Documentation 2nd Submission

Controls:

The player (a flying rabbit) can be controlled with the arrow-key. Controls are like those of a flight-simulator: with UP the nose goes down and with DOWN it goes up. RIGHT and LEFT is “rolling” the character left and right. Controlling the game is reduced on the keyboard. The character itself is going forward in predefined pace (the more the nose points down, the faster is the rabbit) and the camera is following in a defined angle that the player has a good overview of the world.

Description of implementation:

- Complex Objects: we have several very complex models in our game. The first one is the player, which consists of a lot of faces, due to get more details. Other Meshes are the terrain. The skybox and the rings are really simple objects.
- Animated Objects: the rings are turning around and the individual parts are spinning as well around their own axis.
- View-Frustum-Culling: we implemented this with Object-Bounding boxes. In our case it doesn’t help the performance a lot, because the big terrain is one mesh. On the other hand it is very helpful for the circles and the ports. View Frustum Culling is also used while rendering the shadow maps. -> (included with F2)
- Transparency: to include this effect, we decided to make “ports” in the rings (like in stargate). Transparency of those can be turned on/off with F9.
- configFile: to set the screen resolution of the game, the width and height of the screen has be written in the config.txt. Must be done, before starting the game. It is also possible, to play the game in full screen and that can be reached by writing an “F” at the beginning of the file. In that case, you also have to include the width and height cause there can occur problems with the beamer.
- KeyMapping for infos: by pressing F2, the Frame Time is displayed and the triangles Count as well. (for now only if resolution is 800x600), some how the text disappears with other resoulution.

Features of the game:

The aim is, that the player should feel as if he is flying downwards a mountain. To get some points he has to pass the rings. On the right side, there is an altimeter, where the height of the player is displayed. In the lower left area the points are visible. If the player reached the water, he can make a plunge in the water and as a result, the game is over!

Collision detection is working but the end of the game /flight is not yet included.

Illumination:

The world has one directional light (the sun). For the different types of Objects and their view, we have some shaders like for example a transparentShader for the ports as well as shaders for the HUD. (simpleTextureShader, …) Regarding to the textures all models have their own apart from rings and ports.

Additional libraries we used:

Assimp (http://assimp.sourceforge.net/)	…	for loading models
CImg (http://cimg.sourceforge.net/)	…	for loading heightmap
FreeImage (http://freeimage.sourceforge.net/)	…	for loading textures
GLEW (http://glew.sourceforge.net/)	…	OpenGL Extension Wrangler Library
GLFW (http://www.glfw.org/)	…	to create window and handle input
GLM (http://glm.g-truc.net/)	…	mathematics library for mat/vec calcs

Effects:

- Cascaded Shadow Mapping (2.5)
- Motion Blur (1.5)

Shadow Mapping:

First we implemented Shadow mapping with PCF and Poisson Sampling as well. This internet side (http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/) provided us with the information.

Because of our lager world PCF and Poisson Sampling wasn't sufficient at all, so we implemented Cascaded Shadow Mapping.

For Cascaded Shadow Map we stick with the paper from Rouslan Dimitrov (http://developer.download.nvidia.com/SDK/10.5/opengl/src/cascaded_shadow_maps/doc/cascaded_shadow_maps.pdf).

By pressing F6 the different buffer are drawn to the screen and the 2nd, 3rd and 4th buffer show the different depth buffers from lights point of view.

By pressing F10 the different frustums are visualized.

Motion Blur:

To simulate a more realistic look, if the speed increases, we implemented motion blur. For this we read following sides:

http://http.developer.nvidia.com/GPUGems3/gpugems3_ch27.html

http://mynameismjp.wordpress.com/samples-tutorials-tools/motion-blur-sample/

http://john-chapman-graphics.blogspot.co.uk/2013/01/per-object-motion-blur.html

We took the last one for our implementation. As describe, we had much background bleeding if the player flew near over the surface, as well as if the player took a sharp curve.

As described on that side, we used the depth of the current fragment position and the depth of the sampled fragment position to decide if it is part of the fragment to blur.

Again by pressing F6 the different buffers are drawn. The first one shows the velocity buffer of the current frame. Due to that the velocity can be very small, the values are scaled only to visualize them.

The velocity can be negative as well, which doesn't show up as a color. For example by flying to the right. Almost the whole right part of the velocity buffer is green (y-velocity > 0), the left part is black, because the y-velocity is smaller than 0.

This is because we calculate the velocity in normalized screen space coordinates.

Tools for creating the models:

The most important tool we used is Blender (http://www.blender.org/) to create and organize the whole world. UV-texturing of some models is done in Blender as well. Only for the rings and their parts we used Maya.

The Game is tested on AMD.