each task is called by Panda every frame ([Tea08c], II-J). Thus the entire virtual environment depends on the frame rate, e.g. the behaviour of the car or the embedded reaction test. It is obvious that the frame rate is important for the game to be consistent on different CPUs and graphical devices so as not to cause weird effects. One solution could have been to require fast enough hardware. But as children, who perhaps have not the money for new technology, and office workers with ordinary PCs are target groups, this solution does not suffice. Thus, differences in the frame rate need to be avoided. There are many potential reasons for a low frame rate and heavy differences in the frame rate.

The following list pinpoints potential performance issues with appropriate solutions. Some common issues are raised by the Panda3D development team ([Tea08c], III-V). All points are considered in CarDriver and if the issue causes a performance problem or simply lowers performance a bit, the corresponding solutions are applied.

- Too many meshes – the models are chosen with this in mind.
- Too many state changes.
- Shader – are all removed.
- Particle systems – are all removed.
- Memory consuming textures – all textures are minimized.
- Collision detection – net used in Panda3D.
- Number of polygons – visual Level of Detail is installed ([Tea08a], LODNode).
- Printing messages on the console – no more such messages are printed after debugging.

- Reload the same model every time – load a copy of a model ([Tea08c], II-A-1) can load multiple copies of a particular model without re-reading from the disk each time.
- Too many nodes – there are intermediate nodes, which are added to *render* and have many child nodes. There is one node reparented to render which is parent of all road elements. Additionally, these and other nodes are flattened together to reduce the overall number of nodes.
- Mipmapping – this feature is deactivated.
- Workload of CPU and GPU – networking is one solution for this matter.
- Dynamic creation af the world – does not matter.
- Physical simulation – this is a big issue, see next section.

3.6 Physical Level of Detail

The physical simulation causes a heavy performance load. The simplest solution would be to use an alternative approach to a physical simulation, but this solution does not suffice, because the physical simulation is a basic part of CarDriver. Thus there needs to be another way to handle this problem.

Every model in the virtual environment consists of physical geoms, the houses, the sidewalks, and also all other street elements. It is very CPU-expensive to calculate this big amount of geoms at once. The solution in CarDriver is a *Level of Detail for the physical geoms*. Therefore, the current position of the car serves as the center of calculations. As described above, the world is composed of squares, one square for every street part.

The physical Level of Detail algorithm works as follows. At the beginning of the game, all physical geoms are initialized but immediately disabled. Disabled geoms are not calculated for the physical simulation, so they do not consume CPU power. Then, the position of the car, in particular the current square, where it is located, is retrieved. The physical geoms on this square and on all directly neighbouring squares, total of 9 spheres, are enabled. Enabled geoms are considered in the physical simulation. If the car moves to another square, there are new geoms, which are enabled. Additionally, there are also geoms, which can be disabled again, because the car is out of their range. This approach achieves a huge performance benefit.

3.7 Networking

To increase the fun of the game and keep people playing it, a network mode is added. To be able to play together with another person, one version needs to be a server and the other a client. Both are included in the same program and it can be switched by changing one variable. Both versions use the Panda3D datagramm protocol ([Tea08c], II-AA-1) to establish a connection between the players. There are two new tasks which are added to the task manager. One is polling for new connections and if one is found, the connection is

added to the connection list and the other is polling for new messages and is queuing them. Both tasks are set to have a high priority to ensure that network packages are sent and received as fast as possible. A connection between two computers is established if a client opens a TCP connection to the server. When the connection is established, the two systems can exchange datagrams.

There are some parameters to improve the network speed. Panda3D advices to send datagrams immediately instead of collecting them to groups and sending it as a bigger package, because this would delay the packages to much. The delay explicitly needs to be turned on to avoid the use of the Nagle's algorithm [Nag84]. Its purpose is to avoid flooding of the network with too many packages. All these features are not documented in the manual of Panda3D and the API reference.

After the connection has succeeded, both game instances build datagrams including the position and orientation of the car. These are sent to the other participant. The data is decoded and a second car model is positioned and oriented with the received data.

4 Results

CarDriver is already a well functioning game. The use of the physics engine ODE was not too easy but driving is possible. Its parameters need some more polishing and it needs to be more general to be independent of the frame rate of a certain computer. It was not possible to improve the steering to fit all computer speeds. The use of a physical Level of Detail gave the possibility to go on with the physics engine and some things were hard to achieve because PyODE does not support all functions of ODE and its documentation is not complete.

First tests with people playing the game showed great success of the networking functionality. It lacks some synchronization features but it is already a useful instrument to set people trying out the game. In general, people are enjoying CarDriver as a game, and along with the networked social presence, the potential here for an effective learning environment is high.

5 Future Work

CarDriver can be extended easily in different ways. Different levels with varying traffic situations can be added easily with further world images. The same technique can allow players to create their own maps. This is a good way to make the game more interesting for multiple plays, i.e. the player does not know all the paths ahead of time.

More traffic rules with additional embedded games can be added to extend the educational aspect.

A smoother change of the different camera positions can improve the flow of the game even more.

The behaviour of the car can be extended by adding the functionality of gearshift.

There should be more synchronization in the network mode, i.e. the game would start at the same time for all participants, a global time, and the statistical overview at the end would display all and not only the user statistics.

A formal user study needs to be conducted to evaluate how effective the system is for teaching the traffic rules.

6 Acknowledgements

We want to thank Prof. Dr. Hannes Högni Viljámsson who helped us to get in touch with Python, Panda3D and the Open Dynamic Engine in his course Virtual Environments.

We thank the people of the Panda3D forum for their numerous informations.

Furthermore we thank fellow students from Reykjavík University for letting us testing our program on their computers.

References

[AB08] Secret Labs AB. Python imaging library (pil). http://www.pythonware.com/products/pil, last visited: 2008-04-08.

[Bes08] H. Besier. 3d-fahrschule. Besier 3D-Edutainment Wiesbaden, http://www.3dfahrschule.de, last visited: 2008-04-08.

[Fou08] Python Software Foundation. Python programming language – official website. http://www.python.orgl, last visited: 2008-04-08.

[FSF08] Inc. Free Software Foundation. Gnu general public license - version 3, 29 june 2007. http://www.gnu.org/licenses/gpl-3.0.html, last visited: 2008-05-08.

[Gam08] GamePuma.com. Taxi driving school. http://www.gamepuma.com/skill-games/Taxi-Driving-School.html, last visited: 2008-04-08.

[GR03] W. A. Ijsselsteijn G. Riva, F. Davide. Being there: Concepts, effects and measurement of user presence in synthetic environments. Ios Press, Amsterdam, The Netherlands, http://www.vepsy.com/communication/book4/4_01RIVA.PDF, 2003.

[H08] V. Henrich and T. Reuter . Cardriver. http://car-driver.sourceforge.net, last visited: 2008-05-08.

[Nag84] J. Nagle. Rfc896 – congestion control in ip/tcp internetworks. http://rfc.net/rfc896.html, 1984.

[Smi08] R. Smith. Open dynamics engine. http://www.ode.org, last visited: 2008-04-08.

[Sou08a] Inc. SourceForge. Open dynamics engine – manual. http://opende.sourceforge.net/mediawiki-1.6.10/index.php/Manual, last visited: 2008-04-08.

[Sou08b] Inc. SourceForge. Pyode. http://pyode.sourceforge.net, last visited: 2008-04-08.

[Tea08a] The Panda3D Development Team. Panda3d – api reference: classes. Entertainment Technology Center, Carnegie Mellon University, http://panda3d.org/apiref.php?page=classes, last visited: 2008-04-08.

[Tea08b] The Panda3D Development Team. Panda3d – web site. Entertainment Technology Center, Carnegie Mellon University, http://www.panda3d.org, last visited: 2008-04-08.

[Tea08c] The Panda3D Development Team. Panda3d manual. Entertainment Technology Center, Carnegie Mellon University, http://panda3d.org/wiki/index.php/Main_Page, last visited: 2008-04-08.

REYKJAVÍK UNIVERSITY

HÁSKÓLINN Í REYKJAVÍK

School of Computer Science
Reykjavík University
Kringlan 1, IS-103 Reykjavík, Iceland
Tel: +354 599 6200
Fax: +354 599 6301
http://www.ru.is