Real-Time Rendering
Final Documentation

Group 33 “The Way of Light”
11776187 Martin Crepaz
11778247 Gabriel Ratschiller

Technical Basis

For our project we have used C++ and GLSL as programming languages together with OpenGL as a graphics API. We have built our application on our own framework and used following libraries:

- Assimp for loading models into the scene
- GLFW for context and window management
- GLM for mathematics related stuff
- GLAD for managing function pointers
- irrKlang for playing sounds in the application

Configuration parameters

Application parameters can be adjusted/toggled in the settings.ini file that includes following parameters:

- Width, height of the window
- FPS
- Fullscreen/Windowed
- Brightness of window
- Debug camera or automatic camera
- FOV, Near/Far plane of camera
- Time after which the automatic camera starts

Effects

We have implemented following effects in our scene:

- Shadow mapping with percentage-closer filtering[1]
- Omni-directional shadow mapping[1]
- CPU particle system
- Bloom

We have various light sources in the scene, including a directional light and several point lights. The directional light is implemented with shadow mapping and PCF, the effect can be

[1]complex effects

observed in the shadows cast by the buildings. Furthermore, the omnidirectional shadow mapping is visible in the shadows of the streetlights, the fireball and the bonfire.

Additional non-complex effects are CPU particles and bloom. For the automatic camera movement we implemented the camera path using a Bezier-curve interpolation.

Shadow mapping with PCF

The first step involves rendering the scene from the perspective of the light source and creating a depth map. This depth map contains information about the distance from the light source to each point in the scene. During the main rendering pass, for each pixel on the screen, the depth value is compared with the corresponding depth value in the shadow map. If the depth of the pixel is greater than the value stored in the shadow map, it means the pixel is in shadow; otherwise, it is lit. To address aliasing issues and create smoother shadows, PCF introduces additional samples around each pixel in the shadow map. Instead of just comparing the depth values at the center of each pixel, multiple samples are taken within a neighborhood around the pixel. The depth values from the neighboring samples are then averaged, and the result is used to determine the final shadowing intensity for the pixel.

Omni-directional shadow mapping

Omni-directional shadow mapping is used to render the shadows of objects illuminated by a pointlight. The first step is to render the scene from the perspective of the point light source and generate six images (one for each side of a cube) to create a cube map.The depth information of the scene is then extracted from each of the six cube images. The result is a depth cube map from which a 2D shadow map is created. During the main rendering pass the depth values of the scene (from the perspective of the camera) are compared with the values stored in the shadow map. If the depth of a pixel is greater than the corresponding value in the shadow map, this means that the pixel is in shadow. This information is then used to determine the darkness or intensity of the shadow at this point.

CPU particle system

This effect was employed to generate the sparks in the bonfire. To achieve this, small particles are generated, and their positions and velocities are randomly adjusted upward and to the side, simulating a realistic flight of sparks. At each iteration, the system calculates the new state of each particle based on its current state and the rules that govern the behavior of the particles. Additionally, the shader modifies the color of the particles as they move through the air. Each particle is assigned a specific lifespan, predetermined at the outset. To enhance realism and control the density of particles, four different lifespans are utilized. This variation ensures that individual particles ascend higher into the air than others, preventing an excessive number of particles from existing simultaneously.

Bloom

The Bloom effect is used to create a glowing effect. This effect can be seen in the glowing fireball, the fire particles and the glasses of the streetlights. The first step is to render the entire scene as usual. A threshold is applied to the rendered image to identify high intensity pixels that normally correspond to bright light sources. These high intensity pixels are used

for the Bloom effect. Then a Gaussian blur is applied to the image, which transfers the intensity from the bright pixels to the surrounding areas. The blurred image is combined with the rendered original image using a combine shader.

Models, Textures and Sounds

Models

- “Medieval village part 1 Free low-poly 3D model” by runsystem

Textures

- Ground: “Floor 06” from 3dtotal.com
- Skybox: From OpenGameArt.org

Sounds

- Background: Sound effect "Night Ambience" from Pixabay
- Bonfire: Sound effect "Fireplace with crackling sounds 2 min. RK" by RonKoster2023 from Pixabay

Button Controls

Several buttons are defined that can be clicked to influence the scene and switch effects on and off. The following list describes which buttons can be pressed. Some buttons can only be clicked (or only make sense) if the debug camera is activated in the settings.ini. For example, with the debug camera, the fireball path can be started at the click of a button, whereas this button would make no sense with the automatic camera, as the fireball path is triggered automatically and the scene would otherwise be unsynchronized.

ESC - close window

W,A,S,D - control debug camera

Arrow Up, Arrow Down - move debug camera up and down

B - toggle bloom effect

F - start the fireball path

M - toggle all sounds

Hardware

We have tested the application on the following graphics cards:

- NVIDIA GeForce RTX 3070
- NVIDIA GeForce RTX 2060