Submission 3: Heartbeat

Martin Rupp (11709466)
Michael Franz Landauer (11778912)

Controls:

- **WASD:** Camera Movement
- **Left mouse button:** Rotate Camera
- **Right mouse button:** Camera Panning
- **Mouse wheel:** Zoom in/out
- **ESC:** Exit Application
- **F1:** Toggle free camera
- **F2:** Restart demo
- **F3:** Draw debug info
- **F4:** Draw white sphere in the middle of the screen
- **F5:** Toggle between gauss- and bilateral-filter for volumetric light
- **F11:** Toggle fullscreen
- **P:** Pause time
- ← → **:** fast backward/forward (can only be used when paused)

Settings.ini:

Is loaded when starting the demo.

```
width=1920      # Render width in pixel [int]
height=1080     # Render height in pixel [int]
fullscreen=1    # Fullscreen [bool]
monitor=0       # Monitor index for fullscreen [int]
draw_debug=0    # Draw debug information [bool]
```

Demo

Libraries:

Proposal: We will use C++ with OpenGL 4.5 Core Profile on Windows Platform. We plan to program our own framework from scratch. Based on our effects we will use Assimp for model loading, stb for image loading, GLFW/glad for setting up the OpenGL context, glm for math and OpenAL for music.

Everything stayed the same except for the use of OpenAL, which we replaced with irrKlang.

Testing:

Tested on 2 PCs with the same GPU NVIDIA GTX 1070.

Scene:

The current Scene consists of a cave, stalactites, rocks, voronoi bunny, brick wall and a dragon. All of these models are drawn with glDrawInstance and their position is read out of a model matrix file. This file can be changed at runtime and gets hot reloaded. 3 Spheres (point lights) in different colors illuminate the scene, these spheres are also animated in the vertex shader and get blurred by the bloom shader.
Most of the models contain a diffuse map, normal map and a specular map. For the temporal positions in the scene we use b-splines and a timecontroller (located in ressources) to visualize the animations during the demo, these can also be updated at runtime.

Effects:

We implemented 3 Major Effects, which are Omni-directionals shadow maps, relief mapping and volumetric light.
Other effects we implemented are: Bloom, gamma correction, specular and normal maps.
All shaders are hot reloaded when saved, so it's easy to change parameters like tau/phi for volumetric light or the displacement factor for relief mapping.

Omni-dir shadow maps:

We implemented omni-dir shadow maps for 3 point lights in the scene. Most of the code is very similar to the code in the lecture presentations. In addition we implemented pcf with 16 samples. The shader for rendering the cubemaps is named “omni_shadow.glsl” and the sampling can be found in “basic.glsl”

Relief Mapping:

Relief mapping is used on the wall to the right of the stanford bunny. It is implemented in the shader-file “relief_mapping.glsl”, where a ray is traced in the height-field of a given texture. To find the intersection with the height-field, we search for the first intersection using linear steps, and refine the found intersection using binary search. There are multiple adjustable parameters, including the step count in the first linear search step (linera_step_count) and the binary search step count (binary_search_step_count). The depth of the height-field is normalized (zero to one range), and it can be adjusted by the parallax_depth uniform, which is animated during the demo.

Volumetric Light

Our approach for implementing volumetric light is that we first render the scene, so we have the depth information of the scene. As a consequence we can recalculate the absolute position of each pixel, which we need to render volumetric light. We render volumetric light at a quarter resolution (half in each dimension) and as input we have the depth information as texture. The depth information needs to be downsampled, for this we use the min depth of the 4 pixels, we also used a checkerboard pattern but the resulting image looked nicer when using the min value. We use 50 samples for each pixel, but this can be changed in the shader “vol_light_optimized.glsl”, phi and tau can also be changed in the shader. For calculation we use the same formula as in the presentation, expect for the multiplication factor we use the attenuation factor instead of tau. We also use a bayer pattern to avoid

aliasing artefacts. After the calculation we apply a bilateral-filter 2 times and then a gauss-filter, with Hotkey F5 this can be switched to 3 times gauss filter. These both filters are implemented using compute shaders as 2 pass filters (“bilateral_hori_compute.glsl”, “bilateral_hori_compute.glsl”, gaussian_hori_compute.glsl”, “gaussian_hori_compute.glsl”). The bilateral filter uses a linearized depth texture which is calculated inside the vol_light shader.
After this the final image gets composed in the “blend.glsl” shader and upsampled by hardware lerp. Due to color banding effects we also apply a dither noise for the resulting image.

Minor Effects:

We implemented bloom/glow in the “blend.glsl”. In the basic shader we render to an additional texture for those pixels we want to blur. This effect is mostly used for the light spheres in the “light_sphere.glsl” shader. We experimented with a lower threshold factor but it didn't fit well in the scene.
For more details we implemented normal mapping and specular maps to give the cave a realistic appearance. This is done in “basic.glsl”.
We also implemented correct gamma correction and hdr rendering with an exposure time.

Additional Features:

- Shader, B-Splines and Timings for B-Splines Hot Reloading:
  Our shader and bspline files are stored in the ./resources/shaders and ./resources/bsplines directories, and are hot-reloaded by the application. For timing the b-splines a time controller is used with time files under ./resources/times.
- Window resizing
  The application window is resizable and all affected framebuffers are automatically resized.
- Model Matrices:
  For rendering multiple objects with the same meshes we use Instance rendering where we store the matrices in an SSBO. The model matrices are red out at the start of the demo. These model matrices are located in ./resources/model_matrices/ and are encoded in 3 x vec3. Where the first vec3 is the scaling vector, the 2nd the rotation and the 3rd the translation vector.