

Amusement Park

Submission 2

Martin Novak, 01425662
Bernhard Langer, 00427400

Controls

Keyboard Commands:

Space: Switches between automatic and debug camera

Backspace: Resets the animation to the beginning

F4: Toggles Postprocessing on/off (default on)

Debug camera controls:

Mouse movement for camera direction.

Mouse wheel to zoom in and out.

Keyboard WASD to translate camera horizontally and vertically.

Engine

Hardware and Build

The Demo targets both Windows and Linux, thus only cross platform libraries were used.

We use CMake to generate a platform and IDE independent solution.

We only support 64bit architecture.

Main development was done on an nVidia GPU, but we also tested it on an AMD.

Scene Description

We use simple text files to describe the scene that we load dynamically at runtime.

These text files contain the models and lights that we use and parameters such as their position, scale and orientation.

Models

A simple model loader is implemented using AssImp.

We included textured models of a ferris wheel, a carousel, a stone tower, a firework volcano and the park terrain itself.

Lighting

For Illumination we have implemented both point and spot lights. There is already some code present for directional lights, but they have not been added to the illumination model yet.

Automatic Camera

Two paths describe the movement of the automatic camera:

One path for the camera position and one path for the camera direction (Point of interest).

The camera paths consists of segments, where each is implemented as a Bezier curve.

For a smoother start and stop (acceleration) we included a sinusoid transfer function.

Various

Textures are loaded using STB (stb_image)

Additionally we included a skybox.

Effects

Particle Effects (Compute Shader Based)

We included two different firework effects:

A firework fountain (aka volcano firework) that constantly shoots particles vertically in the air

Classic firework rockets that generate multiple particles of the same color in an explosion (where each explosion comes in a different color).

Similar to the TTL value in the position ssbo, we take advantage of the unused float in the velocity ssbo where we store a single float and treat it as the hue of the HSV color model.

Volumetric Lighting and Shadows (with PCF)

We included a spotlight mounted on the ferris wheel that lights up the area between the wheel and the carousel. Shadows of the carousel horses can be seen on the carousel floor and the light cone of the spotlight is emphasized using volumetric lighting.

Volumetric lighting is implemented as postprocessing effect and thus only rendered once for each frame. For the volumetric lighting we limited the sampling depth to 20 meters for performance reasons, thus the cone appears as the camera comes close enough.

Screenspace Reflections and Refractions

We implemented both reflections and refractions for a lake of the scene and included a sunken stone tower model to emphasize the refractions below the water surface. The reflections of the tower and the mountain behind the lake and said refractions can be seen in the beginning of the animation. We first use a fixed step size to find the rough intersection coordinates and then refine them using a binary search. This allows a relative high quality intersection even on further away objects, while reducing the average amount of samples needed. We have both single and double refractions. Single refractions are used for the lake and is illustrated by the tower, showing the subsurface refraction. Double refractions are used for the sphere to demonstrate the internal refraction of glass. To hide the cuts where sample lookups leave the image area, results are fading out towards those edges.

Ad Postprocessing

For postprocessing we render the scene into a framebuffer and render its content using a postprocessing shader. Both volumetric lighting and screenspace reflections and refractions are implemented in this shader. When postprocessing is turned off (F4), the scene is rendered directly to the screen instead of the pp framebuffer.

Resources and References

GLFW and GLEW for OpenGL integration and window management

GLM for mathematics <https://glm.g-truc.net>

AssImp for model loading <http://www.assimp.org/>

Stb for texture loading <https://github.com/nothings/stb>

The textures were taken from the total textures repo provided by the course as well as created by hand using GIMP.

All models were created using Blender.

The skybox was borrowed from <http://www.custommapmakers.org/skyboxes.php> (GNU GPL2 license)

Particle Effects are based on the implementation given in the RTR repetition lectures for compute shaders and particle systems.

Volumetric Lighting Effects are based on the implementation given in the RTR repetition lecture of the same name.

Shadow maps with percentage closer filtering and framebuffer implementation are based on the tutorials given by <https://learnopengl.com/>.

SSR is based on the corresponding slides from the RTR lecture for screenspace effects and Wyman, Chris. "An approximate image-space approach for interactive refraction." ACM transactions on graphics (TOG) 24.3 (2005): 1050-1053.

Some code for serialization was taken directly from my personal project.
For reference while building the engine we used the tutorials from <https://learnopengl.com/> and <http://www.opengl-tutorial.org/>.