Real-Time Rendering - Submission 2

Lukas Lipp 1425235
Michael Oppitz 1227129

January 15, 2018

Description

The main inspiration for the scene comes from the vapor wave artstyle. A lot of vibrant neon colors are used there and different filters are integrated. In this scene a car is followed while driving through this abstract scenery.

Additional Libraries

- GLM OpenGL Mathematics
  `http://glm.g-truc.net/0.9.7/index.html`
- GLEW OpengGL Extensions
  `http://glew.sourceforge.net/`
- GLFW OpenGL Window
  `http://glew.sourceforge.net/`
- PhysX Physik
  `https://github.com/NVIDIAGameWorks`
- FreeImage Imageloader
  `http://freeimage.sourceforge.net/`
- Assimp Asset importer
  `http://www.assimp.org/`
- FMOD
  `http://www.fmod.org/`
- FreeType
  `https://www.freetype.org/`

Effects

Implemented by Lukas Lipp

Multi-pass Glow

The goal is to let the floor glow in a neon like style. Three framebuffers were used for this task. The first one is used to render everything that is behind the floor and the floor itself. The second one is used for the object which should receive the glow. In this case it is the floor. The third is used for everything that is above the floor, for example the car that is used.

In order to let the floor glow, the texture of the second framebuffer is blurred with a gaussian filter and than

again written into a texture. This blurred texture of the floor is than blended over the texture of the original floor.

The last step is to bind the texture with the floor and everything behind it, the texture with the blurred floor, and the texture with everything in front of the floor to a shader which blends these three textures together and draws the result. The reason why so many passes were needed is, the floor is just a grid so everything behind it is visible, also it is occluded by the objects in front of it. Therefore, it was needed to isolate it properly.

The gaussian filter that was used blurs in x and y direction in a single pass. When blurring a large grid the problem is that lines that are far away are much smaller and therefore are blurred less than the parts that are really close to the camera. If the sample rate is increased, it is possible to blur the lines in the back better but as a result of the high sample rate the lines in the front are blurred much more than they should.

The knowledge used for this task was mainly acquired from gpugems.

CRT Filter (Shader)

A CRT filter is used to simulate the look of an old CRT TV. This is done by rendering the whole scene into a texture. Then the texture is drawn on a rectangle with a modified fragment shader. Often Emulators of old gaming systems use them to simulate a nostalgic look and feeling. This shader changes the sharpness and adds scanlines to the image. Sometimes the edges are distorted to simulate the curved screens of old tvs. For this implementation the curved edges were ignored.

For the implementation a gaussian and a lanczos filter is used. A lanczos filter is a reconstruction filter and is helpful to create a more blocky version of the input image but also blures the image. The gaussian is used to blur the input image and add a glow to the shader, like a crt tv has. The scanline weight is how much influence a given scanline has on the current pixel. The mask weight is used to adjust how much of the effect influences the current pixel.

The gaussian and two lanczos of the pixel position are calculated. One lanczos with a low and one with a high sharpness is used. The two lanczos are then interpolated to get a blocky image which has still sharper contours. This is the crt grid. Then the gaussian and the lanczos results are combined through the help of the scanline weight and the mask weight to create the end result.

There is basically no explantation on the internet how to create crt shaders, therefore some shaders on gametechwiki were studied to come up with a solution for this implementation.

Implemented by Michael Oppitz

Light and Shadow System

Different types of lights were implemented into the scene that can illuminate the scene at the same time. The different types of lights are ambient, point, spot and directional lights. The fragment shader stores them in an array of structs which contains all the properties of the lights. The illumination is then computed per light and the result is summarized. The resulting combined color is gamma corrected by a gamma value of 2.2.

All of these types of lights, except the ambient light, can cast shadows simultaneously. For the directional light, shadow mapping with an orthographic shadow camera was implemented and for the spot light, shadow mapping with a perspective shadow camera was implemented. For both of these methods the computation was implemented with a simple shadow pass to compute the depth and a sampler2DShadow texture in the fragment shader that has to be evaluated per fragment.

Omnidirectional shadows of a point light are created on a cube texture. This approach was implemented with the use of a geometry shader to creates the vertices of the shadow cube. The cube texture is passed to the fragment shader and is evaluated as a samplerCube there.

All lights and their according shadows can be adjusted easily. The values of position, intensity, the option to cast shadows, the shadow map resolution and the clipping planes of the shadow camera can be adjusted for all lights. Values like attenuation, a specified target, and a cone angle can be specified for the lights that have these properties.

The ambient light is visible throughout our entire scene. The spot light can be seen illuminating the car while the camera is outside of the car and the directional light is illuminating the car from the outside while the camera is inside the car. For both of these lights shadows can be seen on the interior of the car. The point light can be seen at the scene inside the car, where the point light is inside of a spinning object to demonstrate the ability to cast shadows in all directions.

For the implementation of the different types of lights this tutorial by Tom Dalling was used for guidance. The shadows of the directional light and spot light were implemented with the help of this tutorial. For the omnidirectional shadow mapping the slides from Peter Houska were used for help.

Shading and Environment Mapping

For the environment map and reflections of it, a cube texture is passed to the fragment shader. The actual environment is drawn with a cube for which the depth mask has been disabled and which position is always the position of the current camera.

For the car, a simple Blinn-Phong shading was implemented. There is a possibility for every model to specify either a texture or a single value for the amount of reflectivity. If a texture is specified, this texture is sampled in the fragment shader to get the amount of reflectivity for this fragment. The final color of the fragment shader is a linear interpolation of the color that is computed for this fragment and the reflective color from the environment map. The amount for this linear interpolation is specified by the reflectivity value.

This effect can be seen on the side of the car where there is a bar with a higher reflectivity than the normal amount. This is due to the reflectivity map where the reflectivity values where sampled from. On the inside of the car, the normal Blinn-Phong shader can be seen on the dashboard, as this area does not reflect the environment map. Due to aesthetic reasons, the floor does not receive any illumination and the color of the according texture is just passed through as the color of the fragment.

For this part, this tutorial on learnopengl was used.

Color Banding Fix

The issue of color banding for areas where the colors are similar was fixed by dithering.

Idea from shadertoy user Reedbeta. Function for a random value in the fragment shader from this discussion.

Models

- **Camaro**: The camaro model was taken from user dskfnwn on TurboSquid. The geometry was adapted to fit the requirements of the scene with the help of Maya. The material was created with Photoshop.
- **Skybox**: The skybox was generated with following tool. `http://wwwtyro.github.io/space-3d/`
- **Other Models**: All other models were created with Maya. The textures were created with Photoshop.

Music

- **Main Theme**: Accelerated by Miami Nights 1984

Additional

The debug camera can be activated at any time by pressing the ”n”-key.
The scene can be started in fullscreen with the ”fullscreen.bat”.
This scene was tested on NVIDA.