Game Programming Laboratory --- 251-257-00L

Final Report

Jens Puwein

Pascal Rota

22.06.2007

Table of Contents

Table of Figures

Introduction

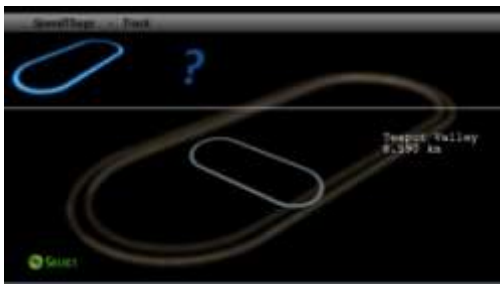
SpeedThugs is a futuristic high speed racing game. The player drives a hovercraft vehicle on a track that is located somewhere in outer space and tries to finish the track as fast as possible. Two tracks can be chosen. One has a simple oval shape, the other includes loopings, twists and bumps. Slowdown areas beneath the track boundaries make the vehicle slower. Therefore it is wise to stay on the inner area of the track. Walls at both sides make sure the vehicle stays on the track and doesn't get lost in space.

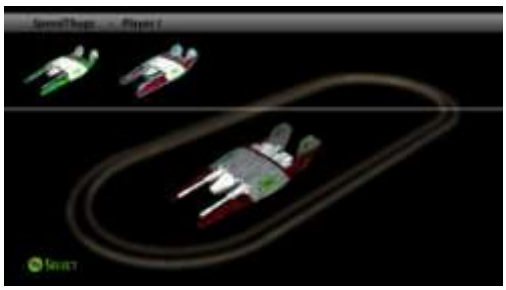
The controlling system is simple and adapted from most existing racing games. Besides brakes and acceleration the only thing that is needed is nitro. Nitro allows to drive and accelerate much faster. It is depleted when it is used but it is refilled every new lap.

The game modes featured are “Time Attack”, where you race against the clock, and “Two Player”, where you race against another human player in a split screen. A highscore system tracks the fastest laps and motivates further time improvements.

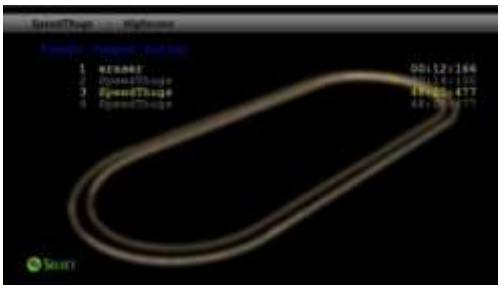
The main focus of the game is on speed. To give the player the impression of driving very fast motion blur and tunnel vision are used. The second track has a shape similar to a roller coaster. Together with the high speed this should be a fun ride.

In game Images










Design

Initial Idea/Project Proposal

Game Description

SpeedThugs is a high-speed racing game with hovercraft-like vehicles. As an exemplary game you might think of the WipEout series, the F-Zero series or the Extreme-G series, but only in the wide sense.

Detail

Game Modes

The goal of the game is to finish the track as fast as possible or crossing the finish line in first place, depending on the game mode. If you play in the time attack mode, you have to finish the track as fast as possible without exceeding a certain time limit. Single player race lets you drive against computer guided enemy vehicles and in the two player mode you can race against your friends. The possibility to tune vehicles in terms of their properties would be even more fun. Good results in the time attack mode let you unlock better vehicles, giving you the chance to master harder time attack challenges and again get a better vehicle...

The Track

We intend to make one track with curves, jumps and possibly movable obstacle. The track should be a challenge, including shortcuts and alternative ways, where shortcuts are harder to make than the normal route or may only be reached by using a nitro boost. Power ups are placed regularly and sometimes there will be power ups which are hard to reach but worth the effort. Gravity fields along the track force you to adapt your driving style. Not making a jump to the other side results in a reset of the vehicle on the track, which will cost time. Imagine the track looking like a roller coaster. Background details around the track could be added.

The vehicles

We intend to have two different hovercraft vehicles, each with different properties (e.g. Acceleration, handling etc.). They should behave physically correct and have typical features like nitro boosts, rocket launchers, energy guns, mines etc., each one allowing you to either drive faster than your opponents or to slow your opponents down. Of course there is only a limited amount of each special feature, but you may collect power ups during the race to refill ammo and nitro. The vehicles have a limited energy shield. If you take damage, the shield is reduced. An empty shield will result in a reset of the vehicle and therefore cost time. Power ups to refill shield may be collected.

Weapons, Nitro and Energy

Initially the vehicle has no nitro and no weapons, but it has a full energy shield. Everything can be refilled by collecting power ups. Using a nitro boost results in higher acceleration and higher maximum speed. The boost only lasts a limited time. Energy gun shots look like pulsing, light emitting balls. They do medium damage and are also available as a multi shot version, where three shots are fired at once into three different directions. Rockets have a homing ability and they avoid collisions with the track boundaries and obstacles, tracking down the next enemy vehicle. Mines are placed behind the vehicle and they explode when a vehicle drives over them, damaging that vehicle.

Collisions

Collision detection is performed between the different vehicles, obstacles and the track itself. Collisions may result in damage to the vehicle or in a slow down. If you get too close to the boundary of the track, you will be slowed down. If you hit an obstacle, the boundary of the track or another vehicle, you will take damage and be slowed down.

Rigid Body Engine

A good Rigid Body engine should allow us to place movable obstacles and jumps along the track. Hitting objects is fun and therefore should not always be penalized with damage or a slow down. Varying the gravity at different places along the track should look cool and make the track more challenging since players have to adapt to alternating environments. Imagine for example regions of low gravity and regions where the direction of gravity changes and therefore you will be dragged towards the track boundary or you even drive “upside down”.

Graphics

The game comes in a cel shading look, giving it a futuristic arcade look. Additional shaders may support effects like motion blur at high speeds and other effects for explosions and weapons. Especially motion blur helps to make the player “feel” the speed. The camera will be set behind the vehicle.

Sound

Sound effects are supposed to add to the experience as they demonstrate the power of a rocket or a nitro boost, for example. Varying style and speed of the background music gives a better feeling of the current situation, for example when enemies are close or time is running out.

Mockups

Figure 1: A scene from WipeEout

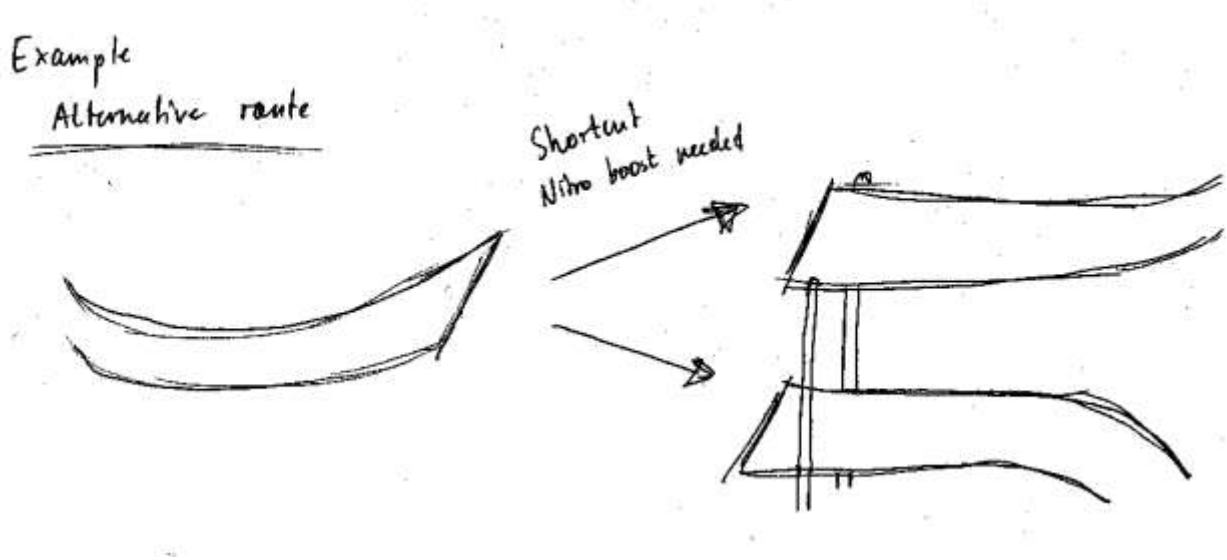


Figure 2: Illustration of the idea about alternative routes:

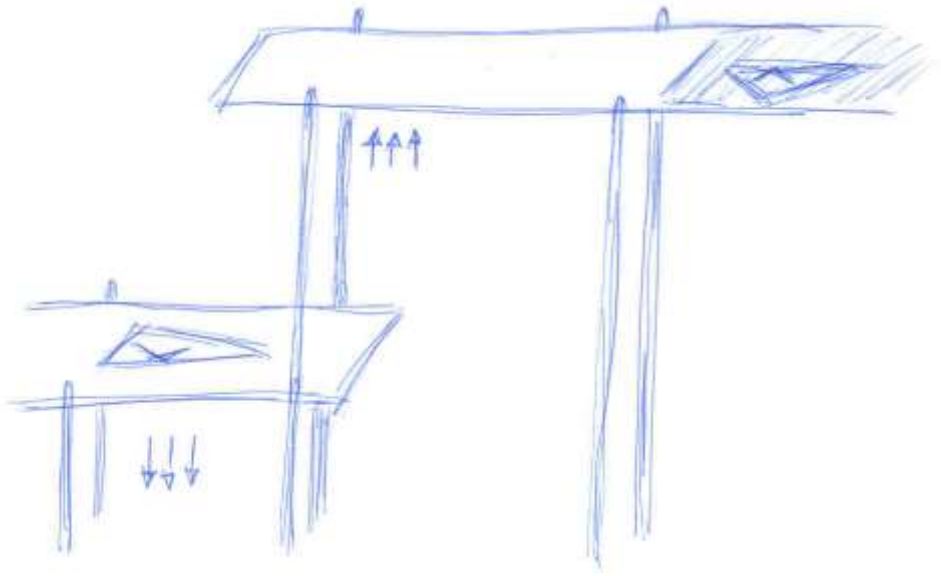


Figure 3: Change of gravity

Figure 4: The energy shot

Figure 5: The missile

The development of the game design

To find the presented idea for our game we started to brainstorm, looking for a game idea that would be realizable and fun. We oriented ourselves at existing games. The initial idea was inspired by existing games like F-Zero, Extreme-G and WipEout. We decided to create something similar and to extend it with new ideas. This is where the jumps and gravity fields along the track come into play. We thought that this might add a lot to the excitement. Of course we wanted to also include weapons since shooting your opponents is always fun. Unfortunately we couldn't stick to our proposal and exactly these components were not included in the game. More details can be found in the following sections.

Development

From the initial idea to the game

We separated the work more or less into the graphics part and the gameplay/logic part. Pascal did most of the latter and Jens most of the former. In the following we describe how the game was created step by step, in chronologic order. For each step a short description including the technical difficulties that we faced is given. The descriptions are adapted from our interim reports, therefore they demonstrate well what problems we had at the given time.

We started implementing the Physics, made ourselfs familiar with HLSL and implemented a cel-shading shader. We also started to get familiar with different modeling tools to model the track:

Interim Report I -15.04.2007

Physics Engine – Pascal

The Physics Engine was quite a challenge. First we searched the internet for physics libraries and we found lots of them. To name some: Ageia PhysX, ODE, Bullet Physics Library and lots of other stuff.

The problem with those libraries was that none of them was designed for C# or even for languages without pointer arithmetic or other low level optimizations. So we decided to build our own physics engine supporting the features we need.

Later on we needed an LCP Solver for resting contacts and we found one in a library called “Simpack” (Details can be found at simulator.wordpress.com or sourceforge.net/projects/simpack). An LCP Solver in Java, let’s port it to C#!

Our decision was not optimal, because porting was not as easy as thought and the differences between Java and C# are big. Mainly the differences in the class inheritance structure and the deep nesting of the “Simpack” library gave me some headaches and sleepless nights.

While porting was reaching an end and some feature were already implemented (rigid body motion, collision detection and resting contacts (nearly finished)) our assistant informed us about a project called XNADev.Ru where the “Bullet Physics Library” was ported to XNA. Well, I spent some hours for nothing and ended up with a library, which is after testing it, good suited for us.

At the moment I am working on the vehicle physics and a demo of it will be hopefully be shown on Tuesday.

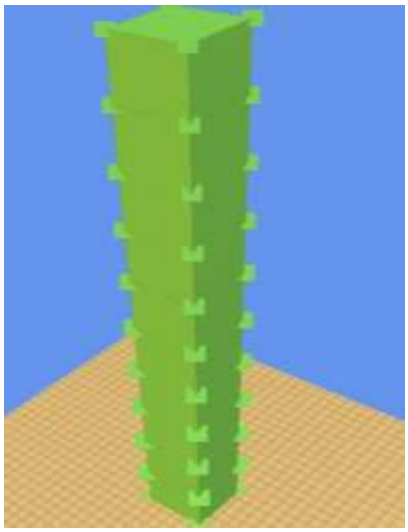


Figure 6: Own physics engine

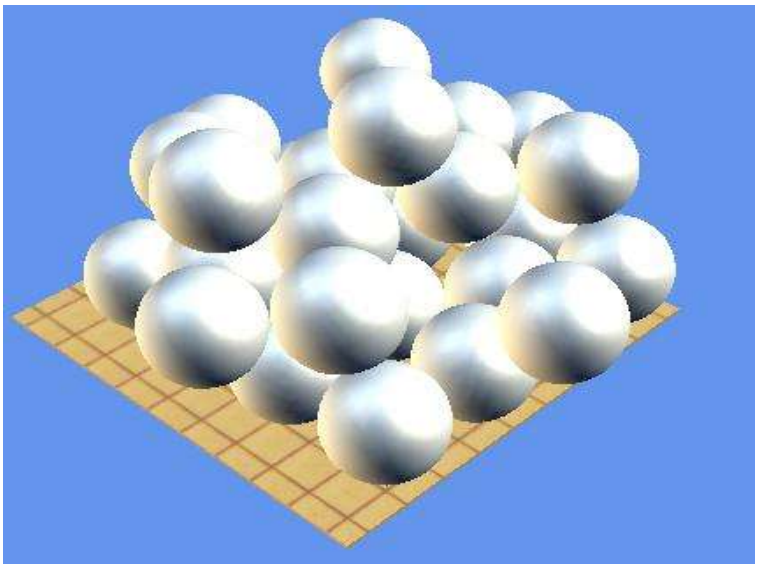


Figure 7: Bullet physics engine

Cel Shading – Jens

Cel shading was not as difficult to implement as we thought. Getting started took some time, though. To quantize the lighting we did not directly use the factors obtained from the diffuse and the specular lighting as scaling factors. We used it as a texture coordinate to a 1D grey value texture instead. This works fine if there is no color interpolation between vertices. Otherwise the color has to be quantized as well, which is not implemented yet (in fact we tried to quantize the different color channels separately, but that didn't look good).

The black lines at the object boundaries are achieved by rendering the whole scene a second time, only rendering backfacing faces, moving the vertices along their normal and using black color.

The fine tuning of the shader has to be done at a later stage when we have the models etc ready. Things like different orientations of the triangles, shading coefficients etc have to be set.

Cel shading experiments:

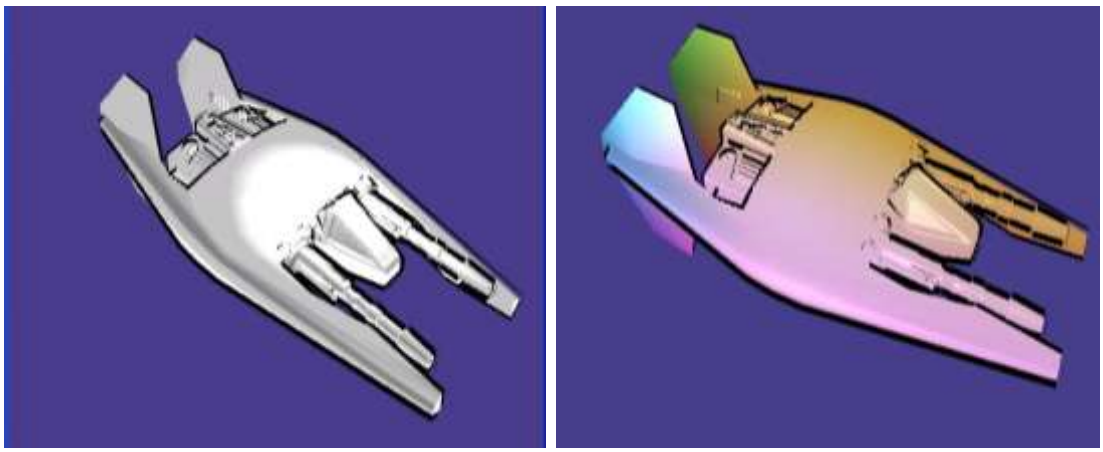


Figure 8: Cel shading tests

Track – Jens

Designing and modeling a track has turned out to be much more difficult than initially assumed. We started using Blender, but we are not sure if we really want to stick with it or change to Maya (if there are no bad surprises, it should be possible to export the models made so far...). Orientation of faces (CW/CCW) have to be made consistent.

Making straight lines is easy of course, just stitching base elements together. Designing curves and twists is what's on the schedule now... Once a prototype is ready, the design of the final track can begin (using pen and paper first, using elements we know we can model and integrate).

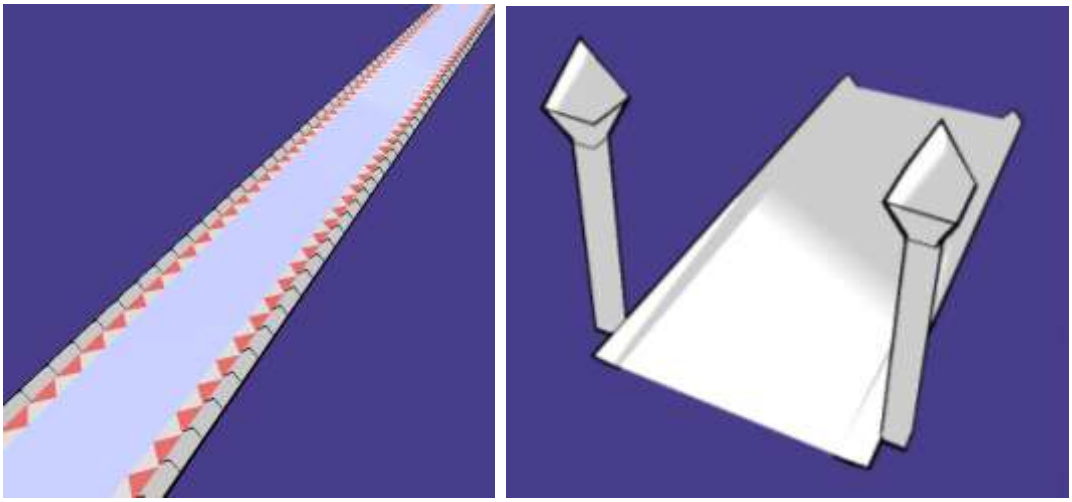


Figure 9: Track, first try

Creating/finding a good, general purpose rigid body engine turned out to be much harder than expected. Therefore we decided to use a pseudo physics engine, custom tailored for our game. We decided to use Maya as a modeling tool and finished a first, simple oval track which is still used in the final game (track0, aka Teapot Valley).

A HUD was created for later use:

Interim Report II -23.04.2007

Physics Engine – Pascal

It turned out that the “Bullet Physics Engine” port to C# is not as usable as expected. The main drawback is that on the Xbox 360 it can sustain only about 10 moving colliding objects without dropping the frame rate under 30.

Therefore we decided to begin a new game engine, which is not based on rigid body physics or on real world physics. It will be a pseudo physical engine completely tailored to our game and supporting different gravity directions and possibly also jumps.

Track – Pascal/Jens

A first, very simple track model was finished by Jens and it will be implemented into the game during following week.

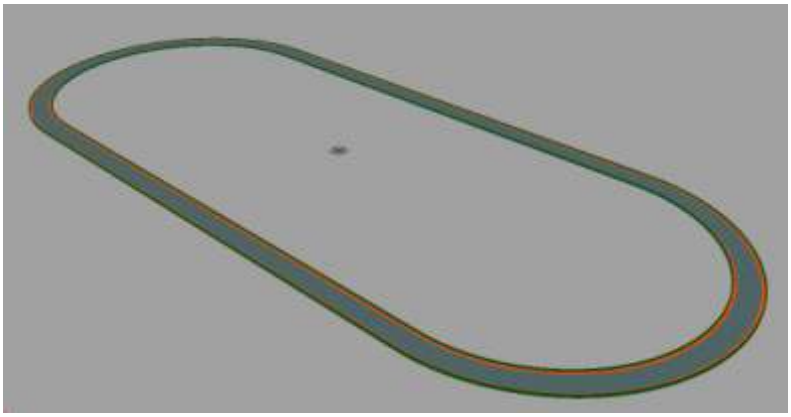


Figure 10: Track0 - first version

HUD – Jens

We implemented a HUD which shows an energy bar for shield and nitro, the time for the current lap, the available/selected weapons and the speed.

Further improvements to the physics engine were made. Additionally we created a meaningful class structure. The shader was extended to cast shadows using a shadow map technique and to support motion blur and tunnel vision. We finished a pre-alpha release and started creating particle effects:

Interim Report II -30.04.2007

Pseudo Physics Engine – Pascal

The new game engine is going well. It is now possible to race on a track which has different gravity direction, but it still needs a bit of tuning.

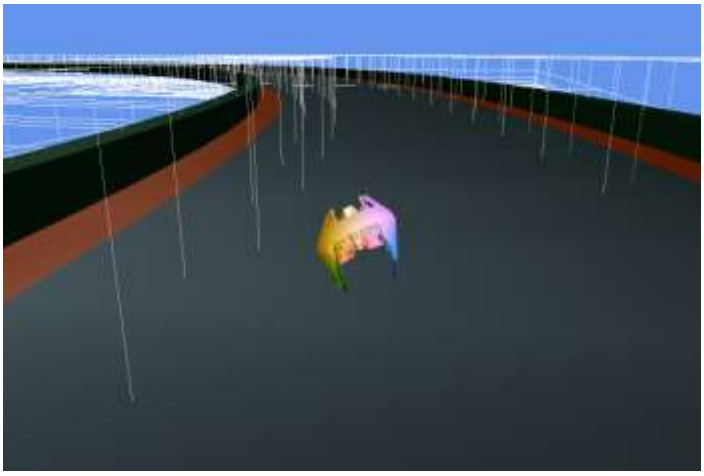


Figure 11: Debug mode for the engine

Game Foundation – Pascal

I implemented some game foundation, e.g. class structure for different game modes and also a vehicle and track selection screen. Nothing interesting, but someone has to do it.

Figure 12: Menu

Shaders – Jens

To cast shadows we use a shadow map technique. At the moment a point light source is moving along with the car to simulate the shadow of directional lighting. This has the drawback that also the track boundary may cast a shadow if it is inside the light region. We intend to place point light sources along the track. It is also possible to cast lights with texture, i.e. for example a light that projects “START” onto the track and the vehicles.

The motion blur/tunnel vision effect is pretty simple. To achieve motion blur we simply display a weighted sum of the previous frames and the current frame. Tunnel vision is simulated by darkening pixels far from the image center. Parameters are passed to the shader to control the amount of blurring and tunnel vision.

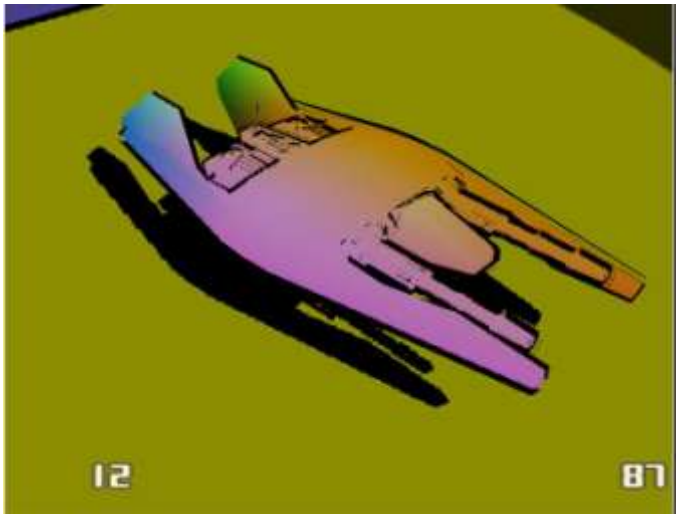


Figure 13: The HUD and shadow map tests. The “12” at the lower left corner is debug information

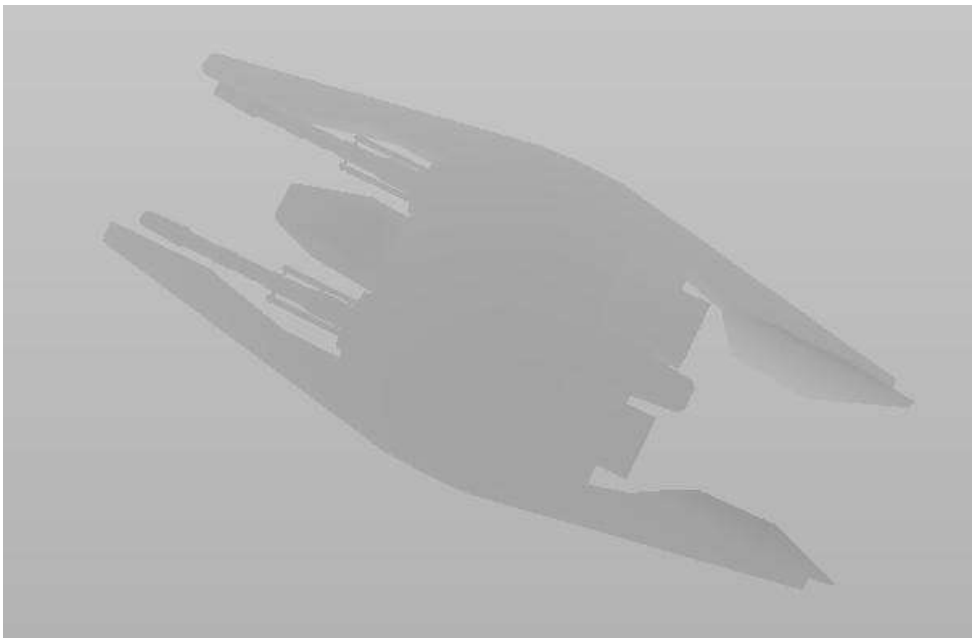


Figure 14: Shadow Map

Particle Effects – Jens

Getting started.

Pre-Alpha Release – Pascal / Jens

Over the weekend we combined our work resulting in a pre-Alpha release. We could play one track, which has two checkpoints where time is stopped. So we did some race against the clock and Jens won.

What is a game without sound? We added music as well as sound effects. Split screen and an additional track had to be done. On the graphics side we continued with the particle effects and some problems concerning the shader kept us busy:

Interim Report III -13.05.2007

Sound – Pascal

I was browsing around the web and stumbled over the "Racing Game" example on the XNA Creators Club web page. Good for us, there is some racing sound with the packing, which we included in our game.

To include sound in our game, I used the "Microsoft Cross-Platform Audio Creation Tool (XACT)" as explained in the tutorial. The only problem here was that XACT doesn't work on Windows Vista. No problem, I simply took my old XP.

For the engine I found an F/A 18 engine sound, sampled it, and modified the pitch etc. with Audacity. Then I included it in XACT, included some "RPC Presets" which are bound to a variable called "Speed" and now it is possible to control the pitch and volume of the jet sound during game play.

Split screen – Pascal

Split screen was not a real issue only something that has to be done. To avoid too much code duplication some classes had to be split up and some new ones had to be created. Nothing exiting at all.

Car-car collision works as expected. We used the XNA Sphere Bounding Boxes to achieve this in a simple and efficient manner. The collision is physically like an inelastic impact.

Figure 15: Split Screen

Particle Effects – Jens

I just followed the tutorial that was posted on the twiki (http://jsedlak.org/node/177) and adapted it to our needs. Right now we use our particle emitter to produce smoke.

Unfortunately we had/have some performance issues. After several experiments we came to the conclusion that the pixel fillrate might be the problem since drawing e.g. 2000 sprites of 96x96 pixels means a lot of pixels (~100*100*2000*20 (frames) = 400 Megapixel...) and reducing the sprite size increased the performance. But once we reduced the size of the sprites we got performance problems with the managing of the particles (updates etc). So we thought we could handle the particles on the GPU. After a while we came up with a suboptimal solution: one texture for the particles positions and one for their velocities (e.g. 300*300 pixels). In a pseudo drawing pass where we render a quad that occludes the whole screen (and therefore all pixels in the pixel stage are accessed) we update the positions and velocities. To render the particles, we pass the appropriate number of points to the graphics card (e.g. 90'000) and read their positions by fetching the positions texture in the vertex stage (this is a shader 4.0 feature, supported by the Xbox). This is done by assigning each point a unique texture coordinate referencing one pixel in the positions texture.

Unfortunately this didn't work completely since some artifacts remained that we were not able to resolve.

There has to be an easier, more elegant way to do the particle handling on the GPU but we could not find it in a short time. But since the results we get with the CPU look allright we will probably stick with the CPU implementation.

We will probably use more particle effects if our schedule allows it (e.g. fire like effects etc).

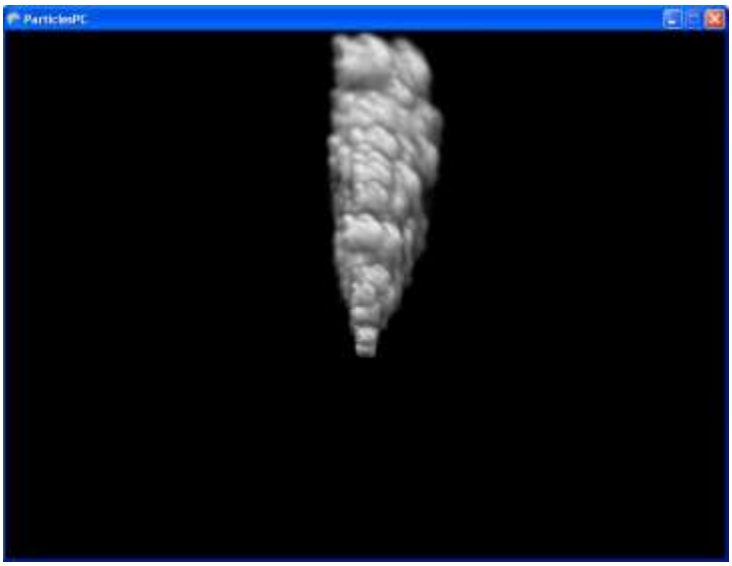


Figure 16: Our Particle emitter

Shader debugging – Pascal/Jens

We had some problems with the rendering. It turned out that one of the problems was the orientation of the faces (clockwise vs. counterclockwise) which is not chosen in the same way by different exports from different programs. A problem which remained is the following: if we use *.fbx files which include materials then those parts of the mesh using materials and no textures are rendered black if we use our custom shader.

Track

Now that we know that we are able to drive the vehicle on non-horizontal tracks we may start modeling a more sophisticated, probably final track. It should be about ten times as big as our current track (which has a simple, oval form) and take between 90 and 120 seconds to complete a lap. We are not sure yet whether we will include jumps/are able to include jumps.

Final Report

Since we had problems using the second track, track2, with our pseudo physics engine we modeled another track, track1 aka Asteroid Field, which suited our engine. Background details and textures were added to the first, oval track. In a later stage also the new track was extended with background details and textures. In both cases we used a skybox textured with images taken from outer space as background.

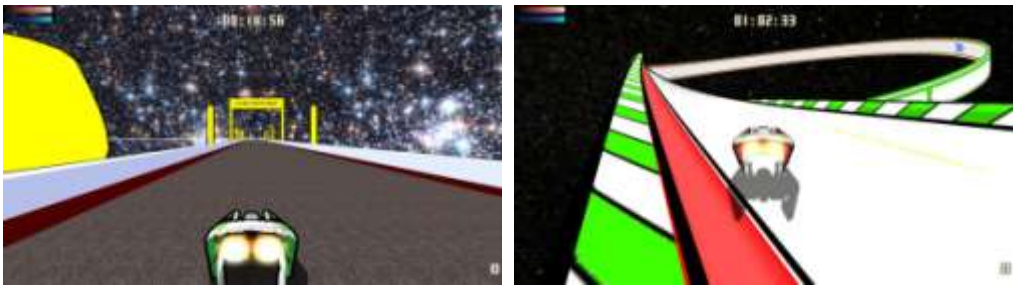


Figure 17: Track1 and Track2

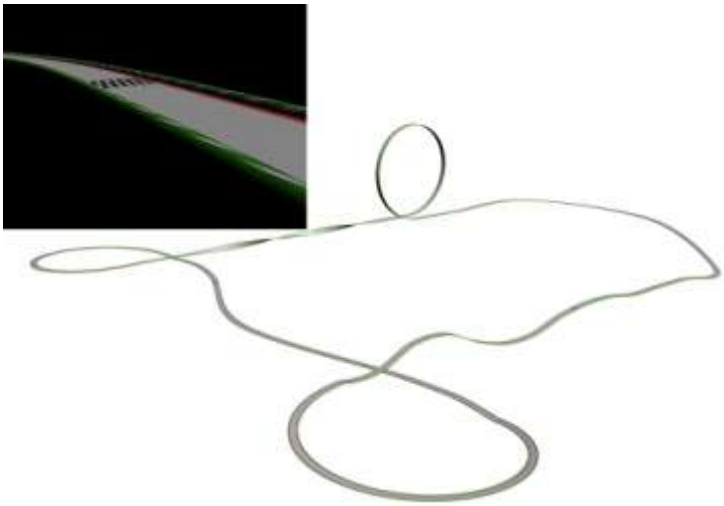


Figure 18: Track1

To make the vehicle nicer we added thrusters. The first version didn't look good in the game. In a second version a bloom effect was used to demonstrate the high light intensity of the thrusters. This was realized as a new shader.

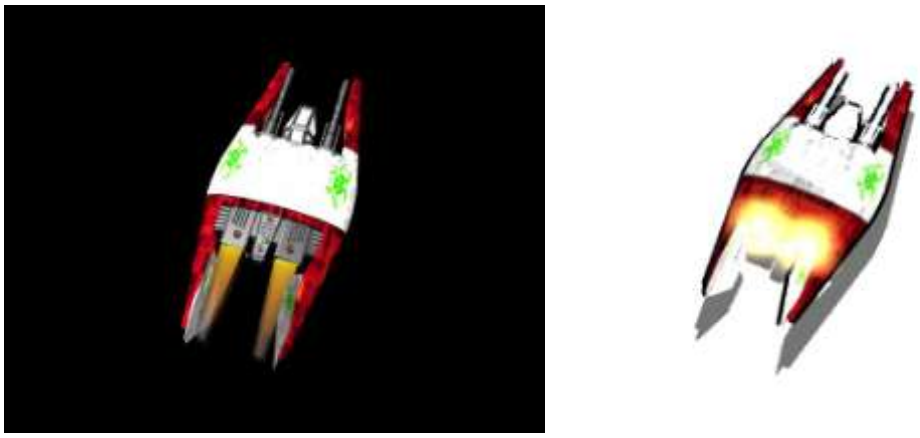


Figure 19: Thrusters w/o and w/ Bloom effect

To enhance the overall look of the game some menus were redone and a loading screen explaining the control system was added.

Much time was spent in a desperate attempt to make the huge track2 work, but we couldn't do it during the time that was left.

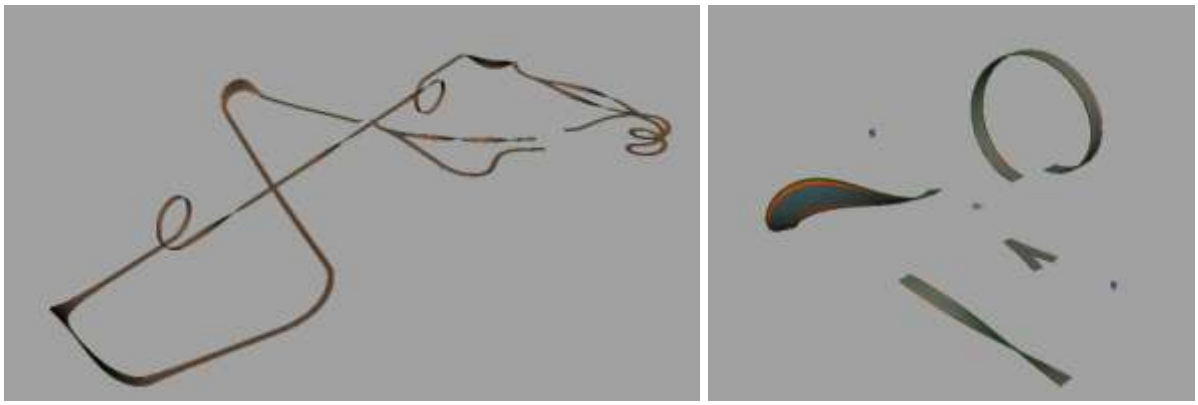


Figure 20: Track2 in its glory

Unfortunately this wasn't the only thing that had to be skipped. Weapons and AI for computer guided enemy vehicles didn't make their way into the game. Basically we underestimated the time and effort. Problems that we didn't expect came up and some bugs were hard to find. The modeling was harder than expected. Transforming the ideas in your head into a 3D model is quite difficult and texturing it can be very time consuming since there is a lot of fine tuning going on if you want to avoid visible seams and the like.

Some bugs that were very hard to trace and eliminate arised on the shader's side. Sometimes it is enough to slightly change the order or parameters of certain functions and your graphics will be totally messed up. There were also some XNA specific problems that we had to overcome.

A very big drawback was the physics engine. Trying many different things (full rigid body physics, 3rd party physics library and finally our pseudo physics engine) until you got a working engine is very, very time consuming.

After all we think that we could have included all the features if our development process would have run more smoothly and without the just mentioned problems, except for the additional features that would have needed a sophisticated, more general purpose rigid body engine. We "wasted" a lot of time with things that didn't make it into the game, but we gained some useful experience for other projects.

Development tools

For the modeling we preferred Maya over Blender. Maya was basically used for all modeling tasks. It needs some time to get used to it but it offers much functionality and we think that we only scratched the surface. Knowing this tool better would certainly have made our lives easier.

Other tools we used are XACT and Audacity for the sound, GIMP, MS Paint and Paint.net for textures and of course well known office products and their also well known alternative office suite for our presentations.

To handle different versions of the game, exchange and merge code and content we used CVS.

Playtesting

We had four people testing an alpha version of our game, two casual gamers and two experienced gamers. One of the experienced gamers is a car fan and likes racing games. The game supported the first, simple track, track0, and a buggy version of the second track, track1. Time attack mode and two player mode could be played.

The setup

All testers played on an xbox360 which was connected to a Dolby Digital 5.1 surround sound system. Two of them played on an older Pal TV and the other two played on an HDTV. We offered them food and drinks, not to bribe them but to show them our gratitude and make them feel welcome. The casual gamers tested alone. Therefore they only played the single player version. The other two gamers tested together on the same device. They played single and two player mode.

The Feedback

Since we didn't want to ask suggestive questions we let the testers more or less talk freely and tell us what was on their mind. We asked them to tell us what they liked and what they didn't. At the end we asked them what they would add to the game and what they would change to make it better.

On the positive side the game idea and the simplicity of the game were mentioned. The testers could play it right away, no explanations were needed and the learning phase was quite short. They liked the feeling of the speed.

On the negative side the bugs dominated. The steering had to be fixed, the camera controls were not working as they should have and the shadow map was not working properly on track1. Not everyone liked the skybox in track0.

The biggest list was the one containing suggestions for possible improvements. Things that were mentioned which we already had planned to integrate were: AI (computer guided enemies), weapons, textures, ranking in two player mode, more vehicles (looks and physics). But our testers also brought up some ideas we didn't think of: a minimap showing the players position on the track, a pit stop to refill attributes like nitro, other recharge areas and speed bumpers on the track to increase speed. The racing game fan would have liked curves that he could anticipate more, s.t. he could prepare himself and s.t. he would not have to rely solely on his reactions. To achieve that he suggested for example landmarks.

The feedback was much as we expected it to be, not because we asked suggestive questions, but it was obvious that some things were missing or didn't work.

Conclusion

Since this was our first game of this scale we had some difficulties estimating the time needed to complete the different tasks. If we create another game in the future we will probably identify time consuming and difficult tasks more easily and plan the development process in more detail (e.g. engine, graphics).

Overall we are quite happy with the results even though we think it's a pity we didn't include weapons and AI. This would have added a lot to the excitement of playing the game.

The course was a good experience. Implementing different techniques and seeing it all come together was nice. However this is also the part where we have some suggestions for improvements of the course. It would be very nice to see examples of successful, large scale games and what tricks are used to make them stand out. For example it would be interesting to see how a highly detailed 3D model is created, what its complexity is and how they made it look so nice even though the model complexity is maybe not as high as one might think. Another example would be some amazing looking graphical effects or stable game engines with good AI and physics.

We don't know whether it is possible to get such information, but we think that many tricks and creative ideas are used in games to make them better. Of course we should come up with our own ideas, but examples might give us some impulses and maybe let us do more in the same amount of time.

Having XNA tutorials is a nice idea. But since some tutorials were given only after several weeks we basically didn't need them anymore. Maybe it would be a good idea to split the tutorials and have the different group members visit different tutorials at the same time. Or hand out some tutorials at the beginning of the course. It would also be nice if the tutorials contained some sample code, for example a shader example and how it is used within the XNA framework.

Many tutorials could be found on the internet. Therefore this wasn't a big drawback.

We don't think that the course schedule was too compressed. 10 credit points is a lot and therefore it is clear that some work has to be done. If the development schedule is realistic and not too many unexpected problems occur, 14 weeks allow you to do quite much. The goals shouldn't be set too high. It is useful to have some time left at the end to tune and enhance the game visually and gameplay wise without having to add more gameplay logic like additional modes etc.

All told we learned a lot and would probably take the course again.