PS2 Game Development under Linux

Christian Fackroth, Rico Kubitza, Lars Stockmann, Jens Holze, Roland Winkler, Niklas Röber and Maic Masuch

{nroeber|masuch}@isg.cs.uni-magdeburg.de

Games Group, Department of Simulation and Graphics, Otto-von-Guericke-University of Magdeburg, Germany

Abstract

We describe the development of a 3D action arcade game, Magnetic Masters, on the Playstation2 using solely the PS2 Linux Kit. Motivated by the idea whether or not it is possible to develop a 3D game using the PS2 Linux Kit, the development process soon became a struggle between the limitations caused by the Linux system and the PS2 hardware. Many challenging low-level problems had to be solved, in order to complete the initial goal.

Although, it is possible to develop a sophisticated 3D action game using the Linux Kit on the Playsation2, we encountered numerous difficulties. The technical deficiencies and limitations of the Linux Kit restrict the possibilities by a large scale. To our best knowledge, Magnetic Masters probably is the most advanced game existing for the PS2 Linux system.

1 Introduction

The idea for starting this project was to find out if it is possible to design and develop a 3D action game using only the Playstation2 and the Linux Kit. As the Sony PS2 middleware programm is available only to professional game developing companies, the PS2 in combination with a Linux Kit would present an interesting alternative in PS2 game development for educational facilities and independent developers.

Only for a brief period after the launch of the Linux Kit, Sony supported a PS2 Linux community and several developers, which were working on supplementary tools for the PS2 [6], [5]. At this time, Sony also initiated a rendering contest using the PS2 Linux Kit to show off the potential inherent in the Playstation2 Linux version. But the majority of these tech-demos were rather minimalistic and exhibited only very simple rendering techniques with just a few hundred polygons rendered in realtime. All demos that were available within the Kit as well as from the community websites, were far away from being considerable as starting point for game development. This seems a bit weird, as the Playstation2 is known as a game console with many impressive professional game titles. The only reference about using the PS2 at a more advanced level is reported by Fortuna et.al., who used the PS2 Linux Kit for teaching students game console programming [8]. Unfortunately, there are no results from this course in terms of games or game prototypes.

In this paper we describe our game development process for the PS2 using Sony's Linux Kit. Starting with 3D graphics experiments up to the final development of a game engine, that features a graphics system which is able to render scenes with up to 30.000 polygons with textures, transparency and up to 4 independent light sources interactively at 30 fps. Furthermore, this engine consist of a sound and physics subsystem, and a simple artificial intelligence, that drives the computer controlled opponent. With respect to all existing demos, our achievements are beyond anything else available, and the developed game is even challenging and fun to play. However, the way getting there was long and very demanding. We were confronted with rather unusual PC and console game development problems, which we will highlight during the discussion of the game engine.

The initial idea of the game is an advancement of the classic Space War game from Atari [1]. We wanted to project the game into the 3D realm and instead of orbiting around a sun, the combatants fight inside a huge arena, orbiting around a giant magnet. Everything in this arena, including the ships and the shoots, is affected by the magnetic field generated by the center magnet. The goal of the game is, similar to the original version, to shoot at the opponent and to earn money to outfit the own ship with additional armor, ammunition or a better jet engine. Once the game is won, the player can call himself *Magnetic Master*.

This work is organized as follows: The next section briefly reviews the hardware components of the PS2 and the Playstation2 Linux Kit. After this we present a brief overview of the game concept and discuss in the following sections the designed game engine with a special focus on the 3D graphics system. This section also covers details about the rendering of sound, the aiming mechanism, the game physics, and additionally, the modelling and design of the game. In the end we discuss the results achieved, and conclude with a summary of the game development possibilities using the PS2 Linux Kit.

2 The PS2 Linux Kit

The Playstation2 Linux Kit is an original Sony developed expansion pack for the PS2, that extends the game station by several hardware components and converts it into a fully functional Linux system. The additional hardware package consists of a 40 GB hard drive, a memory card, an USB mouse, a keyboard and a network adaptor. The kit comes along with a customized RedHat Linux distribution. The installation of the hardware, as well as the software is straightforward.

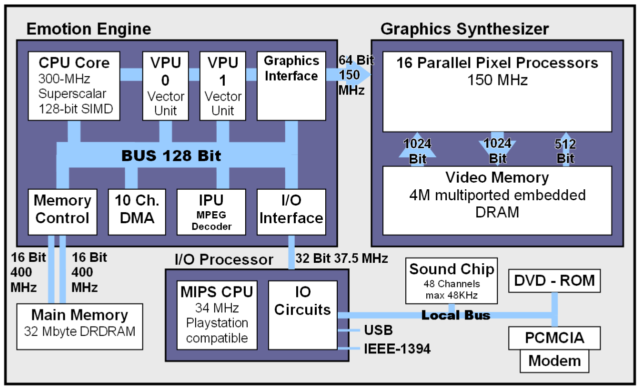


Figure 1. PS2 Block Diagram (after [7])

Although, the system has a similar look and feel compared to a Linux system on a standard PC, the underlying hardware is completely different, which makes the programming far more difficult. Figure 1 shows a diagram of the hardware components of the Playstation2. Interestingly, the hardware features two main processors, the 300 MHz Emotion Engine and the 34 MHz IO processor, that work both independent from each other. These two processors are connected via a bus system and can work together, but the necessary compilers are unfortunately not available within the Linux Kit. Furthermore, the only compiler available (GCC) was not able to produce PS2 optimized code. Additionally, there are two other processors that help to balance the work load: the Graphics Synthesizer (GS) and a 48 kHz sound chip. These two chips support the two main processors and are connected via a highspeed rambus system. Along the Linux Kit, Sony provides free references for the Emotion Engine, the Graphics Synthesizer and the DMA controller, but not for all the other components.

The Linux system itself is a customized RedHat Linux distribution that consists of the general Linux tools (GNU, KDE 1.0, Gnome, etc.) and some special libraries and programs that are necessary to develop applications for the PS2 architecture. The majority of the system is fairly outdated, and due to the minimalistic Linux system, many applications run very slow, especially in conjunction with a graphical user interface (KDE). Internet services, such as SAMBA and SSH can be used and have proven to be very useful, as it is possible to control the PS2 over the network. For programming purposes, the common compilers and libraries such as gcc, SDL and PS2GL are available.

Graphics programming with OpenGL is a bit more difficult on the PS2 using the Linux Kit. The rendering of standard OpenGL applications is extremely slow, as the entire rendering is performed in software, thus only very few polygons can be rendered at interactive rates. During the launch of the PS2 Linux Kit, Sony Entertainment initiated a rendering contest to demonstrate the capabilities of the game console [4]. The results of this rendering contest were very elementary (the winner was a demo with just about 2000 polygons). Further examples were found within the PS2 Linux communities [6], [5], but none of them used the true potential of the Playstation2, that is known from many professional developed game titles.

The graphic API of choice was soon to be found as PS2GL. This API had been specially developed by Sony Entertainment for the Linux Kit to demonstrate the 3D capabilities of the PS2 under Linux. In order to use PS2GL as the main rendering API, several changes on the operating system had to be made. The first requirement is a patched Linux Kernel, so that PS2GL is able to allocate and reserve a locked memory area for its own usage. Using the boot menu and a memory manager (PS2STUFF), it is possible to allocate up to 16 MB of main memory for the graphics subsystem. After several experiments we decided to use PS2GL for our own implementations for the following reasons:

- The syntax of the API is similar to OpenGL, and therefore makes it possible to evaluate parts of the 3D graphics system on other platforms.

- Nearly all relevant components of the Playstation2 (Vector Units and Graphics Synthesizer) are supported by PS2STUFF/PS2GL.

- The memory was reserved in advance and provided an upper limit for the necessary 3D data.

- It was the best solution within our time frame (6 month).

Besides the advantages of PS2GL, there are many drawbacks with this library. One of the biggest problems was that only a fraction of the known OpenGL instructions were supported. This caused several problems during the development of the graphics engine, as know and proven to work algorithms had to be redesigned in order to run on the PS2. Additionally, as the library was only available as alpha version, several major bugs have been found, of which some of them could be removed only with the assistance of Taylor Daniels, a developer working for Sony Entertainment and who initially designed PS2GL. Some of the bigger problems were a bug in the far-clipping algorithm and too slow keyboard response functions. As further clipping problems occurred, the model for the arena was carefully designed in order to minimize additional near-clipping problems, see also Figure 2 and Section 4.2.

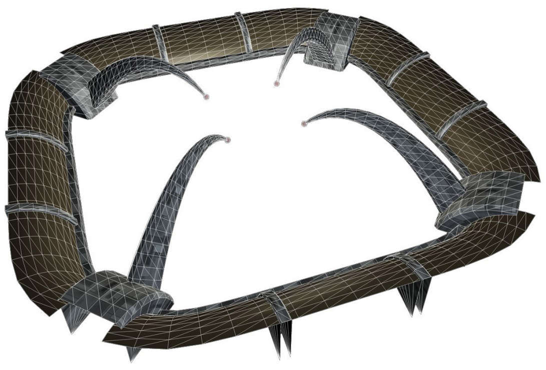


Figure 2. Arena - minimum Tessellation to avoid bad Clipping

3 Game Concept

Magnetic Masters was intended to be a classic arcade style game with fast action. The idea was to develop a fast arcade game that would benefit from the technical possibilities provided by the PS2. We decided to create a game similar to the classic 1978 Space Wars from Atari [1]. In this scenario, two space ships orbit around a planet and try to shoot each other. As the ships, as well as the shoots, are dependent on the g-forces from the center object, the shooting and the steering of the ships have proven to be very difficult. With the ambition to move the game into the 3D realm, we first experimented on a PC with the controlling of such ships in 3D space. As this turned out to be rather difficult, we decided on using a plane over which the 3D ships can fly. Although the game still is 3D, the steering and the physics are only 2D. To make the game more interesting, we changed the emptiness of space to an arena, the ships to hovercrafts and the center sun into a giant magnet.

The background story is about a future combat game, in which pilots act as gladiators and have to fight each other for the entertainment of a depraved society. As a tribute to our story we decided to let the player chose one out of four heros. The opponent ship is controlled by the computers artificial intelligence. The strength of the characters vary, and it is possible to play at different levels of difficulty, just by choosing a different character. Similar to many other games, the hired gladiator has to be equipped with weapons and armor in a shop-like system. Here, one can buy additional equipment such as armor, ammunition or tuned jet engines. It was initially planned to have many different items, but for sake of time, we limited this to a mini and a plasma gun at different levels, two types of armor and two different jet engines. The computer opponent has the same stock of weapons but has to chose a different character than the player. One idea for the shoots was that the weapons should not just kill the other player, but rather influence and change his physical attributes.

To win the game, one just needs to win all the duels to gain the rank of a *Magnetic Master*. We planned to let the player compete against more than one opponent in one level, but again time restrictions did not allow us to implement it, although, it would have been no problem for the implementation nor the performance of the PS2 system. Due to time constraints and difficulties in programming the PS2, many initially designed items had to be left out. Even though, the game is not as complete as planed, the result is still very impressive and many people assured us that they like playing the game. Especially the use of magnetism to distort the shoots is very challenging.

4 PS2 Game Development

After developing the game concept in theory, we started experimenting with the PS2 to find out the limitations as well as the potentials provided by the available hardware. To get an idea of the technical possibilities, we looked at several professional games and derived our own expectations:

- The graphics rendering should support at least 50.000 to 80.000 polygons with textures, transparency and additional special effects.
- The sound engine should be able to perform real time pitching and Doppler sound effects.
- The physics engine should be as realistic as possible and able to calculate elastic impacts with rotation transfer in 3-dimensional, magnetic fields.

- The computer controlled opponent (AI) should not just cheat, but behave as a realistic enemy by only using the same weapons, vehicles and physical parameters.

Soon, we had to realize that such a game could not be implemented with the Playstation2 Linux Kit. Due to many limitations, several adjustments to the original game idea had to be made, in order to develop a functional 3D arcade game. Although, some of the original ideas did not make it into the final implementation, the result is still impressive and fun to play. The following sections present an overview of the developed parts of the game engine as well as the design of the 3D models.

4.1 Game Engine

To drive the game, we designed and developed our own game engine within the the PS2 Linux Kit. This game engine consist of a graphics and sound subsystem, a simple physics engine to calculate the magnetic distortions as well as an aiming mechanism for the computer controlled opponent. The development was sometimes very difficult, as no additional PS2 hardware documentation (VU Assembler etc.) was available.

One of the main problems with the game engine was the exact time synchronization between the graphics and the sound engine. In fact, this problem turned out to be the most demanding and time consuming task of the entire project. As later described in Section 4.1.3, the game physics had to be synchronized to the sound hardware in order to use realtime sample pitching effects. Depending on the sound buffer size, an engine cycle was completed every 11 ms or every 21 ms. Unfortunately, PS2GL does not implement an equivalent of the $glutPostRedisplay()$ function used in the GLUT enhancements of OpenGL. It is not even possible to suspend the rendering process, to offer more complex rendering tasks additional time for their calculations. The PS2GL main loop function initiates the redrawing at no more than 60 Hz, which is the sync rate of the connected PAL/NTSC display device. Only if one reaches the limits, the frame rate will drop to 30 Hz. However, as we had no control of the rendering process at all, two major problems occurred:

- Rendering constantly consumes CPU cycles required by other tasks (AI, sound).
- The game runs completely desynchronized, which results in cycles painted twice, while others being dropped.

A solution to this problem by changing the main loop of the graphics rendering system has failed, and only an extensive parameter tuning could help a bit to keep the game at a stable average frame rate of 30 frames per second.

4.1.1 Graphics Engine

PS2GL works a bit different than the familiar OpenGL. Although, it supports an immediate rendering mode, the maximum possible polygon count for rendering at interactive rates using this technique is limited to just 600. Another in realtime applications often used rendering technique are display lists. But as only seven of these are supported by PS2GL, we decided to discard this technique as well. A more memory-efficient and faster techniques to display 3D data are vertex arrays. As these were fortunately supported by PS2GL, we have chosen this method in our own implementations for the 3D graphics engine. Nevertheless, several limitations still applied with the utilization of vertex arrays, as we found that PS2GL allows only 49 independent 3D-objects in one scene.

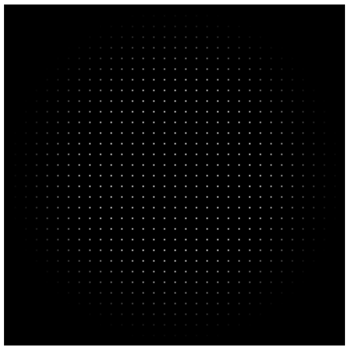
We also evaluated the possibilities of using textures within PS2GL. Although, the implementation is very efficient, some restrictions apply here as well. PS2GL only supports a maximum number of 147 different texture maps. Once the texture memory has been allocated assigned, it can not be reused by another texture. This is mainly due to an incomplete development of the PS2GL texture manager.

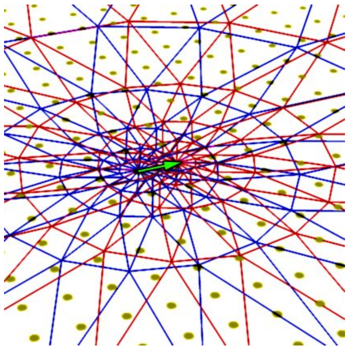
To overcome these limitations, we used batch-scripts, to divide the game into small subprograms in order to free up texture memory. We also found that the number of texture maps used reduced the maximum number of displayable polygons. Fortunately, the utilization of textures seems to have no influence on the overall performance at all. As it is common practice in game development, we decided to use textures wherever applicable, to reduce the number of polygons from our complex models [9].

With the decision on using vertex arrays, the graphics engine has been optimized for using this technology for the 3D rendering. The model data is stored externally to minimize compiling time and to maximize the loading efficiency. We developed a file conversion tool, that extracts all the necessary information from 3ds data files. The engine was optimized on this basis and able to display about 50.000 polygons with illumination and transparency effects at about 25 frames per second. This is also the maximum performance that can be achieved using the hardware limitations of the Playstation2 with the Linux Kit and PS2GL (according to Tyler Daniels) [3]. Interestingly enough, PS2GL does not support any 2D graphics, therefore all menus and the star wars intro in the beginning of the game, were designed as quads with overlayed textures.

4.1.2 Sound Engine

In order to create a compelling arcade game, good sound and music effects are as important as fancy 3D graphics. Fortunately, there was a good working multipurpose API for the PS2 present. SDL (Simple Direct Media Layer)


(a) Floor Texture.


(b) Floor - stepwise Movement.

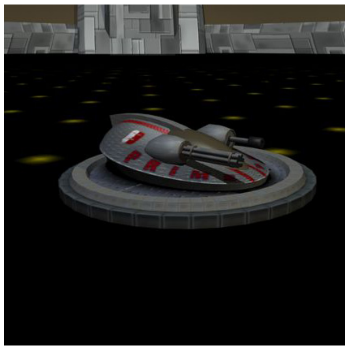

(c) Floor rendered.

Figure 3. Simulating a static Floor.

is freely available for various platforms and used in many multimedia applications [2]. From this library we only employed the functions for loading and the playback of sound. In comparison to PS2GL, SDL was post version 1.0, and thus very reliable.

The additional sound processing for Doppler effects requires sample pitching in real-time. This produced in the first experiments disturbing artifacts, which were caused by an insufficient hardware timer of the PS2. Therefore, we decided to use the sound hardware as game clock to control the game. The Playstation2 and the SDL sound API turned out to be quite powerful in playing sound data. It is no problem to keep the sound buffer at 512 bytes, which would produce disturbing buffer underruns on a standard PC. Using a sample rate of 48 kHz, the buffer has to be filled every 10.7 ms, therefore giving us a theoretical maximum of 94 game cycles per second.

4.1.3 Aiming Mechanism

From our initial design document, the AI was planned to be very flexible, giving the computer controlled opponent a dynamic behavior that would adequately react on the players input. Because it is difficult to define tactics in a nonlinear environment, we decided to limit the AI to good aiming algorithms only.

One of the main problems for computing the AM (aiming mechanism) was the limitation of processing time. Due to the point-based approach of our magnetic sources, the environment is neither static nor linear and is constantly changing. Thus it is not possible to determine the proper shooting directions analytically. In order to shoot in the right direction, the AM has to simulate many shots. Using several approximations it was possible to achieve very good results. The simulation runs with an increased magnetic field and an increased temporal speed. Using this approach, we were able to reduce the total number of steps and compute the shoot simulations distributed over three game cycles. Small imprecisions occur only very close to the center magnet. The final AM is efficient enough to compete and win against human opponents.

4.2 Modelling and Design

The 3D modelling of the arena, the hovercrafts and the weaponry were accomplished using Cinema4D. The models were textured and exported as 3ds files and converted into our own data format. In contrast to modelling for a standard PC game, many restrictions applied in designing the models with PS2GL back in mind. The first problem was the maximum allowed number of polygons, which was limited to about 30.000.

One of the bigger problems was a bug in the algorithm for the near clipping plane, which forced a minimum tessellation of the arena in order to draw all polygons (see also Figures 2 and 3). In this case a polygon is entirely clipped if only one vertex is outside the viewing frustum. To avoid this clipping behavior, the polygons need to be subdivided and designed with a higher tessellation. The floor of the arena was here the biggest problem. As the camera is always behind the hovercraft, constant clipping would occur. In order to circumvent the requirements for a high polygon tessellation, we had to come up with a trick to create the illusion of a resting floor, Figure 3.

This could be achieved by using an adaptive floor model, that is highly tessellated in the center and less at the outer perimeter, Figure 3(b). Assuming the camera to be always near the center of the floor, we used a rastered alpha texture to simulate atmospheric absorption effects, see Figure 3(c). Additionally, we prevented aliasing effect, which would occur with an infinitely tiled texture. During the game, the floor moves along with the hovercraft, not continuously but stepwise with the step size being the same as the distance in the raster of the texture, Figure 3(b).

5 Results

Game development on the PS2 using the Linux Kit was an extremely challenging task. In the end we were able to develop a game engine which renders 30.000 polygons with transparency and a maximum of 4 light sources at 30 frames per second. The sound engine is able to generate realtime pitching and Doppler effects in stereo with a sound quality of 48 kHz. The physics engine can calculate 3D impacts close to physical correctness, and the artificial aiming mechanism is hard enough to fight, without too heavy CPU-usage. Figures 4(a), 4(b) and 4(c) show different screenshots of the final game. The game itself is not to hard for beginners, but it needs some skill to win the entire battle. Because of the lack of time, we were not able to include all ideas from the initial game design document.

More information can be found on our website[1]. This includes additional screenshots and videos, as well as the source code and a demo of the game.

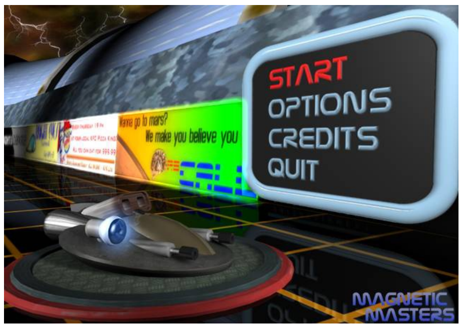
6 Conclusions

To sum it all up: Yes, game development using the PS2 Linux Kit is possible – but with many restrictions and not to the extend of professional PS2 games. Many restrictions apply, such as that not the entire hardware is supported and that many libraries are still in an early alpha stadium and not further developed.

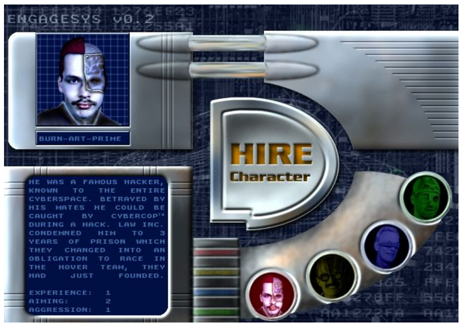
The results achieved, as discussed in the last section and throughout the paper, are remarkable, but they were only possible by using several elaborated tricks and with a lot of patience in experimenting. With Sony Entertainment having ceased the support of the Linux Kit and the Linux PS2 community meanwhile non-existent, we would recommend to use a different platform for game development. Another way to summarize our work is to quote the API of PS2GL: *This is not intended for game development*.

7 Acknowledgment

The authors would like to thank Tyler Daniels, the developer of PS2GL, who assisted us in many ways, and without whom this work would not have been possible.



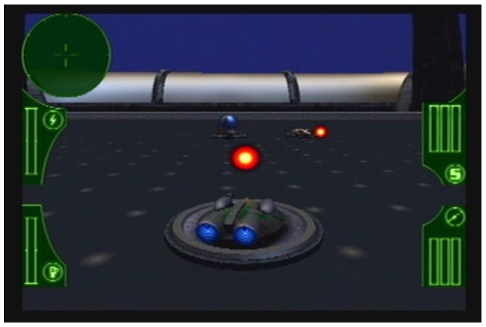
(a) Splash Screen.



HE WAS A FAMOUS HACKER, KNOWN TO THE ENTIRE CYBERSPACE. BETRAYED BY HIS MATES HE COULD BE CAUGHT BY "CYBERCOP" DURING A HACK. LAW INC. CONDEMNED HIM TO 3 YEARS OF PRISON WHICH THEY CHANGED INTO AN OBLIGATION TO RACE IN THE HOVER TEAM, THEY HAD JUST FOUNDED.

EXPERIENCE: 1
AIMING: 2
AGGRESSION: 1

(b) Ingame Player Menu.



(c) Ingame fight.

Figure 4. The final Game.

References

[1] Atari. SpaceWars, 1978.
[2] S. Community. SDL. Simple Direct Media Layer, 2004. `http://www.libsdl.org`.
[3] T. Daniels. eMail Communication, 2004.
[4] S. Entertainment. PS2 Linux Community - Rendering Contest, 2003. `https://playstation2-linux.com/projects/vudemocontest/`.
[5] S. Entertainment. PS2 Linux Community, 2004. `http://ps2dev.org/`.
[6] S. Entertainment. PS2GL Development Community, 2004. `http://ps2gl.playstation2-linux.com/`.
[7] R. Green. Procedural Rendering on Playstation2, 2001. `http://www.gamasutra.com/gdce/2001/green/green_01.shtml`.
[8] Henry S. Fortuna. Teaching Console Game Programming with the Sony Playstation2 Linux Kit. In *CGAIDE - 5th GameOn International Conference*, pages 365–369, Reading, London, UK, 2004.
[9] J. Watt and F. Policarpos. *3D Games: Real-Time Rendering and Software Technology*. Addision Wesley, 2002.

[1] `http://yodahome.de/magnetic/`