# Developing a Racing Game

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

Michael Lellmann[*] Slava Belopolski

## Abstract

In this paper we present a racing game through the streets of Aachen. The game takes place in the center of Aachen where you drive through the walking area around the city. It is a single-player racing game.

**Keywords:** game programming, Aachen, street racing game

## 1 The Graphics-Engine

The game is rendered with a deferred shading technique. Therefor the scene will first be rendered to a GBuffer and afterwards we use that buffer to do the lightning calculations. After the lightning is done we will do some post processing to make the image look better.

### 1.1 The GBuffer

We store rather many data in the GBuffer, but that does allow us to not only have normal and texture data for each pixel, but also data which are important for the lightning calculations. This data includes the material properties which like the ability to reflect different types of colors as well as the influence of different light sources which are ambient, diffuse and specular.

We have the ability to use textures to set those values for each pixel individually making it possible to have very good visual effects.

[*]michael.lellmann@rwth-aachen.de

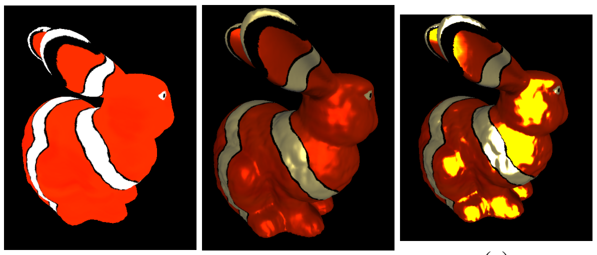

**Figure 1:** *The bunny rendered with different lightning models.*

### 1.2 Lightning

For the lightning we support 3 different models. The first (cf. figure 1 (a)) is the simplest one which will only take the texture color and does not do any further calculations. This simple model is best suited for surfaces that should not be lighted like the sky box.

The second lightning model is used for dull surfaces. It is the Blinn-Phong lightning model (cf. figure 1 (b)) which does support ambient, diffuse and specular terms. It is used for most models.

The Cook-Torrance lightning model (cf. figure 1 (c)), our last implemented lightning model, on the other side is used for shiny surfaces because it works for those the best.

We do support shadows for each light individually. We use a simple shadow algorithm that uses 4 points around the pixel to check them for shadows and we calculate from those 4 points the shadow value which results in smooth shadows.

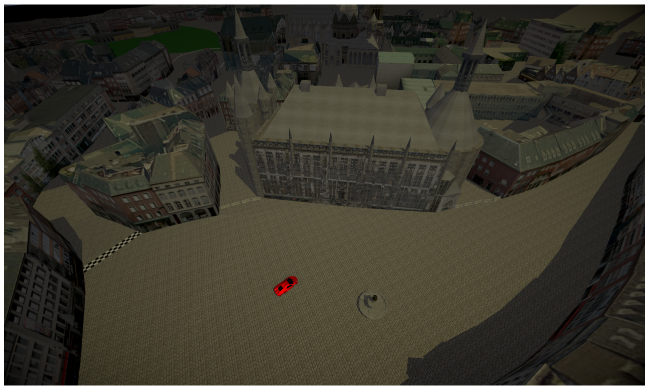

**Figure 2:** *The aachen model.*

### 1.3 Post-Processing

We use high dynamic range rendering for the lightning. In the post processing we apply Jim Hejl's ALU filmic tone mapping. The tone mapping also includes adapting the screens luminance according to the average luminance of the screen. We also darken the sides of the screen to make the player focus more on the center of the screen with a vignette effect.

For Anti-Aliasing we used the nVidia FXAA one which we adapted to work with our code.

### 1.4 Oculus Rift Support

We also support the Oculus Rift. For that we render the scene twice into the GBuffer, for each eye into the appropriate side of the GBuffer. All other processing is than done in one step on this GBuffer. We take care that the shader use the correct values for computations depending on the coordinate of the pixel. Doing it that way allows us to only need to render shadow maps once for both eyes. It also makes the post processing easier because the adaption of the brightness of the screen is done in the same pass thus we don't need to care that both eyes get darkened or brightened the same. This method has one negative aspect, because it will to the Anti-Aliasing in the middle of the Buffer with values from the left and the right eye. But as the border of the screen is darkened and with the effect of the distortion shader needed for Oculus Rift, we have no visual impact with that.

## 2 The Physics-Engine

For the physics we use the bullet physics engine [bul August, 2013]. For the car model we use a simple model which is included in bullet that allows to set breaking and accelerating forces at the wheels and that allows the car to move around.

## 3 Models

### 3.1 3D model of Aachen

The 3D model of Aachen has been provided by the computer graphics chair of the RWTH-Aachen university. The model consists of an untextured mesh with the whole city. As the game takes place only in the center of the city, the mesh has been reduced to the racing area. This allows fewer computation due to smaller size of the mesh. Furthermore the mesh had to be textured (Figure 2 (b)).

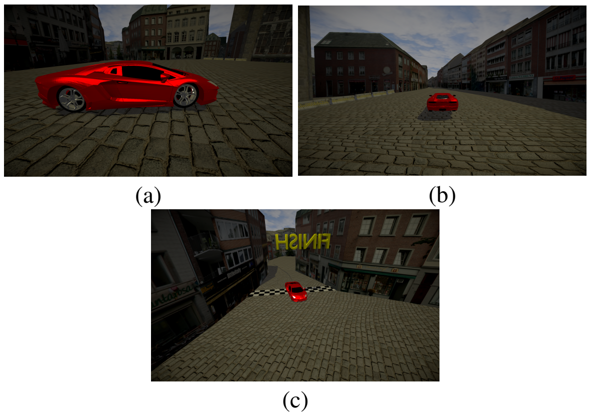

**Figure 3:** *The car model and the barriers.*

### 3.2 The car model

The car model has been taken from [TF3 August, 2013]. It has been prepared for rendering, like the wheels have been removed and stored in a separate .obj file due to requirement of the Bullet Physics. Furthermore some materials have been removed or substituted as the interior of the car is not visible in the game, to reduce the file size (Figure 3 (a)).

### 3.3 The barriers

The barriers model has been taken from [Tur August, 2013]. The barriers have been placed around the track to mark the borders and to allow the player to follow the track easily (Figure 3 (b)).

### 3.4 3D text

Before the game starts the counter appears in the game with the spinning camera around the car. 3-2-1-Go! After that the camera position changes and the player will be able to drive the car. Each of the numbers is an .obj file and the files will be rendered one after another (Figure 3 (c)).

## 4 Work distribution

Michael did concentrate on the programming, especially the full rendering engine, the use of the physics engine and supporting Oculus Rift.

Slava did work on the 3D models like on the map, sky box, car, counter, texturing and modeling the track. This also includes loading and rendering models in the game.

## References

August, 2013. bullet. http://bulletphysics.org/wordpress/.

August, 2013. Tf3dm. http://tf3dm.com/3d-model/lamborghini-aventador-42591.html.

August, 2013. Turbosquid. http://www.turbosquid.com/3d-models/barrier-obj-free/612650.