Simon Maximilian Fraiss BSc., 01425602
Robert Maximilian Fischer BSc., 01425684

Final documentation “transition”

- **Effects**: See below for more detailed description
    - Bloom
    - GPU Particles using Compute Shader
    - Volumetric Lighting (directional, spot lights and omni directional)
    - Shadow Mapping (Directional- and Omni-directional)
- **Camera Movement**: Camera is moving along a predefined Catmull-Rom spline and camera rotation is interpolated using quaternions to specified target positions.
- **Model Loading:** We load most of our level with Assimp from a Collada file.
- **FPS**: Our testing on the Vislab PCs had 200 FPS average (lowest measurement was at around 70 FPS)
- **GPU:** We tested it in the Vislab on the Nvidia graphics card. Additionally our development machines were running Nvidia GTX 1080 and Nvidia GTX 1060

Shortcuts

- **PAGE UP**: Increases speed of animation, but music speed is not increased, so it will get out of sync
- **PAGE DOWN:** Decreases speed of animation, music speed is not decreased
- **ESC**: Terminates the demo

Additional Libraries

- GLFW: http://www.glfw.org/
- GLAD:: https://github.com/Dav1dde/glad
- Assimp: http://assimp.sourceforge.net/
- GLM: https://glm.g-truc.net/0.9.8/index.html
- FreeImage: http://freeimage.sourceforge.net/
- IrrKlang: https://www.ambiera.com/irrklang/
- SplineLibrary: https://github.com/ejmahler/SplineLibrary

Special Tricks and Information

- The piano is actually playing the notes of the music
- We perform view frustum culling. Additionally we only render rooms that are necessary
- A Keypoint based animation system was implemented, were we can define how the camera moves and where it looks at, for a specific duration - combined with an action system were certain actions happen at certain keypoints.
- The music is composed by Simon Fraiss
- There is a car sound playing in the background, when the car drives by
- The point light in the first room is “flickering” simulating bad electricity
- One of the photos in the living room is actually us (Simon Fraiss and Robert Fischer)
- There are various (partially) transparent objects in the scene, such as the feet, the particles or the leaves.
- The point light in the first room is shortly swinging once it gets bright

Simon Maximilian Fraiss BSc., 01425602
Robert Maximilian Fischer BSc., 01425684

Volumetric Lighting

Volumetric Lighting can be seen in the first room as fog around the light bulb as well as light shafts that come in through the window. Light shafts can also be seen in the end at the tree.

Volumetric Lighting is implemented using Raymarching techniques, since it is the most interesting and flexible approach available. The main idea of this algorithm is to ray march for each fragment and accumulate the light function in each step. Problem with this

approach is that is computationally very expensive, so it must be optimized using several techniques.

The following steps are facilitated:

- Render scene using Forward Rendering into a color buffer and a texture buffer using FBOs
- Downsample depth buffer of the scene using a depth aware downsampling in a chessboard pattern to half resolution
- Do the ray marching on the half resolution depth buffer:
    - For each pixel and each light:
        - Find start- and end position of the ray in world space. But since this effect has low frequency, the starting position can be dithered by a bayer matrix, so less samples are needed
        - Convert start- and end position of the ray from world space into light space
        - Sample n times the lighting function times a fog function to get the fog (a 3d noise was used)
        - Add the result to the volumetric lighting buffer
- Do a depth aware Gauss on the volumetric lighting buffer, to hide the fact that we were dithering the initial ray marching
- Do a depth aware upsampling to full resolution

This technique needs around 16 samples per pixel to get convincing results (depending on several parameters, like far plane and geometry) The up- and down sampling and the gauss are done depth aware, to keep the edges of the volumetric lighting sharp.

Possible improvements:

- Limit rays: Currently the whole ray from view position to the far plane is being sampled for each light. But only samples inside the light volume have impact on the volumetric lighting. Doing this optimization efficiently needs a light volume, which we do not have, so we opted out.
- Compute Shaders: Doing the ray marching in compute shaders might improve the performance even better.
- Multiple light scattering: Currently the system has the assumption that light gets scattered only once. There exist approximations to get multiple light scattering.

Sources:

- https://www.youtube.com/watch?v=RdN06E6Xn9E (The game Inside facilitates a very nice looking volumetric lighting effect, which inspired this effect)
- https://www.youtube.com/watch?v=lz85j8bnChA (general algorithm and the 3d noise idea)
- https://www.shadertoy.com/view/4ts3z2 (3D noise implementation)
- https://github.com/SlightlyMad/VolumetricLights (some inspiration regarding open implementation details)
- http://www.alexandre-pestana.com/volumetric-lights/ (Bayer matrix idea)

Simon Maximilian Fraiss BSc., 01425602
Robert Maximilian Fischer BSc., 01425684

Shadow Mapping

The car light and the light behind the tree are implemented using spot lights using directional shadow mapping and the light of the living room is using a point light using omni-directional shadow mapping.

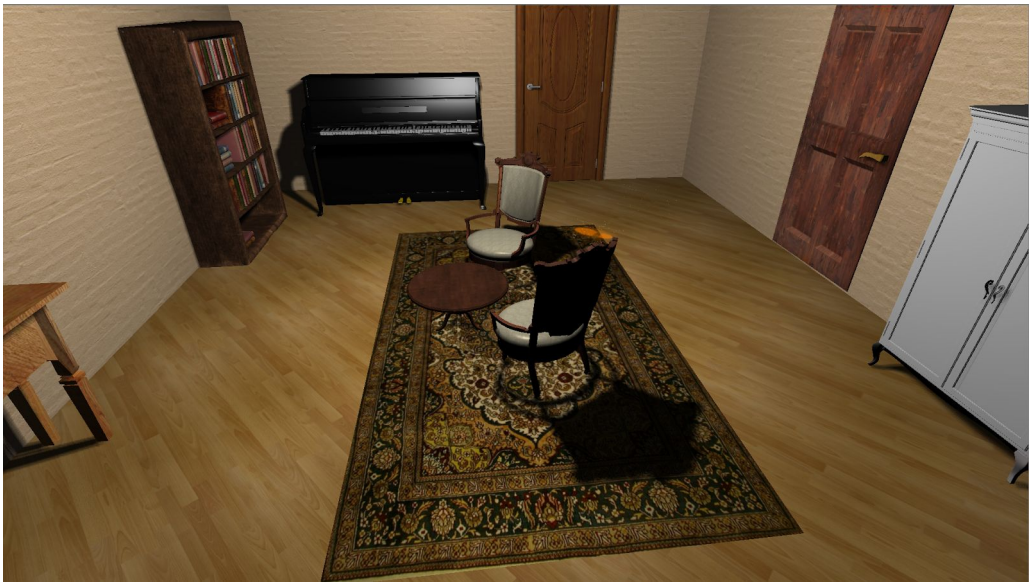
Our demo implements classical directional and omni-directional shadow mapping using percentage-closer filtering. They can be seen pretty much everywhere (the tree also has shadows, however they can hardly be seen because of the bright lights around).

For the depth map of the omni-directional shadow mapping the scene is rendered onto a cubemap using a single draw call by a geometry shader.

Culling is performed for rendering the depth buffer.

Sources:

- CGUE 2017 entry: Regime Runner
- EZG lecture
- https://learnopengl.com/#!Advanced-Lighting/Shadows/Shadow-Mapping
- https://learnopengl.com/#!Advanced-Lighting/Shadows/Point-Shadows



Simon Maximilian Fraiss BSc., 01425602
Robert Maximilian Fischer BSc., 01425684

Bloom

For the bloom effect, the brightest parts of the image (usually only those with a brightness of 1.0) are extracted, blurred in an extra shader and added to the original image.

In the scene, Bloom can be seen on the lamps, especially in the first room (note that the intensity of the lamp is varying, so the bloom effect disappears every now and then). When the tree area is entered, bloom fades out to avoid overusing. When the camera moves upwards through the leaves, Bloom is increased again.

Sources:

- CGUE2016 entry: Unseen
- EZG lecture
- https://learnopengl.com/#!Advanced-Lighting/Bloom

GPU Particles

A particle system was implemented using Compute Shaders. The particles can be seen every time a foot step is flashing as small circles that rise from the foot. Also, particles can be seen on the floor in front of the tree, as small green circles rising from the floor.

Two position, velocity (moving direction + speed) and color SSBOs were used. The compute shader updates the positions according to the velocity and the delta time. The speed is reduced each frame. Also the moving direction is slightly adapted to become more and more vertical (can be seen in foot_part.comp).

Sources:

- EZG repetitorium

Simon Maximilian Fraiss BSc., 01425602
Robert Maximilian Fischer BSc., 01425684

Models/Textures/Sound

We mainly used Blender for the modelling the scene. Souces for the models:

- https://www.turbosquid.com
- https://free3d.com/
- https://www.cgtrader.com/
- CGUE 2016 entry: Unseen

Sources of the textures:

- https://lva.cg.tuwien.ac.at/textures/
- https://www.flaticon.com/
- Own photographs

Sources of the sound:

- https://www.freesoundeffects.com/free-sounds/cars-10069/
- Music: self-composed

Git

Please note, that we use Git-LFS on our github-repository so in case you should clone it, make sure to have LFS installed!