Documentation

Group/Game Name: Get Out Of My Swamp!

Brief description of implementation: The game is a round based "action" game, where Greg, a friendly troll, has to defend his home in a swamp from pawns from a neighbouring village. Pawns come in rounds and attack Greg, who defends himself.
For the implementation we tried to start with a really good structure, which helped a lot the more we programmed, however there also were some structurs which we had to redo or edit a lot, when it came to implementing new features. In general we did a lot or Pair Programming which also helped with finding bugs and thinking about solutions. In terms of performance and efficiency, the progam could for sure be improved, but we did our best in the time we had.
As always, the time was not enough, so we had to really focus on the effects and mandatory gameplay. We plan do add a bit more environment objects and clean up the game a bit for the final game event.

Additional libraries: Assimp (Model loading), Physx (Collision Detection), stb_image.h (Image/texture loading), (ExtendedShader.h from https://learnopengl.com/code_viewer_gh.php?code=includes/learnopengl/shader.h to allow for geometry shader, allowed according to TUWEL forum)

Gameplay:

Mandatory:

- 3D Geometry: All objects are loaded using Assimp (trees, map, pawns, ...). Most models are from the internet (free to use with credit, see modelReferences file), some models were made or edited by us (map, Greg was edited, Pawn models were rigged and animated). For model loading we mostly used this resource: https://learnopengl.com/Model-Loading/Assimp
- Playable: The game is playable, One can move with WASD and attack with left mouse button.
- Advanced Gameplay: Currently there are 5 rounds of increasing difficulty. The player has to survive all 5 and then the game is won. It is possible to die, i.e. lose, and to win.
- Min 60 FPS and Framerate Independence: Everything that could be influenced by framerate is "scaled" using delta t which indicates the length of the frame. The game runs constantly on 60FPS on our PCs (GTX970) and minimum FullHD.
- Win/Lose Condition: There is a clear win condition. When the player has survived 5 rounds, the game is won (also displayed).
- Intuitive controls: The movement is similar to other 3rd person games, also requirements with continous input being handled with polling is implemented aswell.
- Intuitive Camera: The camera is implemented as intuitive as possible. It is also possible to look around holding the left mouse button. Once the player moves,

the camera is reset to its initial position, with the exception of the height. So when moving, the player always looks in the direction they are running.

- Illumination model: There are 5 lightsources, a directional light and 4 pointlights. The directional light represents the moon and the pointlights are torches. Normal vectors are also provided by the models.
- Textures: All objects have a texture assigned.
- Moving Objects: There are moving objects, the player character (Greg) itself and additionally the pawns.
- Documentation: Everything we implemented is documented in this file.
- Adjustable Parameters: The required parameters can be configured in the settings.ini file, including brigthness.

Optional:

- Collision Detection (Basic Physics): For this we used physx. Collision detection is provided between pawns, Greg (the player character), the map and trees on the map. Physx is a bit of a struggle, because it is hard to get it do do what we want, in terms of movement and collision detection. We also struggled a little bit to handle the slopes on our map with it, as it is hard to find the right parameters so walking up a slope works, but not gliding when walking down. We definitely plan to improve upon this for the game event.
- Advanced Physics: -
- Scripting Language Integration: -
- View-Frustum Culling: View Frustum Culling is implemented, the result can be seen in the console. Every 30 seconds a "statistic" is printed, about how many objects are currently culled. It can be turned off with the F8 key. Note that we do multiple render passes, and view frustum culling is only applied in some, so that we can e.g. provide a shadow of an object even if not in view. For this we used https://learnopengl.com/Guest-Articles/2021/Scene/Frustum-Culling and https://www.lighthouse3d.com/tutorials/view-frustum-culling/
- Heads-up Display: A Heads up display is also implemented. It shows the healthbar of the player. Additionally, at the end of the game it is displayed how whether the player won or lost. This endscreen is also a non-trivial object as required (using a texture and blending). For rendering this, we mostly used this resource: https://learnopengl.com/In-Practice/2D-Game/Rendering-Sprites. For this, we used a separate shader, which only handles 2D. **Can be turned off/on with F10.**

Effects:

Lighting:

- Lightmap using Separate Textures: -
- Lightmap using In-Game Calculation: -
- Shadow Map with PCF: We implemented this using https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping and https://learnopengl.com/Advanced-Lighting/Shadows/Point-Shadows. We are

using shadowmapping with PCF both with directional lights and point lights (using the appropriate ShadowSamplers). Countering the artifacts was hard and it is always a tradeoff between different things, but we tried to find good parameters. For this we used different shaders, and also used a geometry shader to be more efficient. For this we included an extra Shader.h file (renamed to ExtendedShader.h) from a tutorial website (https://learnopengl.com/code_viewer_gh.php?code=includes/learnopengl/shader.h) but this was explicitly allowed in the Tuwel Forum

- Shadow Volumes: -

Advanced Modelling:

- CPU Particle System: This effect was pretty straight forward. We used http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/particles-instancing/. Initially we wanted to do it with a Compute Shader, but in the end it was not necessary since we only wanted to use the particles as torch smoke, and there not a lot of particles are necessary, so a Compute Shader would be overkill. This effect can be observed as mentioned when looking at the torches. There is smoke rising, and these are particles. We also used a separate pair of shaders for this.
- GPU Particle System using Transform Feedback: -
- GPU Particle System using Compute Shader: -
- Blobby Object Using Marching Cubes: -
- Subdivision Surface: -

Terrain:

- Tessellation from Height Map: -
- Voxel Terrain using an Octree: -

Animation:

- Hierarchical Animation: -
- Vertex Shader Animation: -
- GPU Vertex Skinning: This was probably the most work out of all the effects. We use vertex skinning to provide a walking animation for the pawns and Greg (we want to improve this until the game event, to also include a punching animation, but we had to set other priorities for submission 2). We mostly used the following tutorial https://ogldev.org/www/tutorial38/tutorial38.html. We animated the models ourselves in blender (including rigging, weight painting, ...), so it could definitely be improved. We then export it as a .dae file, which supports animation. Then it was kind of tricky to get everything working with the bones, and weights.

Texturing:

- Procedural Texture: -
- Video Texture: -
- Specular Map: Initially, we wanted to use specular maps on pawns, to make some of their clothes more shiny. However, we could not get the specular map to be exported with a .dae file, so animation was not an option. We then used a specular texture on the map itself, to make it look wet in some areas (as the setting is a swamp). This can be seen by looking around a bit, it is mostly

specular in the lower parts of the map. Since we already handled this when implementing object loading, it was quite easy to then actually use.

- Environment Map: -

Shading:

- Simple Normal Mapping: -
- Cel Shading: We implemented Cel shading, because we thought it would look quite nice with our low poly approach and Contours. However we think it did not turn out to great so we **disabled it by default, it can be enabled with F9.** In terms of implementation it was quite easy, the only new part was the 1D texture. For this we used the provided course material
- Style Transfer: -
- Brush Strokes: -
- Physically Based Shading: -

Advanced Data Structures:

- BSP Tree: -
- kd-Tree: -
- LOD using an octree: -

Post Processing:

- Bloom/Glow: -
- Lens Flares: -
- Contours via Backfaces: We implemented this to increase the low poly aspect of the game, and make it look less real. We think it looks quite nice. The implementation was pretty straight forward. We created a new very simple fragment shader program and then do one render pass with it. It is not done on every object, but can be observed on the pawns for example.
- Contours via Edge Detection: -

Other special features:

- We also added a skybox, which we painted ourselves. We again used a Cube Texture (like in Shadowmapping). This really helped to make the game more immersive. We used this resource: https://learnopengl.com/Advanced-OpenGL/Cubemaps. For this we used a separate shader.

  Walk-through: Once the game is loaded, pawns will start attacking the player. The player can walk around on the map, and attack with the left mouse button. Pawns within some hitbox will be hit. The player is not supposed to be able to "escape" the swamp. After killing all pawns and surviving all rounds, the game is over. If the players health gets to 0, the game is also over. Bear in mind that sometimes (very very rarely) pawns may still get stuck somewhere on the map, so a bit of looking around might be needed.