Developing a Mini Racing Game

Results of a practical course at the Chair for Computer Graphics and Multimedia
(RWTH Aachen University, Germany)

Philipp Bock* Jonas Fortmann† Sabrina Kowarsch‡ Jochen Schmücking§ Leon Staab¶

Figure 1: *A general overview of the whole racing track. It is situated within a harbor-like scenery which includes typical objects such as a moving crane, boxes, containers etc..*

Abstract

The general task was to realize an intuitively playable mini game. The main interest was to implement fancy graphic effects such as bloom, HDR lighting and shadows. For our scenery we decided to situate the racing track in a harbor environment, because it offers a great amount of modeling possibilities. As we can see in Figure 1, the racing game is partially taking place inside of a hangar. The implemented lighting effects contribute greatly to the transition between in- and outdoor situations.

To create the game, we used multiple different tools. Content creation and level design were made with Blender 2.63 whereas for programming the Qt Framework was used. To ease the programming process the Bullet Physics Library and ACGL were provided.

Keywords: game programming, mini racing game

1 Game Logic

The track is a circuit. Main goal of the game is to complete three laps as fast as possible. The environment is closed and hence there is no freedom of taking shortcuts and/or leaving the track. This is also made sure by the use of checkpoints which have to be passed in the correct order. If the player gets stuck, flips over or falls into the water, the car can be reset to the last checkpoint that has been passed. To make the game more fun, we added a leaking oil barrel. This creates an obstacle which should be avoided because going through the oil leads to uncontrollable sliding. A similar chicane are the tight entrances of the hangar.

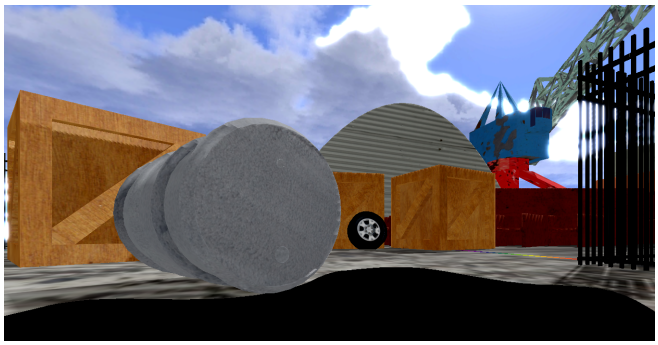

Figure 2: *The spilled oil makes the game much more challenging*

2 Physics

The physics simulation is handled by Bullet. To implement a racing car which shows playable behavior, Bullet provides a raw raycast vehicle class. This class, however, needed to be adapted to our project. For this purpose we had to tweak a lot of parameters. A better simulation of the car velocity can be achieved by implementing a soft speed cap via an arcus tangens function.

The camera is in third person behind the car. To avoid objects getting between the camera and the car we use a three position camera system. The camera position is set closer to the car if it collides with walls or other objects. During the implementation of the camera we were confronted with heavy camera jitter, because Bullet does not frequently update the car position on which the camera

*philipp.bock@rwth-aachen.de
†jonas.fortmann@rwth-aachen.de
‡sabrina.kowarsch@rwth-aachen.de
§jochen.schmuecking@rwth-aachen.de
¶leon.staab@rwth-aachen.de

Figure 3: *The pickup truck which is controlled by the player. In the background one can see some containers.*

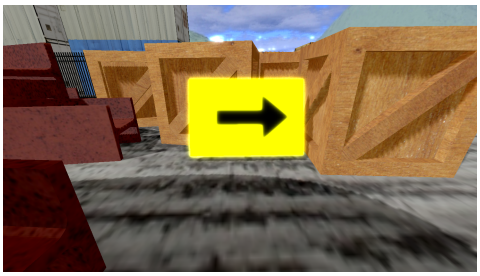

Figure 4: *Bloom: The light of the bright arrow sign seems to bleed over its actual edges.*

depends. That caused the camera to use a slightly outdated position. We fixed this issue by taking the position manually instead of waiting for Bullet to pass it.

3 Content Creation

Since we planned to create a lot of content we were in need of a way to easily port our blender designed models into our game. As Blender is able to export 3D geometries along with texture coordinates itself, we just needed a way to store their composition within our game world.
Because of Blender's powerful API and Python scripting support we decided to do so by arranging our level in Blender and store the performed transformations in an XML file. This enabled us to design a level and load it into our game during runtime by parsing the auto-generated level file.
Still, the car did not physically interact with our world. So we had to describe the physical behavior with Bullet Physics. Although Bullet provided many possibilities we only used several basic shapes to set the physical boundaries of our meshes. Using Blender and its scripting capabilities we were again able to automatically export this association into XML. Now even the physical representation could be created or altered during runtime.
Originally it was planned to associate shader and shader options too, but this was discontinued and hard coded.

To make the scenery look more individual, the containers' appearances randomly change every time the scenegraph is reloaded (see Figure 3). In order to achieve a more lively atmosphere, the crane's arm moves up and down. This also creates a big dynamic shadow on the ground which strengthens the effect.

4 Graphics

When discussing which graphic effects should be implemented in the game, we decided to focus heavily on lighting effects to create an atmospheric game.

To create a more dynamic view of the scene, we make use of HDR lighting. By using more than 256 different values for each of the color channels, extremely bright sections can be displayed. On these bright sections, a blur filter is applied and the resulting image is added back to the original one to create the bloom effect. The bloom effect imitates the behavior of our eye when looking into bright areas. These areas seem to bleed over the nearby sections, like when you look near the sun. To save computing time, the blur filter is used on scaled down versions of the image. Due to the resulting upscaling, the blur effect becomes stronger while needing less computing time. To imitate the function of our eyes, a dynamic tone mapping filter is applied to the result.

For tone mapping, the average luminance of the visible picture is calculated each frame and the HDR picture is scaled down to RGB space using a factor calculated with the luminance. When entering a dark room, the brightness level of the scene slowly adapts, as does the human eye.

Basic shadow mapping is also implemented in our game, enhancing the degree of realism strongly. The shadowmap is calculated for each frame, such that the shadows of moving objects as the crane or the car are always updated.

A real time reflection is visible on the surface of the water which enhances the visual quality of the game. To achieve this effect, the scene is rendered from a perspective beneath the water surface. The car also reflects the skybox, such that the scenery seems more realistic.

The shading models used for the materials is the Phong-Blinn shading model. This was in our opinion the best choice to make the scene look less plastic-like, especially on the containers which now look more like they are made of metal.

5 Particle System

We wanted the car to burst smoke out of the exhaust pipe like an old truck. Therefore we use a particle effect which emits smoke patterns attached to a billboard. The billboard is aligned to the camera so the particles will always look like they are 3-dimensional. The emitter is attached to the car. To make the particles look more realistic, they are given a slightly different color and a start angle on which they are rotating. As time goes on, the particles are getting bigger to simulate the fade of the smoke. Also when the particles are created, the floating direction will be randomized in a range of [-0.5, 0.5] respective to the old direction. The particles are also given the speed of the car added to their own speed so when the car stops, the emitted smoke will fly forward. To avoid that the particles are seen as a big black cloud when the car is not moving and are not seen when it is driving we added a speed control to the shader which makes the alpha channel more saturated when the car drives.

6 Conclusion

We realized an easy, but still entertaining, Mini Racing Game. Several lighting effects as well as a particle system have been implemented to make the game visually appealing. The car has a realistic physical behavior and can be controlled intuitively. As the level was created using Blender as an editor and then exported via a Python script, additional levels/content can now easily be included into the game. Weekly meetings supported us maintaining a constant workflow during the whole practical course and exchanging our state of work. The work distribution was satisfying and we have learned a lot in the respective field of work.