Echtzeitgrafik Documentation

Gruppe 99 EZG 19 - Into the wild

Jonas Auer 1426685
Lukas Kobler 1427198

Illumination models:
All objects except some primitives are rendered using the blinn phong illumination model. This means that the calculations between the vertex- and the fragment shader interpolate all attributes, i.e the fragment position, texture coordinate and normal are interpolated from the values of the surrounding triangle vertices. This allows us to get very smooth illuminations without having the models to be super detailed. After this interpolation, usually all fragments in a triangle will be having their own interpolated values, which are then used to calculate the illumination of the fragment through the light source.
Other illumination models, like the ones for primitive objects, use a simple color shader. All it does is to assign each fragment the same color, without considering the lights in the scene or the fragment normal. This is for example used to model a simple light bulb, by rendering a primitive and assign a light yellow color to it.
The illumination model of the particles is almost done the same way as the color shader, with the only difference that the color is not constant, but rather coming from a loaded texture.
The last illumination model, the one of the water, is using a bump map together with a dudv map in order to simulate a realistic water. The bump map is giving the normal for a given fragment, while the dudv map can be used to look up the distortion of the refraction and reflection on a given texture coordinate for a fragment. The color is then calculated by taking a mixture of the refraction and reflection (by considering the fresnel effect), before adding the specular highlights of the blinn phong model to it.

Model loading:
In order to load models from different file extensions, we are using the Assimp library, as mentioned below. Using it, we are able to load models from a variety of formats. For this application, we mainly use the .obj file format, since it is widely used and most models are available or easiely converted to it.

Textures:
We are only using 2D textures in our application, except for the skybox, which is a cube map. However, internally we render the six sides as simple quads and single textures, not as a cube map. Most of the textures are diffuse, but for example the water is also using bump maps and distortion maps.

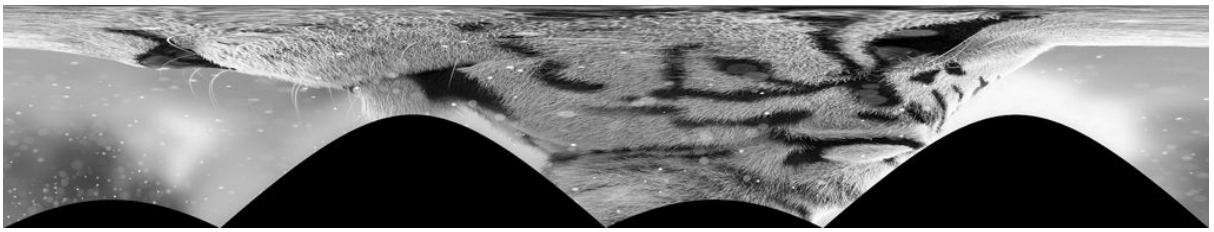
Camera:
The camera is defined by a position and the angles pitch, yaw and roll. This is then used to create the view matrix, which is basically the same as the inverted transformation matrix of the camera. The camera path in the demo is defined using keyframes, which hold a 3D position and the pitch, yaw and roll. Using these keyframes, we can define Bezier curve-segments, each of them holding two endpoints and one control point. The location of

the camera within a curve segment is then interpolated using the corresponding Bezier curve. The rotation of the camera is interpolated linear between the starting and ending point of the curve-segment. By defining multiple curve-segments and sticking them together after each other, we define the camera path.
The camera can also be controlled by using the W, A, S and D keys for the movement and the mouse for the view direction. To stop/start the automatic camera movement, the key P can be used.

Scene:
Our scene is taking place at a late hour, after sunset has already happened. As light sources, artificial lights such as street lamps, campfires and particle systems are used instead of the sun. The main light source is be the moon, which acts as an ambient light source. Our scene is located at a lake during winter. Snowflakes are falling from the sky and there is a small fisher cabin with a campfire in front of the water. In the middle, a few street lambs are highlighting the scene. Appart from that, trees are placed in the scene.

Models:
All models are downloaded from the internet and if necessary, refined in blender. If the models were not in the .obj format, we converted them using blender. All models are downloaded from the website free 3D https://free3d.com/de/.

God Rays:
The god rays are inspired from the following tutorial:
http://chemaguerra.com/circular-radial-blur/
Since this effect is not the cheapest one in terms of render time, only the moon in the scene has god rays. To achieve this effect, we render to a separate output texture only the moon in the original colors, and all other object totally black. Then, in a post processing step, we transform this image into polar coordinates, the center of the coordinates is the screen space corrdinate of the moon. This image is then blurred vertically, such that then back in the cartesian coordinates, the blurred out light shafts of the moon are visible. This blurred image is transformed back to cartesian coordinates.
This final texture is then added to the scene, which gives us the god rays effect.



GPU Particles:
The GPU Particles were implemented similarly to the slides.
For both the snow and the fire, there are 4 SSBOs (2 for positions and 2 for velocities), which are swapped each frame, so we can keep track of the particles.

Due to the fact that we want to randomly spawn particles around a point this pseudorandom function was incorporated into the compute shader. Additionally we have a spread parameter, which is multiplied to this random number.
New particles are spawned in the following way: We calculate an amount (N) of particles which should be calculated in one call of the compute shader. The first N threads spawn a new particle besides processing the old position and velocity.
The positions calculated by the compute shader are extended to a full quad utilizing a geometry shader. For the particles textures we used a texture atlas, where 2 consecutive texture coordinates are calculated and blended.
The particle fire is located at a fixed point, namely the firepit. The snow particle system spawns snow-particles around the translation location of the camera. For the Snow ParticleSystem we spawn 10000 particles per second and the maximum particle Age (TTL) is 10 seconds. (The snow particle system is implemented in GPUParticleMaster and the Fire particle system in FireParticleMaster)

Libraries:
For the window management the library GLFW is used. For the OpenGL calls the library GLEW is used. The Open-Asset-Importer-Lib is dealing with the model loading.
In order to load the textures, we are using the library STB image. To simplify the calculations concerning the vector and matrix maths the library GLM is used.

Testing:
We tested our application on the upper GPU in the computer graphics laboratorium (nVidia).