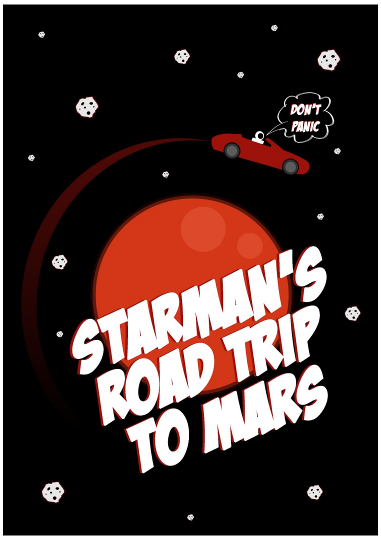
Starman’s road trip to Mars

Gerhard Andreas Stog (00954038)
Julia Eder (01229278)



Brief description of the implementation

Intuitive camera

We have implemented a third person camera, which follows the car. The camera moves forward automatically. It is possible to move around with keys in the 3D space.

Moving objects

The asteroids can rotate around themselves, and other asteroids can rotate around a parent asteroid.

Win/Loose Condition

The player wins the game, if 5 battery-boxes were collected. The player loses the game, if the health points were below 0.

3D-Geometry

The car model is loaded with Assimp. The model is loaded from the internet.

Texture Mapping / Simple lighting and materials

All objects are illuminated with the Blinn Phong Illumination model and textured. Directional Light is used. The skybox doesn't have lightning.

Controls / Adjustable Parameters

WASD	Move Player
ESC	Close Game
F3	Fullscreen On/Off
F4	Cel Shading On/Off
F5	Contours On/Off
F6	HUD On/Off
F8	View Frustum Culling On/Off

Brightness

The brightness can be changed in the settings.ini file.

Gameplay

The camera moves forward and you have to avoid crashing into asteroids. In the beginning the player has a health of 100. If the car crashes into an asteroid the player loses 20 points of his game life. The game is over if no life points are left. The mission of the player is to collect 5 battery-boxes. If the player collects enough battery-boxes and has enough health points, the player wins the game.

Feature

A skybox is implemented to give a better view of the movement of the camera. The skybox is textured with DDS faces.

Collision detection

The collision detection is done with bounding spheres. It is not possible to crash into asteroids. When you crash into asteroids, you get pushed back.

Optional Gameplay

Heads-Up Display → **Stog**	The HUD is implemented with the library freetype. In the game the fonts “Obelix Pro” and “Arial” were used.
View-Frustum Culling → **Eder**	The frustum is calculated with die view and projection matrix of the camera. Every geometry object is approximated with a sphere to cull.

Effects

Cel Shading → **Eder**	For this effect, we used the texture fragment shader provided from the template and expanded the phong shading model. In the first the light intensity of ambient, diffuse and specular is calculated. Then the intensity is multiplied with the shading level, in our case 5.
Contours with Edge Detection → **Eder**	The contours were added in post processing. For edge detection the sobel operator is used. The shader not only returns the color but also an image of the normals.
GPU Particle System with Compute Shader → **Eder**	For the particle system, we followed the instructions of the slides in Tuwel. In the Particles class some initial particle data is created and copied to the SSBOs. The compute shader is updating this particles and returns the values to the ping-pong SSBOs. The geometry shader transforms the particles into a quad that always faces the camera´s viewing direction. In the fragment shader the color for each particle is set.

Hierarchical animation → **Stog**	For the implementation of the animation, we have a scene object class. The asteroid and the battery class have an inheritance on the scene object class. With the method addChild any number of children objects can be added. The children objects do not need to be updated and drawn.
Environment Mapping → **Eder**	For this effect, a skybox is necessary which shows the background from the scene. The six faces were loaded in the Material class. In the texture shader the ray of reflection is saved and the the right position in the texture is found there. In the end with the method mix() the reflection part from the texture is shown. The coefficient in mix() determines the degree of the reflection.
Shadow Mapping with PCF → **Stog/Eder**	The scene is rendered two times. In the first pass the depth map is rendered and in the second pass the scene is again rendered with the generated depth map. To achieve this procedure we have to shaders. One is responsible for the creation of the depth map and the other is a extension of the texture shader, which was provided from the framework. In the texture shader the calculation of the shadow and the correction of the artefacts is done.

Additional libraries and Links

Skybox	➔ https://spacedock.info/mod/924/Pood%27s%20Calm%20Nebula%20Skybox
Cel Shading	➔ http://www.sunandblackcat.com/tipFullView.php?l=eng&topicid=15&topic=Cel-Shading
Contours with Edge Detection	➔ http://adrien.io/opengl-course/post-processing/ ➔ https://computergraphics.stackexchange.com/questions/3646/opengl-glsl-sobel-edge-detection-filter
GPU Particle System with Compute Shader	➔ CG Repetitorium Slides
View-Frustum Culling	➔ http://www.lighthouse3d.com/tutorials/view-frustum-culling/ ➔ http://www.rastertek.com/dx10tut16.html
Battery Texture	➔ https://i.imgur.com/Hevv8Gj.png ➔ https://ya-webdesign.com/explore/logos-transparent-flash/
Environment Mapping	➔ https://learnopengl.com/Advanced-OpenGL/Cubemaps
Shadow Mapping with PCF	➔ https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping ➔ http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/

Model Loading **(Assimp, SOIL)**	➔ http://www.assimp.org ➔ https://www.lonesock.net/soil.html ➔ https://learnopengl.com/Model-Loading/Assimp
HUD **(Freetype)**	➔ https://www.freetype.org ➔ https://learnopengl.com/In-Practice/Text-Rendering ➔ https://www.dafont.com/obelixpro.font ➔ https://docs.microsoft.com/en-us/typography/font-list/arial