# Spooky house

Real-time rendering prototype

Group 5
https://github.com/Welko/ezg18-spookyhouse

| | | |
|---|---|---|
| Lucas da Cunha Melo | 01429462 | e1429462@student.tuwien.ac.at |
| Elitza Vasileva | 01426939 | e1426939@student.tuwien.ac.at |

## Brief description of the implementation

In our demo, the following features were implemented

- 3D rendering with OpenGL 4.5 (tested on Windows 10 64-bit).
- Effects:
    - Specular Mapping
    - Normal Mapping
    - Omni-directional Shadow Mapping
    - Range-Based Fog
    - Ambient Occlusion (SSAO)
    - Bloom
    - HDR
    - Volumetric Fog
    - GPU particle effects
- Camera movement: Camera is moving along a predefined path consisting of some key positions and directions and using linear interpolation between them.
- Model loading: Our scene is one big model which is loaded using Assimp from a Wavefront (.obj) file.
- Animations: There are some moving objects in the scene, which were created by applying simple matrix transformations.
- A simple illumination model. Including the following components:
    - Ambient illumination
    - Diffuse reflection
    - Specular reflection
    - Multiple light sources on different positions

For debugging and to explore the scene, a controllable camera was also implemented, which is currently disabled due to the automatic camera movement.
Its controls are listed and described in the table below.

| **Controls** | | |
|---|---|---|
| `W` | Move camera | Forward |
| `A` | | Left |
| `S` | | Backward |
| `D` | | Right |
| `Q` | | Down |
| `E` | | Up |
| `Mouse movement` | Rotate camera | |

## Shortcuts

| | | |
|---|---|---|
| `F1` | Specular mapping | (OFF, **ON**) |
| `F2` | Normal mapping | (OFF, **ON**) |
| `F3` | Bloom | (OFF, **FRAG**, COMP) |
| `F4` | HDR | (OFF, **LINEAR**, DRAGO, DRAGO MODIFIED) |
| `F5` | Shadow mapping | (OFF, **ON**) |
| `F6` | Fog | (OFF, **ON**) |
| `F7` | Ambient occlusion | (OFF, **ON**) |
| `F8` | Bubbles | (OFF, **ON**) |
| `F9` | Fire | (OFF, **ON**) |
| `F10` | Rain | (OFF, **ON**) |
| `U` | Increase auto-camera speed | |
| `I` | Decrease auto-cameras speed | |
| `SPACEBAR` | Camera mode | (FREE, **RAIL**) |
| `ESCAPE` | Quit | |

## Special Features and Information

- The animation is based on keypoints, so that at each of this key points we define the camera position and where it looks at.
- The model of the house was actually modeled in Blender by following a tutorial in YouTube.
- All the normal and specular maps were created by ourselves using the program Crazybump.
- There are sound effects playing at certain keypoints of the camera movement.
- To clone our repository you must use git LFS.
- We perform frustum culling not only for the camera (so objects outside of the field of view are not drawn), but also for each of the six directions of a point light that emits shadows (for shadow mapping).
- It starts to rain at the end of the demo.

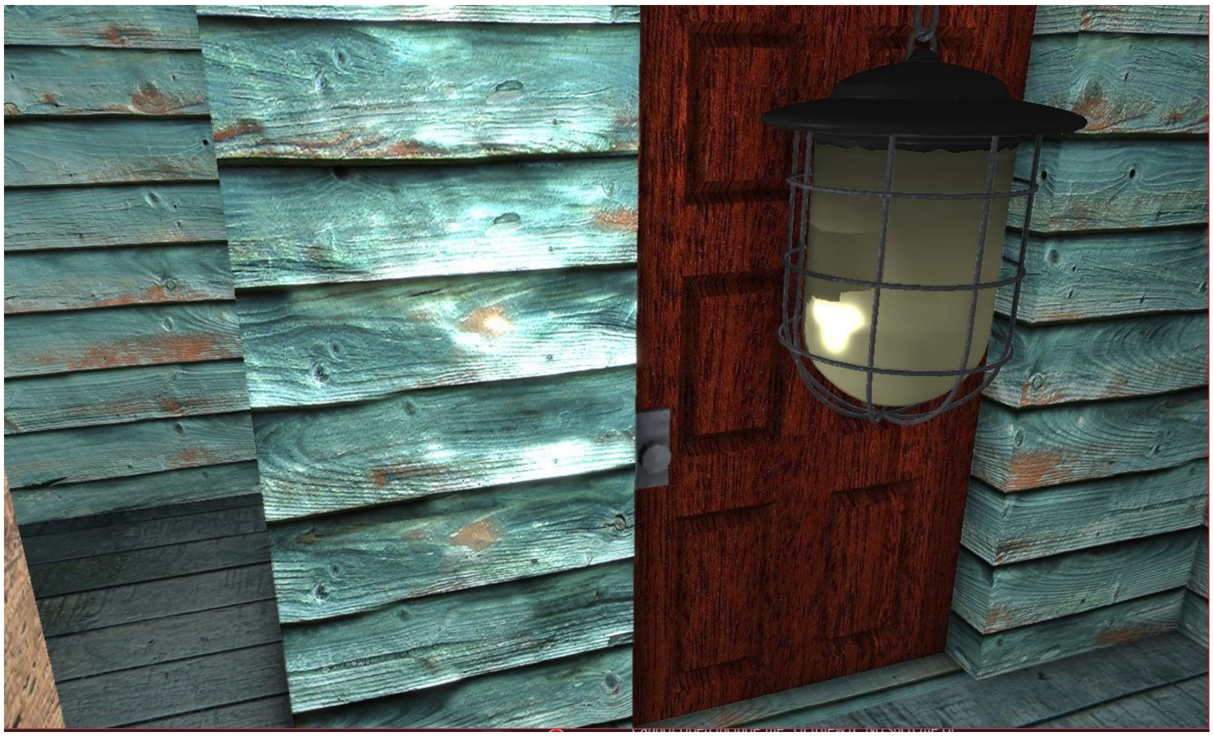
## Additional libraries

The libraries used in the prototype are listed and described in the table below.

| **Library name** | **Use** | **Link** |
|---|---|---|
| GLFW | Window creation, management and input for OpenGL | https://www.glfw.org/ |
| GLEW | Query and load OpenGL extensions | http://glew.sourceforge.net/ |
| GLM | Mathematical objects and functions based on GLSL | https://glm.g-truc.net/0.9.9/index.html |
| stb_image | Image loading into memory | https://github.com/nothings/stb/blob/master/stb_image.h |
| Assimp | 3D models loading into memory (meshes and materials) | http://www.assimp.org/ |
| IrrKlang | For loading and playing music | https://www.ambiera.com/irrklang/ |

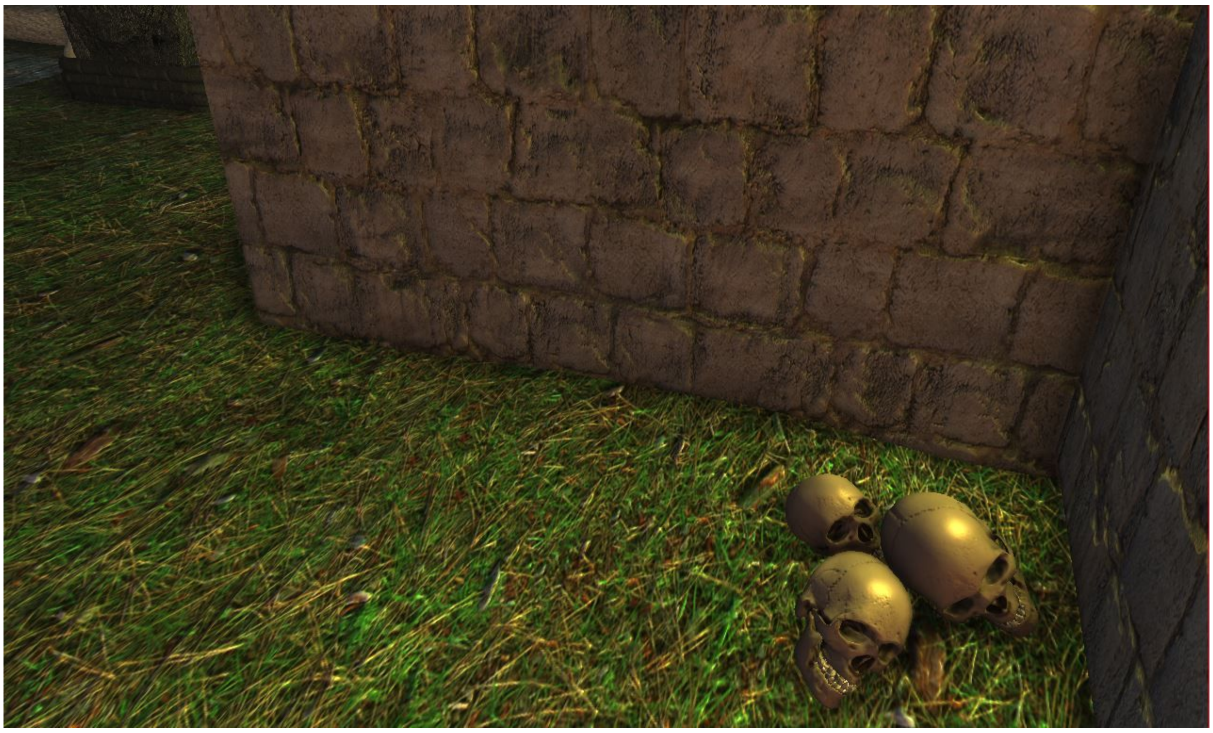
# Effects

## Specular Mapping



Textures are used to specify different specular values for an object at specific positions.

## Normal Mapping



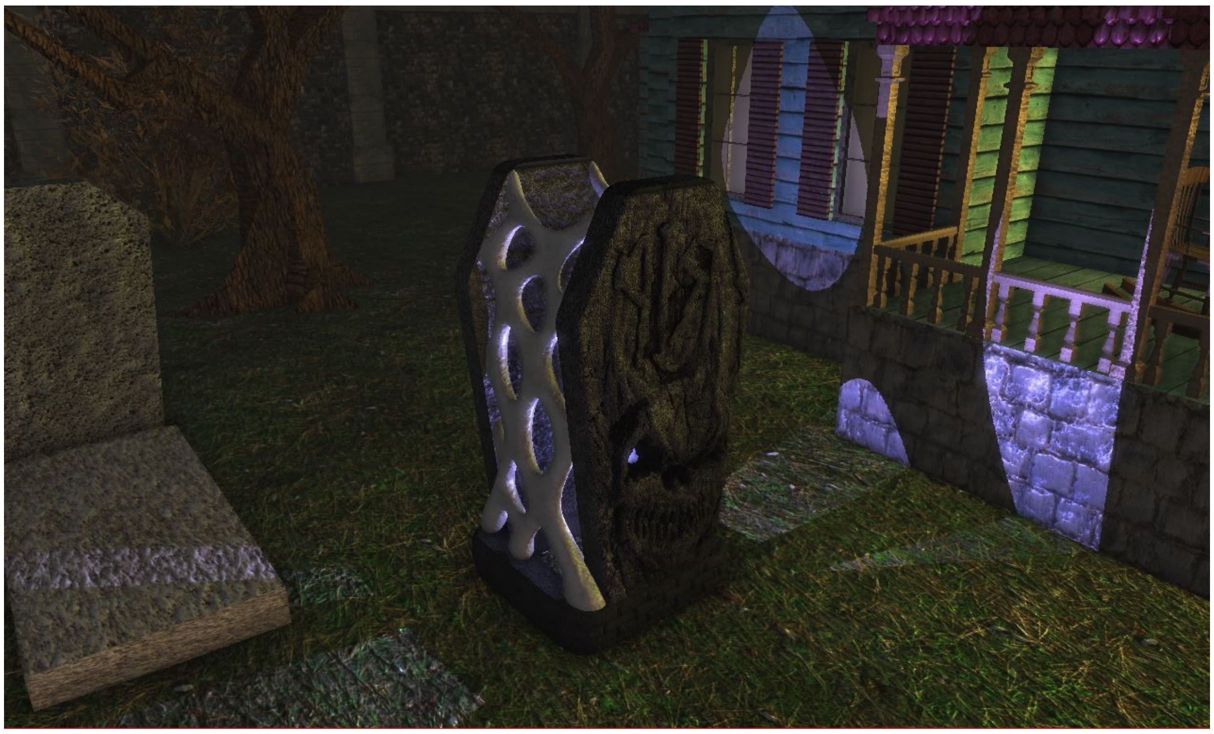
Textures are used to specify different values for the normal vector for an object and thus create the effect of rough surface even though the surface is flat.

Currently, the engine must have normal mapping switched on. Otherwise, the textures will glitch.

Sources:

- EZG Lecture
- https://learnopengl.com/Advanced-Lighting/Normal-Mapping

## Omni-Directional Shadow Mapping

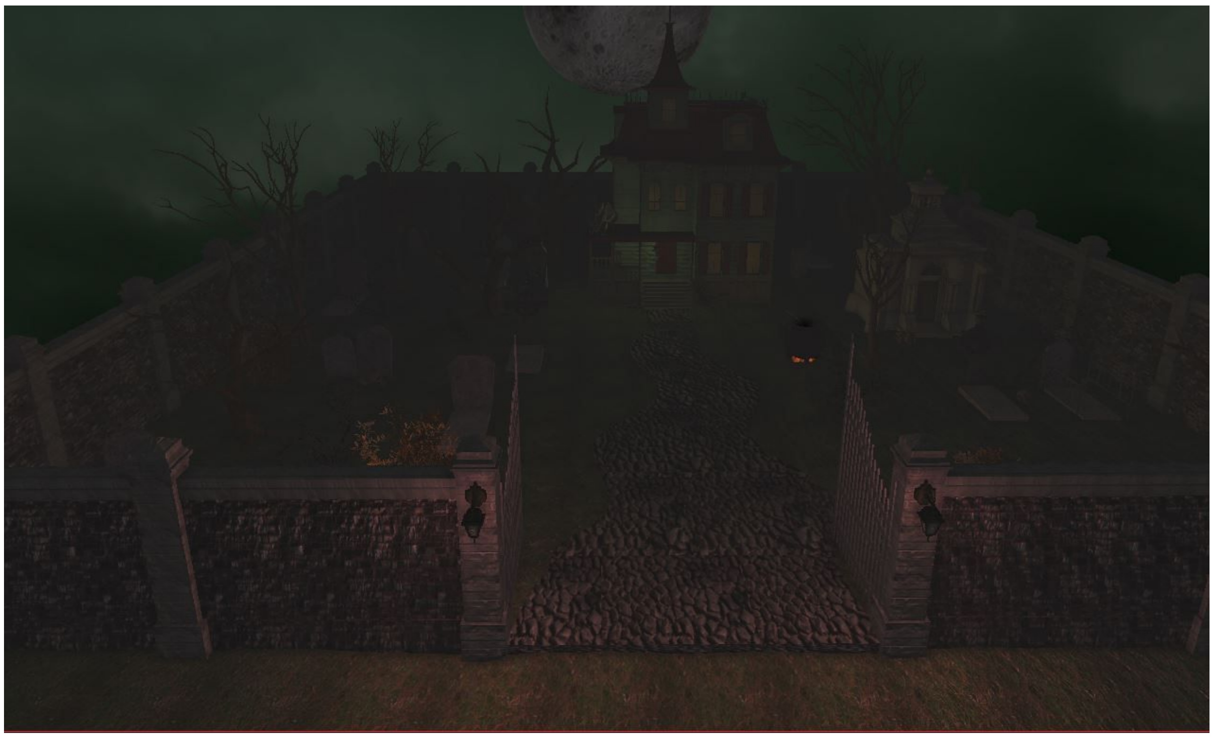


A cube texture is calculated from the point of view of a light source, where the 6 faces represent the “point of view” of the light and render only the depth values of the scene. These are then used to calculate whether a fragment is illuminated or not by objects.

Sources:

- EZG Lecture
- https://learnopengl.com/Advanced-Lighting/Shadows/Point-Shadows

## Range-Based Fog



A range-based fog was implemented by using an exponential function to define the strength of the fog depending on the distance of the camera from the objects.

Sources:

- http://in2gpu.com/2014/07/22/create-fog-shader/

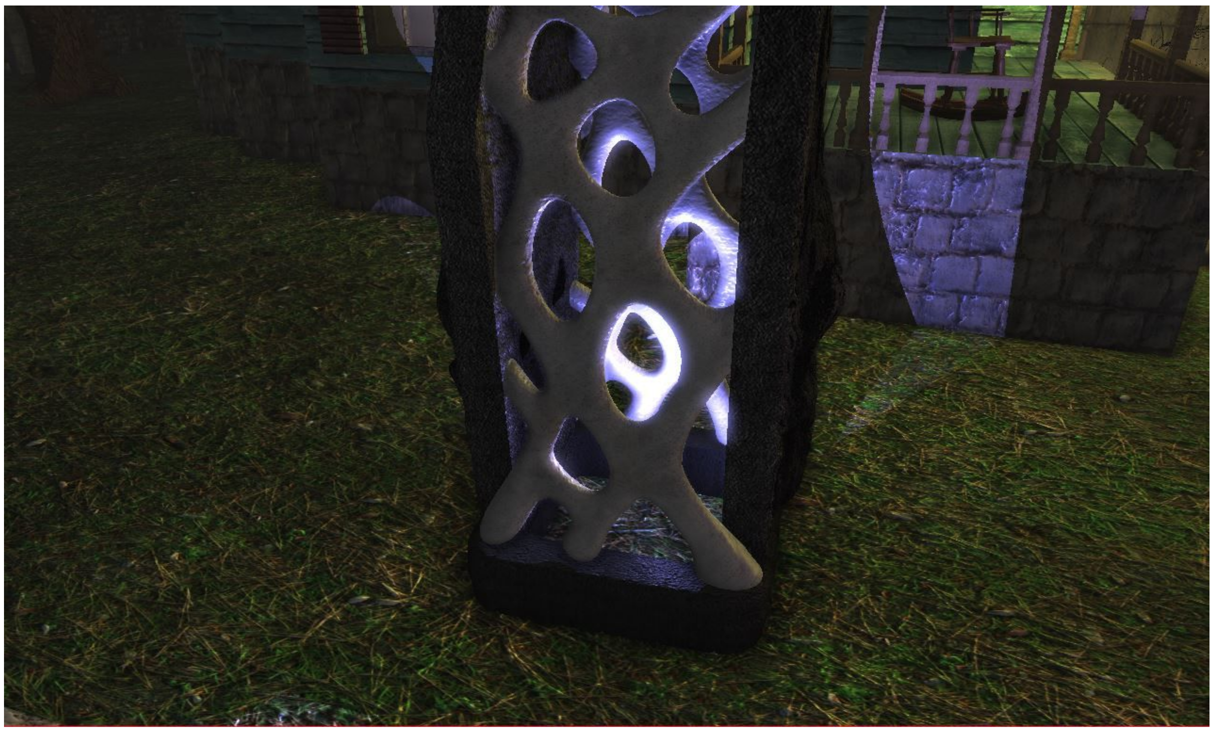
## Ambient Occlusion (SSAO)

We use ambient occlusion to simulate the scattering of light in our scene, so that places where the different objects are close to each other look darker. We use two framebuffers - one to calculate the occlusion and one to apply a blur effect on the ambient occlusion texture in order to reduce noise.

Sources:

- http://john-chapman-graphics.blogspot.com/2013/01/ssao-tutorial.html
- https://learnopengl.com/Advanced-Lighting/SSAO

## Bloom



Two different approaches for bloom were implemented. One using fragment shaders, where the scene is filtered for values above a certain threshold, which are then stored in a separate texture. This texture is then blurred in a fragment shader by sampling points in between texels with linear interpolation, achieving an efficient blur in less passes.

The second blur alternative, using compute shaders, was inspired by the technique mention in the Repetitorium, where, within a workgroup, calculated values can be reused.

Source:

- EZG Lecture and Repetitorium

## High Dynamic Range

HDR was implemented following some models presented in [Luksch 2006/7], with some special attention given to Drago's method [Drago et al. 2003]. All implemented models (Linear, Drago and Drago modified) use compute shaders to calculate statistical values about the scene (such as max values and average) and uses them to recalculate the brightness of the scene in another compute shader.

Sources:

- EZG Lecture
- "Realtime HDR Rendering" Luksch 2006/7
- "Adaptive Logarithmic Mapping For Displaying High Contrast Scenes" Drago et al. 2003

## Volumetric Smoke (Bubbles)

The volumetric smoke was implemented with an Eulerian grid, where each voxel has a velocity vector and one additional parameter (pressure) that is later used for the rendering. The fluid is simulated based on the model presented in [Stam 1999], but several other publications were useful for more straightforward explanations and implementation details.

The simulation itself takes places in a compute shader and is achieved through several passes. The rendering is achieved by rendering the front and back faces of a volume and applying a ray-marching technique through the material.

This effect could however not be implemented to its full capabilities. The fluid simulation seems to not be working as expected, so a simplified use was found (namely, bubbles effect). Additionally, the gradients of each voxel are calculated to help improve the looks of the effect. These, however, seem to be wrong as well. A buoyancy model was not implemented, which would greatly improve the quality and realism of the simulation [Rideout 2010, Rideout 2011], and a collision model is missing as well [Crane et al. 2007]. It is also noticeable that the smoke does not take into account the depth buffer of the scene, often being drawn behind objects where it should be intersecting with them.

Sources:

- “Stable Fluids” Stam 1999
- “Real-Time Simulation and Rendering of 3D Fluids” Crane et al. 2007
- “Simple Fluid Simulation” Rideout 2010 (https://prideout.net/blog/old/blog/index.html@p=58.html)
- “3D Eulerian Grid” Rideout 2011 (https://prideout.net/blog/old/blog/index.html@p=66.html)

## Particle effects (Fire, Smoke and Rain)

Particles effects are used in our demo to create the effect of fire (with smoke) and rain. They are created and managed inside a compute shader, where their positions, velocities and angle are calculated.

For the rendering, the points calculated in the compute shader (stored in a buffer) are used to create quads in a geometry shader and rendered with textures (for fire and smoke) or they are used to create blue triangles (for rain).

The smoothing of the particles (“soft particles”) is, however, missing. It was planned to be implemented as in [Lorach 2007], but was dropped due to time constraints.

Sources:

- EZG Repetitorium
- “Soft Particles” Lorach 2007

## Hardware

The demo was tested in the lab environment (VIS LAB) and had an average FPS of over 60. The prototype was also tested in two machines under two GPUs of different manufactures on Windows 10. The tested GPUs are the following:

- AMD Radeon R5 M330
- NVIDIA GeForce GTX 1050 Ti

## Sources

For the C++ and OpenGL code, the following tutorials were followed, and can therefore share similar code with our prototype:

- Learn OpenGL: https://learnopengl.com/
- opengl-tutorial: http://www.opengl-tutorial.org/
- TheChernoProject: https://www.youtube.com/user/TheChernoProject

For the textures, the following libraries were used:

- Textures.com: https://www.textures.com/
- Total Textures: https://www.cg.tuwien.ac.at/courses/Textures
- Google

Sources of the models:

- https://archive3d.net/
- https://www.turbosquid.com/
- https://www.cgtrader.com/
- House model tutorial: https://www.youtube.com/watch?v=0u1PSx9CcSM

Sources of the sound:

- Some of them were downloaded from https://www.youtube.com/
- https://freesound.org/browse/tags/sound-effects/