Planet Evasion

Lukas Mayr (1225528), Andreas Roschal (1225600)

Implementation status – 2nd Submission

Details of the Implementation (partly already covered in the documentation of the 1st Submission):

Game Loop and Logicals:

The main loop of our game is located in „main.cpp“. In this file we also create and store our logicals („Planet.cpp“, „Player.cpp“, „AsteroidSpawner.cpp“, etc.). These objects are holding data like the scene-object-models, shaders, textures and materials, but also specific information like the player's lives or the spawn-time for asteroids. For loading model-data from .obj-files (created in Blender) we use functions provided by the library „Assimp“.

Camera and Controls:

Camera:

All information and functions to manipulate the camera and it's position are stored in the file „SimpleCamera.cpp“. The object for the camera is created in „main.cpp“.

Controls:

The player-model can be moved around the planet by using the WASD-Keys.

With horizontal mouse-movements, the camera can be rotated around the player-model.

The player-model „follows“ the camera. For example when „W“ is pressed, the player-model will move forward (in the direction the camera is pointing) and rotate so that he will be facing in this direction (analog for ASD-Keys).

When the middle mouse-button is held down, the vertical view-angle of the camera can be adjusted by moving the mouse vertically.

By turning the mouse-wheel the player can zoom in and out.

All button-events are detected in „main.cpp“.

Moving Objects:

Moving objects in our game are the player-model, the asteroids and the power-ups. The movement of the player-model is described above. When the game is started (by pressing Enter) asteroids are randomly spawned and move towards the planet‘s surface with constant velocity. Power-ups are just moving slowly up and down.

Static Objects:

The planet and different decoration-items (houses and lanterns) on the planet’s surface are static and are not moving.

Transparent Objects:

We used transparency for the power-ups. They are half transparent.

Animated Objects:

The only animated object in our game ist he player-model. It has an hierarchical animation. When the player is running around, the legs and arms are moving.

Texture Mapping:

The loading of images, which are used as textures, is done in the file „TextureImporter.cpp“. The uv-coordinates themselves are loaded with „Assimp“ along with all other model-data. All used textures are created by ourselfs (except the night-sky and the clouds of the day-sky).

Shaders:

We implemented cel-shading for most of the scene-objects in our game (more information about cel-shading in the „Effects“-section).

For our skycube we wrote a shader in which the player’s position is compared with the main light-direction. If the player is on the dark side of the planet, the sky will turn into a night-sky. When he’s on the bright side, a day-sky is applied as the texture for the cube. The transition between night- and day-sky is made smoothly by linear interpolation between these two textures.

Lighting and Materials:

Lighting:

For the main lighting we use a directional light source. This light-source is represented by the file „DirectionalLight.cpp“. The light-source itself rotates with constant angular speed around the planet to simulate very short day-night circles and therefore make the scene feel more dynamic and alive.

Besides the main light source there are also lanterns in our game, which work as point-lights with a limited range (to create the effect of a smooth ligthing transition). The point lights are implemented in the file „PointLight.cpp“.

Material:

The material-data of a logical is stored in „Material.cpp“. The material itself is NOT extracted from the corresponding .obj-file (created in Blender), but defined in the code manually. The material contains information for the ambient-, diffuse- and specular-factor and the shininess of highlights.

Basic Gameplay:

When the game is started, asteroids are moving towards the planet's surface. The player's goal is to avoid the asteroids and stay alive until the astroid-incoming ends (parameters can be changed by the user → see „Config Files"-section). When the player is hit 3 times by an asteroid, he loses. The player's lives are displayed in the left upper corner of the window.

There are also power-ups randomly spawning on the planet's surface which give the player benefits.

Power-Ups:

- red power-up: restores one live of the player
- blue power-up: increases the player's speed for 5 seconds
- yellow power-up: makes the player `invulnerable` for 5 seconds

The houses and lanterns on the planet are just for decoration and prevent the player from moving around freely as there is a collition detection.

Collition detection:

The collition detection for our game is written by ourselfs and therefore very simple and a little bit buggy (In our collition detection we expect that every scene-object is a sphere because of simplicity.).

Other techniques:

View frustum culling:

For view frustum culling we inculded the files described in this tutorial: http://zach.in.tu-clausthal.de/teaching/cg1_0607/literatur/lighthouse3d_view_frustum_culling/index.html

These files are located in the filter „ExternFiles" in the project view.

Effects:

Overview of our effects:

Effect	Implementation ingame	Points
Cel-shading + Outlines with edge detection	For our main shading-model we chose cel-shading because we like the style of it. For the black edges we wrote an edge-detection shader (we used this implemenation as a reference: http://coding-experiments.blogspot.co.at/2010/06/edge-detection.html).	0.5 + 1.0 Points
Variance Shadow-mapping	We implemented variance shadow-mapping for our main light source (sun). Due to performance we decided to not implement a shadow-mapping for the lanterns' light sources.	1.5 + 0.5 Points
Bloom	We implemented a bloom effect, because we thought it would look pretty nice on our bright day-sky.	1.0 Points

Planned but not implemented effects:

In our project assignment, we also planned to implement „projected textures", „particle systems" and „lens flares".

We didn't implement „projected textures", because we wanted to use this technique for the shadows of the asteroids, so the player gets a better feeling how far they are away from the planet's surface. Since this would have required a transformation and texture look-up for every single asteroid that is currently moving towards the planet, the technique would be really performance stressful in our case.

We didn't implement „particle systems" since generating a fire-effect or smoke-effect for every single asteroid would also be very bad for the performance.

Instead of „lens flares" we implemented „bloom".

Features of the game:

- The positions of the decoration-items and the asteroid incomings are generated completely randomly (in the corresponding spawner-files). Because of the use of polar coordinates, the decoration-items and asteroids are uniformly distributed on the whole surface and not gathered around the planets poles (Which would be the case, if two random angles would be used to specify a position on the planet's surface.).
- The sky-cube which wraps the scene creates the effect of (more or less) realistic day-night circles. This is achieved with linear interpolation between two textures.
- All important parameters for the asteroid incoming can be set in config files to change the difficulty of the game.
- The difficulty of the game can be freely changed by the user by setting parameters in the config-files.

Controls (summarized):

Input	Reaction
WASD	moves the player-model and the camera around the planet
horizontal mouse-movement	rotates the player-model and camera horizontal around the player-model
vertical mouse-movement (while holding down the middle mouse-button)	rotates the camera vertical around the player-model
mouse-wheel	zooms in and out

Help controls:

All feedback-messages are written in the console.

Input	Reaction
SPACE	Pause/Continue Game

ESC	Close Game
ENTER	Start Game
F2	Framerate on/off
F3	Wire Frame on/off
F4	Textur-Sampling-Quality: Nearest Neighbor/Linear
F5	Mip Mapping-Quality: Off/Nearest Neighbor/Linear
F6	Draw only edges (from the cel-shading) - on/off
F7	Write number of render objects - on/off
F8	Viewfrustum-Culling on/off
F9	Transparency on/off
F10	Sun Movement on/off
F11	Backface Culling on/off

Config Files:

We have 2 config-files. „fullscreenConfig.txt“ and „asteroidSpawnerConfig.txt“.

In the fullscreen config-file there are 3 parameters. The first one is true or false, depending if fullscreen should be used or not. The two other parameters are the window's width and height.

In the asteroid-spawner config-file there are 2 parameters. The first one defines the mean number of asteroids that are spawned per second. The scond one defines the time period for the asteroid incoming. For the first parameter a number over 1000 is not recommended, since this will cause a major headache for our virtual character. :) … and there might be performance issues as well.

Used Libraries:

Glfw (newest version): http://www.glfw.org/

Glew (newest version): http://glew.sourceforge.net/

Assimp (newest version): http://assimp.sourceforge.net/

Devil 1.7.8: http://openil.sourceforge.net/