

Group 20 - OINKED

Deep into space in a parallel universe several thousand lightyears away, there is a planet similar to earth. Similar but with one big difference, the animal kingdom got their hands on human-invented brain interfaces and took over society. One special species decided to use there newly gained brain power to have a good time all day loooooong. Let's oink it up!

In the scene the observer is guided to the center of a surreal pool party, where pigs can fly and floor and ceiling are only abstract concepts. The fly-through starts in a long hallway, where the floor, walls and ceiling are decorated with strange shiny and refractive objects. While the viewer is guided to the main attraction the camera pans through the collection of curiosities in the hallway. At the end of the winding and surreal hallway we come to the pool room, where the pigs meet to party. Here the water in the pool causes wild caustics all over the room. After spending a few moments looking around the room, the demo concludes while the party continues.

IMPLEMENTATION:

The whole scene including all models, lights, cameras, materials, textures and animations is loaded from a single gltf file that was exported from Blender. This is used to create all movements, including the automatic camera movement. Scene loading is realized with a custom gltf loader that uses the tinygltf [1] library as a basis.

Basic shading (for all objects without special effects) is done in the texture_pbr.frag shader, where a Physically Based Rendering model has been implemented [2]. The shadows are omnidirectional point shadows [3] with light originating from four light sources. For the static scene in the entrance hall the shadows are only calculated once at the beginning. In the pool room the shadows are updated every frame.

The water rendering is done by rendering the reflection with a seperate camera. The caustics effect is a voronoi texture that is projected onto the area of the pool that is under water. The voronoi shader is based on the following tutorial [4].

EXTERNAL LIBRARIES:

The following external libraries were used:

- irrklang sound library for music playback [5]
- tinygltf for reading the glTF file [1]

EFFECTS:

The following two effects from our official effect list were implemented:

- **Screen space refraction:**
The screen space refraction effect can be seen in the first room when the camera rotates around the refractive snail model. For this effect the snail is drawn over the rest of the scene. The scene sans snail is drawn beforehand and passed as a texture. For the refraction the normals of the front and back faces of the refractive object are drawn. In the refractive fragment shader the normals are used to refract

Echtzeitgrafik

Bettina Schlager 01327281

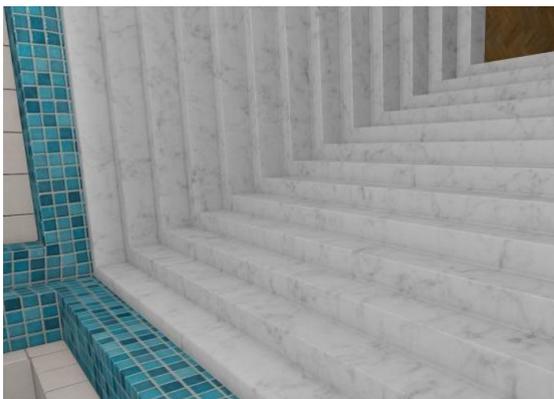
Johannes Eschner 01633402

the uv coordinates when sampling the texture of the scene. This gives the refractive object the desired glass look. In addition to the refraction the object is also slightly darkened to account for light loss in the glass. For this effect the following resources were used [6] [7]

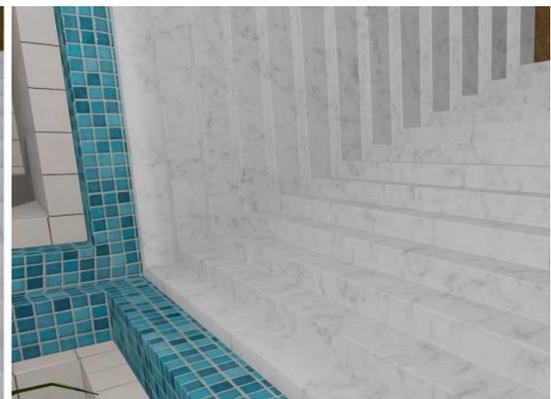
- **Screen Space Ambient Occlusion:**

For more scene depth and realism we decided to add screen space ambient occlusion to our scene. (F5 on/off)

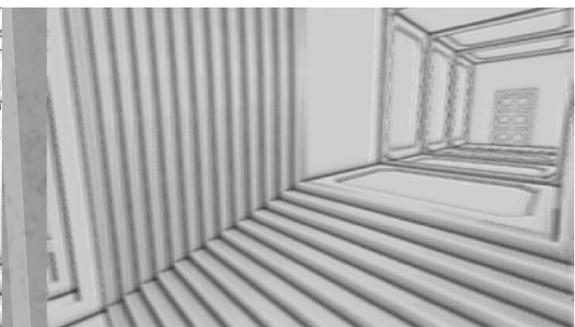
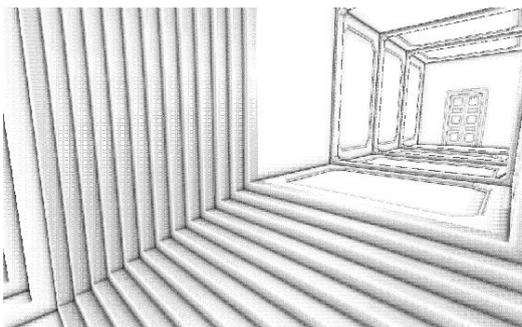
The effect uses three framebuffers. For the first one we generate two textures in screen space where view-space normals and positions of the current camera view are stored. Those textures are then used to estimate the occlusion per frame in the SSAO shader. The normal vector creates a hemisphere per fragment and takes random sample for depth comparison within the hemisphere. A sample point darkens occlusion value for this fragment if it is in front of the geometry (the sampled position) In our implementation we use a random sample kernel and the sample size per frame, accepted radius, bias and strength of the occlusion calculation can be adjusted via the settings file. The shader writes to a texture with values between 0 and 1. The result of the AO map highly depends on sampling, so to remove noise created by the random vectors, the texture is then blurred with horizontal and vertical gaussian [14] During the main draw call the values from the blurred SSAO map to the scene lighting. Since the AO is calculated with a quarter of the actual screen size, we sample from the final blurred AO texture by averaging the 4-neighbourhood. For this effect following resources and tutorials were used [8] [9] [10] [15]



SSAO on



SSAO off



Additional to the main effects there are several more we added to our project depending on time and effort.

Echtzeitgrafik

Bettina Schlager 01327281

Johannes Eschner 01633402

- Omnidirectional Shadows for multiple lights [3]
- Water reflection and refraction fresnel effect [9]
 - The vertices on the water surfaces are displaced with sum of sines [11]
 - Fresnel effect is used to adapt the refractive and reflective of the surface
 - Water reflection is correctly implemented for the debug camera only
 - Water Caustics with support of cell noise created in shader [4]
- Physically Based Rendering with normal mapping [2]

ASSETS:

All assets except for the piggy were modeled in Blender and imported using the glTF format. The piggy model is from BlendSwap [12]. The refractive snail model is a 3D scan of a real clay snail.

The music used in the demo is the song Samurai by Bonobo [13]

CONTROLS:

Key	Function
TAB	Switch to debug cam
WASD	Move debug cam
LMB Drag	Rotate debug cam
F1	Show buffer debug
F2	Show frametime and fps
F3	Draw wireframe
F4	Toggle through shadow modes (Soft, Hard, No Shadows)
F5	Toggle screen space ambient occlusion
F11	Reset Animation

Scene settings such as screen resolution and fullscreen mode can be edited in the settings.ini file located in the assets folder. Here it would also be possible to load a different scene as our custom importer can handle all kinds of different gltf files.

TEST ENVIRONMENT:

Our implementation was tested on the lab PC AMAROK with the Nvidia GTX 1060

References:

[1] TinyglTF library <https://github.com/syoyo/tinygltf>

[2] Learnopengl PBR <https://learnopengl.com/PBR/Theory>

Echtzeitgrafik

Bettina Schlager 01327281

Johannes Eschner 01633402

- [3] Learnopengl Point Shadows
<https://learnopengl.com/Advanced-Lighting/Shadows/Point-Shadows>
- [4] Voroni cell noise <https://thebookofshaders.com/12/>
- [5] Sound library for c++ <https://www.ambiera.com/irrklang/>
- [6] Chris Wyman, "Interactive image-space refraction of nearby geometry", *In: Proceedings of the 3rd International Conference on Computer Graphics and Interactive Techniques*, pp. 205-211, 2005, doi: 10.1145/1101389.1101431
- [7] <https://tympanus.net/codrops/2019/10/29/real-time-multiside-refraction-in-three-steps/>
- [8] <https://learnopengl.com/Advanced-Lighting/SSAO>
- [9] <https://mtnphil.wordpress.com/2013/06/26/know-your-ssao-artifacts/>
- [10] <http://john-chapman-graphics.blogspot.com/2013/01/ssao-tutorial.html>
- [11] <https://medium.com/@joshmarinacci/water-ripples-with-vertex-shaders-6a9ecbdf091f>
- [12] Piglet model <https://www.blendswap.com/blend/18163>
- [13] Samurai by Bonobo <https://bonobomusic.com>
- [14] Gaussian blur:
<https://software.intel.com/en-us/blogs/2014/07/15/an-investigation-of-fast-real-time-gpu-based-image-blur-algorithms>
- [15] Dominic Fillion and Rob McNaughton. 2008. Effects & techniques. In *ACM SIGGRAPH 2008 Games (SIGGRAPH '08)*. Association for Computing Machinery, New York, NY, USA, 133–164. DOI:<https://doi.org/10.1145/1404435.1404441>,