Documentation Submission 2 M.A.F.I.A

Implementation Details (Requirements)

Some of our code (Camera, Model loading, Light) is inspired by tutorials from LearnOpenGL[1] and ThinMatrix[2]

Gameplay

We are now moving in a 3D World generated through a Height Map. It is possible to jump in the different levels of the map. The Ingredients are spawning on 14 different spawn points all over the map. They are moving randomly through the map. If they hit a wall they try to jump on the next level. If they fail they turn around and go another way. Enemies are moving in the general direction of the main character. If they hit a wall they have the same behavior as the ingredients, but after a while they return to follow you.

Complex Objects

Aside from the main characters / enemy's body, the skybox which are self-created and the terrain which is generated, all objects are complex. We used Assimp to load the object files in our program.

Animated Objects

If the main character gets hit by an enemy the main characters hat gets animated. We use a sinus function for the animation. If the main character dies or an ingredient/enemy gets hit a "dying" animation is triggered. Here we just use a linear downscale.

Frustum Culling

For Frustum Culling we used `gl_ClipDistance` in the vertex shaders. We give the clipping planes as input to the shader and openGL culls every vertex outside of the planes. Because of that it's not possible to give a count of drawn triangles or objects, because we did not find a way to retrieve this information from the shader. If you press "X" a mini map is shown where all objects get drawn from above that are also actually drawn in the main window. Here you can see the effects of the culling.

Experimenting with OpenGL

FBOs

We use FBOs for many different effects (SSAO, shadows, …). They are all found in the ShaderLogic.cpp file.

Blending

We use blending for the text overlay.

Mip Mapping & Texture Sampling Quality

Our standard configuration is:

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

By pressing F4 (Texture Quality) or F5 (Mip Mapping) you can change the settings.

[1] https://learnopengl.com/

[2] https://www.youtube.com/watch?v=d-kuzyCkjoQ, https://www.youtube.com/watch?v=PoxDDZmctnU

Wireframe Mode
By pressing F2 you can change into wireframe mode.

Features
By pressing F9 we can switch to god mode where the camera is unlinked from the main character and is completely free moveable.
We implemented a music player using irrKlang. By pressing "M" you can switch on the music. A special feature is the party mode (press "P") where a different music is played, all Ingredients jump on the spot and the left and of the main character is waving.
We also use different sounds e.g. when the character gets hit or hits something, or when you receive an award for hitting the correct ingredients.

Texture Mapping
Every Game Object in our game has a texture mapped to it. We read textures from the files using the DevIL library.

Simple Lighting and Materials
Every object has its own material. We can change the properties for each object (see Ground and MainCharacter). If we don't set material details we defined a standard material for the object.
We defined the moon as our global lighting. It is a point light with infinity range.

Additional Libraries
GLM
We use GLM for all kinds of mathematical operations like the generation of transformation matrices.[3]

DevIL
We use DevIL to load textures from files into our program (we only use IL and some minor ILU feature, NO ILUT).[4]

Assimp
Assimp is used to load 3D models in our program. [5]

OpenGL
We also use GLEW[6] and GLFW[7] for OpenGl support.

irrKlang
irrKlang is used for the sound handling and playing.[8]

freetype
freetype is used for the text overlay of our game.[9]

---
[3] http://glm.g-truc.net/0.9.8/index.html
[4] http://openil.sourceforge.net/
[5] http://assimp.sourceforge.net/
[6] http://glew.sourceforge.net/
[7] http://www.glfw.org/
[8] http://www.ambiera.com/irrklang/
[9] https://www.freetype.org/

Effects

We implemented SSAO, shadow maps (with PCF) and cell shading. All the effects are visible well and can be turned on and off (see Contorls.pdf).

Implementation

SSAO

For SSAO we basically used the code from learnOpenGLs SSAO tutorial.[10] We had some problems to actually apply the result of the SSAO texture generation to the ambient part of our color output because we do not use deferred shading. In the end we rendered all of our normal output into an FBO and then combined the texture we got from this process with the occlusion value from the SSAO texture.

Shadow Maps

In the scene there is only one light source, which is the moon. The moon is a directional light and illuminates the whole Scene[11]. We don't cast shadows for the whole world, because that would be to much unneeded information so we implemented, that the shadows will only be generated around the main character. It's not the best approach but it improved the shadow quality and framerate a lot. We tried to use the Hardware for PCF calculations but we got an Error when using Nvidia graphic cards, that we couldn't fix, so we implemented the PCF in software.

Cel Shading

For the Moving Objects in the Scene we also implemented Cel Shading. This is a very simple approach and was done in a few lines of code in the default fragment shader. For the terrain the applying of the Cel Shading effect was not possible because we don't store the correct normals of the generated mashes. The effect locks really good on the tomato-object.

Tools

We used Blender for all the 3D modelling we did ourselves and for correcting textures from online models.

[10] https://learnopengl.com/#!Advanced-Lighting/SSAO
[11] https://learnopengl.com/#!Advanced-Lighting/Shadows/Shadow-Mapping