RaCity

Submission 2 - Documentation

Important notice:

We used our implementation from the CGUE-2018S course as a starting point. Therefore, most of the basic features (OpenGL 4.x Core Profile rendering, simple illumination model, Textures, Physics via Nvidia PhysX, Debug camera) have already been implemented before we started to work on the demo.
For this submission, we adapted our existing solution for the requirements of the Real-time rendering course, by adding an appropriate scene and automatic camera movement.

Story:

Our demo starts in a garage in a small city. Two cars are parked inside the garage. Multiple moving camera shots reveal the vehicles and various parts of the garage to the viewer. Multiple spotlight sources are illuminating the garage and the cars. After some time one of the cars starts moving outside of the garage into the city. The camera follows the car as it leaves the garage, drives through a spotlight (street lamps) illuminated street, shows some of the city and then pans to the side as the demo ends.

Features:

Texture mapping

Every object in the game has UV-coordinates, is therefore textured and is loaded with assimp from a resource file (.obj, .dae, etc). If a texture is missing (or not set for artistic reasons), a default white 1x1 pixel texture is used, so the color of the object is defined only by the material coefficients (ambient, diffuse, specular and emissive). The skybox is a cubemap wrapped on a simple cube, rendered as last object of the scene and with it's depth value set to max so it always fails the depth test if there are elements in front. The skybox texture was taken from "https://93i.de/p/free-skybox-texture-set"

Lighting

The object loader reads all the light and material data from the resource file of the map (map needs to be saved as Collada-file therefore) and supports directional, point and spotlights. There is one directional light (moonlight), a few spotlights in the garage to illuminate the cars and posters and a lot of spotlights on both street sides to illuminate the main street in the scene. All objects use the phong illumination model.
.

Automatic camera movement:

We create our scene in Blender, export it as Collada (.dae) file and import it in our project using the assimp library. We animate our camera motion in Blender and import it into our project. To create an automatic camera movement we created a class for animated properties that stores a keyframe list. To get the current value of an animated property, a timestamp must be provided. The value of the animated property is obtained via linear interpolation of the two closest keyframes in the keyframe list and using the provided timestamp. We added animated properties for the position, rotation and scale of the camera. By retrieving all three values of these animated properties at a given time the MV-matrix of the camera can be generated.

The demo was tested on NVIDIA graphics cards!

Models:

The models that can be seen in our demo were taken from the following external sources and further edited in Blender.

- https://rigmodels.com/model.php?view=Garage-3d-model__4XEVGVA1GGXMSO1O2N90MVXNG&searchkeyword=garage
- https://www.turbosquid.com/3d-models/3d-car-bugatti-veyron-1161465
- https://rigmodels.com/model.php?view=Skyscraper-3d-model__3QQSRTIVT5E7UFBUSCDI2XYL4&searchkeyword=skyscraper
- https://rigmodels.com/model.php?view=Skyscraper-3d-model__7C8014G7Z0O530ZN5NQG236PX&searchkeyword=skyscraper
- https://rigmodels.com/model.php?view=Skyscraper-3d-model__3S3KLK80T3SYFVSF99WB0MI9R&searchkeyword=skyscraper
- https://rigmodels.com/model.php?view=Elevator-3d-model__SG6ERUHNZB6AF8EP519PMYXUO&searchkeyword=skyscraper
- https://rigmodels.com/model.php?view=Street_Front-3d-model__TB3PD20FR9BTIIFH5TL6LUB99&searchkeyword=street
- https://rigmodels.com/model.php?view=Street_Front-3d-model__74DCD00EGAYWWE14QV69WOTZ0&searchkeyword=street
- https://rigmodels.com/model.php?view=Street_Front-3d-model__D8L1JNI1REZFM9FVZW40RQG93&searchkeyword=street
- https://rigmodels.com/model.php?view=Building_Front-3d-model__3KMR3VGZOESIU6PZ8GXXYGT3F&searchkeyword=street
- https://rigmodels.com/model.php?view=Scene-3d-model__ETFCFLAZQZMXBFYBPSIQ3K6TW&searchkeyword=street
- https://rigmodels.com/model.php?view=Street_Lamp-3d-model__HOKGIVE33UW3O1ABN7TVHD6R2&searchkeyword=street%20lamp
- https://rigmodels.com/model.php?view=Mail_Snorkel_Box-3d-model__F8EZ71ZXBPSZ74JMDKE2STL87&searchkeyword=street

- https://rigmodels.com/model.php?view=Building-3d-model__W3X2VMUKR89GNULCKZH6HJMMD&searchkeyword=city

Effects:

The following effects were implemented:

- **Shadow maps (with percentage closer filtering)**:
  Our demo implements shadow mapping for both directional and spotlights. Since our scene contains only one directional light (the moon light), we do not deal with multiple shadow maps for directional lights. However, our scene contains multiple spotlights. Our first approach was to simply render the shadow map for the closest spotlight to the camera. The problem with this is that the shadow suddenly appeared and disappeared, when the camera was moving. We mitigate this problem by always rendering the twelve spotlights that were close to the camera. Therefore, we always render 13 shadow maps for different lights each frame. This has a noticeable impact on the performance of our demo. However, since we are still able to maintain a framerate of >= 60 fps, we did not further optimize the shadow mapping.
  The shadow mapping is integrated into the deferred shading effect. At first we render all 13 shadow maps, then we supply them to the geometry stage of the deferred rendering effect and render the shadow factors into two additional framebuffer color attachments (one for the directional light and one for all spotlights).
  The shadow maps for the spotlights are rendered using a perspective camera at the spotlight's locations looking into the spotlight's direction. The shadow map for the directional light is rendered using an orthographic camera. We adapt the size of the size of the shadow map (zooming in and out), depending on the camera's location. The closer the camera is to the ground, the more zoomed in the orthographic camera is. This improves the visual appearance of the shadows when the camera is close to it. However, it also generates artifacts due to the limited field of view of the rendered shadow map. These artifacts are noticeable, when the camera is close to the ground and looking into the distance, where suddenly no shadows are rendered anymore. We decided that the benefits of an adaptive orthographic camera for rendering the shadow map of the directional light outweigh the drawbacks.

  Literature:
  - Williams, Lance. "Casting Shadows on Curved Surfaces." (1978).
  - Wimmer, Michael et al. "Light Space Perspective Shadow Maps." (2004).
  - Scherzer, Daniel. "Robust Shadow Maps for Large Environments." (2005).
  - Zhang, Fan et al. "Parallel-Split Shadow Maps on Programmable GPUs" (2007). (url: https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch10.html)

- **Deferred shading**:
  The biggest advantage of deferred shading is that lighting can be calculated for a large number of different dynamic lights regardless of the scene complexity. This is achieved by separating the processing of the scene geometry from the actual lighting calculation. In our demo we created two different rendering passes: a geometry pass and a lighting pass. The geometry pass processes the scene geometry and renders all the information that is necessary for the lighting calculation of a pixel into a separate framebuffer color attachment. We rendered the following pixel information in separate color attachments in the geometry pass: Position, Normal, Albedo color, Shininess factor, Ambient factor, Diffuse factor, Specular factor, Specular color, Shadow factor for directional lights and Shadow factor for spotlights.
  All these separate framebuffer color attachments are supplied to the lighting pass as textures, where the lighting calculation if performed for each pixel. Since the textures already contain all the information of the scene that shall be displayed in screen space coordinates, the lighting pass simply projects the output of the geometry pass onto a plane that has the size of the screen (after performing the lighting calculation for each pixel).

  Literature:
  - Policarpo, Fabio et al. “Deferred shading tutorial” (2009).
  - Kumar, Samir “Optimization to Deferred Shading Pipeline” (2017).
  - Valient, Michal “The Rendering Technology of Killzone 2” (2009). (url: https://www.guerrilla-games.com/read/the-rendering-technology-of-killzone-2)
  - Shishlovtsov, Oles “Chapter 9. Deferred Shading in S.T.A.L.K.E.R” (2005). (url: https://developer.nvidia.com/gpugems/GPUGems2/gpugems2_chapter09.html)

The following additional source were used to implement the effects:

- http://ogldev.atspace.co.uk

Controls:

Key	Effect
ESC	Quit demo
F2	Frametime on/off
F3	Wireframe on/off
F4	Switch screen buffer to display (Position, Normal, etc.)
F8	Frustum culling on/off
+/~	Toggle fullscreen/windowed
C	Toggle camera freeview/automatic

Used libraries:

- GLM for math
  https://glm.g-truc.net
- ASSIMP for model loading
  http://www.assimp.org/
- STB Image for image loading
  https://github.com/nothings/stb
- Inih for settings file parsing
  https://github.com/benhoyt/inih
- Freetype for font loading
  https://www.freetype.org/
- GLFW for window handling
  http://www.glfw.org/
- GLEW for opengl extension loading
  http://glew.sourceforge.net/