

# 193.180 VU Computer Graphics

## Submission 3: Final Demo

Melanie Schiebel 12023986  
Quynh Nhu Jennifer Tran 119090573

## 1 Description of Implementation

The demo was implemented in C++ and GLSL. We used OpenGL 4.6 Core profile as our graphics API. The demo was tested on a NVIDIA GeForce RTX 4070 Ti SUPER and Intel.

We implemented the following effects and features:

- Model loading using Assimp
- Controllable and automatic camera
- Blinn-Phong illumination for Directional Light and Point Light
- Shadow Map with manual PCF for Directional Light (complex effect)
- Omnidirectional Shadow Map for Point Light (complex effect)
- Skybox (Cubemap)
- Model loading
- GPU Vertex Skinning for Model Animation (complex effect)
- Adaptive Tessellation (complex effect)
- (Normal Mapping with manual tangent space computation (complex effect))

In the scene, models are rendered with textures and illuminated using Blinn-Phong lighting, combining a warm directional sunset light and a single point light. The source of the point light is a street light and can be toggled on and off. For the directional light, Shadow Mapping with manual PCF was implemented. These shadows can, for example, be seen on the ground or house wall with the backpack. Additionally, Omnidirectional Shadow Mapping using a depth cubemap with PCF was done for the point light. To reduce artifacts and improve the shadow maps (such as shadow acne), a bias was applied. The shadows can be seen on the house walls from the hanging sign or the animated girl. The scene also includes a Skybox implemented using a cubemap texture showing a cloudy sky background. In the scene we can observe animated characters, which were rendered using skeletal animation. In addition a tessellation terrain is rendered based on a heightmap. Moving the camera forward and backward, we can see that the tessellation level gets adapted accordingly.

We planned to omit Normal Mapping, which we included in the proposal, and instead implement Omnidirectional Shadow Mapping. At the end we also tried implementing Normal Mapping with manually computing the tangent and bitangent vectors, which can also be toggled on and off. This is good visible for example on the pavement and the stone wall of the well.

## 2 Additional Libraries

- GLFW : window management (<https://github.com/glfw/glfw>)
- GLM : math library (<https://github.com/g-truc/glm>)
- GLEW : OpenGL function loader (<https://github.com/nigels-com/glew>)
- Assimp : 2D model loader (<https://github.com/assimp/assimp>)

## 3 Controls

The camera can be controlled with WASD and the mouse:

- W: move forward
- A: move to the left
- S: move backward
- D: move to the right
- Mouse Movement: rotate
- Mouse Scroll: zoom
- F1: toggle automatic camera
- F11: full screen (switching only possible if not in automatic camera mode)
- Space Bar: toggle wireframe
- L: toggle point light
- N: toggle normal mapping
- R: start/stop recording camera path

Moving models (nearest model) (mostly for us to get the position):

- Tab: switch between translation and rotation
- Right Arrow: translate x axis
- Left Arrow: translate -x axis
- Up Arrow: translate z axis
- Down Arrow: translate -z axis
- K: translate y axis
- M: translate -y axis
- P: print position and rotation