Developing a Jump'n'Fly Game

Results of a practical course at the Chair for Computer Graphics and Multimedia
(RWTH Aachen University, Germany)

Karsten Ansteeg* Christian Mattes† Florian Wehling‡ Moritz Werthebach§

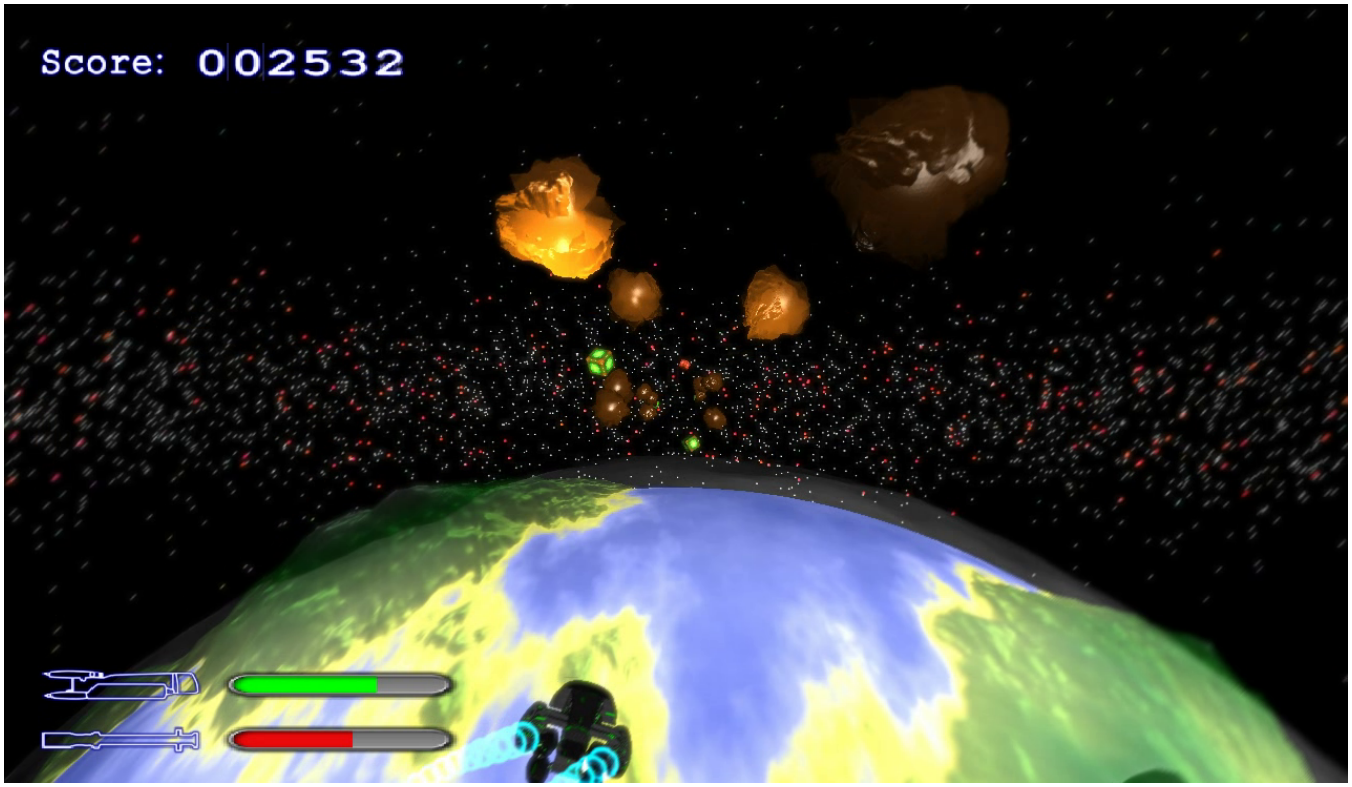

Figure 1: *The final game "Kepler NXT 13" by Karsten Ansteeg, Christian Mattes, Florian Wehling and Moritz Werthebach.*

Abstract

In this year's practical course our task was to develop a jump and fly game. The course was hosted by the chair of computer graphics (i8) and kindly assisted by Ming Li, Dominik Sibbing and Jan Robert Menzel.

Our team, referred to as "Group F", consisted of four students: Karsten Ansteeg, Christian Mattes, Florian Wehling and Moritz Wertebach. This paper is intended to review and recapitulate the stages of this practical course.

1 Game Concept

First, we had to come up with a concept of the overall game mechanics and settings. In your game, called "Kepler NXT 13", an exploration space ship is on a mission to explore new habitable planets.

The ship is orbiting an earth-like planet while the objective of the game is to navigate it safely through an asteroid belt. Using collectable power ups and the on-board laser cannon, the payer has to make it as far as possible through the asteroids in order to achieve a new highscore.

*karsten.ansteeg@rwth-aachen.de
†christian.mattes@rwth-aachen.de
‡florian.wehling@rwth-aachen.de
§moritz.werthebach@rwth-aachen.de

2 Milestone I

The development of the game was divided into three milestones. During the first milestone, we implemented a basic game loop, distributed the various tasks and got used to the provided tools and frameworks.

It took some time to get the git repository working on every machine and since we were more or less unexperienced with techniques in computer graphics, we had a steep learning curve in the initial stage. As a result, the main focus was on understanding the graphics libraries (ACGL, GLFW). Nonetheless, we already managed to create a basic runnable and playable game.

Christian contributed to the project with a static planet that provided a day/night cycle. He also implemented several shaders including a basic lighting model based on a moving sun, as well as specular mapping and city lights at night for the planet.

User input via GLFW was included to navigate the ship. This task was adopted by Moritz. To allow for a more realistic steering experience, he added an accelerated movement to the controls. Therefore, the GLFW_GET_KEY functions were used to have complete ship controls with the arrow keys. The angle acceleration rises with holding an arrow key to a set maximum and after releasing the

key it swings out.

Karsten was responsible for the object management. This included the creation of asteroids, the movement of objects in 3D space and their deletion. It also required a collision detection of the ship with obstacles of the scene, which was also implemented in the first milestone. The ship as well as the obstacles are fixed to a circular path. Thus, the position of each object, called an entity, can easily be defined by variables for distance and rotation (or angle). This limited movement also simplified the complexity of the self-made physics simulation like the collision detection, for example.

The high-res model of the space ship was created by Florian. Since he is most experienced in computer graphics, he also set to work on a multipass renderer, which was already available at the end of the first milestone. The multipass renderer was the basis for several effects later on.

3 Milestone II

A goal for the second milestone was a motion blur effect. For that purpose, Florian refined the multipass renderer and implemented a motion blur shader. He also assisted with the interface shader that was written by Karsten.

In this milestone, we gave up on the static planet and moved on to a planet that is procedurally generated for every start of the game. To achieve this, Christian implemented a perlin noise generator, that was then used to generate a random planet, as well as unique asteroids. As a result, almost everything in the game is created procedurally, so every new playthrough is unique. The planet was detailed with different terrains, snow lands, seas and a randomly changing cloud layer. Furthermore, he designed a basic class for in-game 3D sounds, which makes use of the OpenAL system.

To improve playability during night (in-game time), Moritz added a cone light for the space ship; therefore, he adapted several shaders of the game. The light works only for those asteroids which are in a given angle to the space shuttle. Also, first steps for a game menu were implemented.

In the second milestone, Karsten implemented power ups for laser and life energy. This also affected the collision detection; therefore, there were some modifications needed to handle different types of objects. In combination with a shooting mechanism, he also wrote a shader for the interface, which shows the status of the ship and the laser cannon.

Figure 2: *Milestone II: Planet, lasers, power ups and interface.*

At this stage, the shooting functionality was added for which Karsten also created the new required textures, geometry and shaders. The interface was implemented as a separate render pass that was attached to Florian's multipass renderer. It shows a green bar indicating the remaining life of the ship as well as a red bar for the remaining laser energy.

4 Milestone III

At the end of the last milestone, the game had to be properly runnable. Therefore, we did some polishing and bug fixing.

Furthermore, Moritz implemented the final game menu so he had to do some changes "under the hood". This includes some modifications to the input handling and additional controls for the menu. He also expanded it with a highscore list. The menu controls were realized with the arrow keys. With the beginning of the game, the "Up" and "Down" keys were used to switch between three menu items. To select the desired option, the enter key can be used. If "new game" is selected, the game state will be reset. It is also possible to pause a running game with the "ESC" button; the game continues when pressing "ESC" again. In association with that, a "game-over" screen was implemented. Karsten added a (real-time) score label to the on-screen interface as well as an updated spawn function that provides a consistent increase in difficulty for the player.

Florian tweaked the motion blur and as it doesn't look like we wanted, he replaced it with a collision feedback that is visualized by a blur.

Figure 3: *Milestone III: Collision with destructible asteroids and visual feedback with motion blur.*

At this stage, the game also supports several game pads which was implemented by Christian. Christian also created destructible asteroids so now asteroids break more realistically into fragments when they are hit by a laser or by the space ship. Just like the planet and the asteroids, the skybox is now dynamically generated with every start of the game. The skybox-generator is based on rulesets for groups of stars, including the number of stars in the group, their color and their distribution in respect of the galactic plane, allowing for a milky-way effect. The collisions were also accompanied with appropriate sound effects for the laser and explosions. Finally, a glow effect was implemented into the game.

5 Final Product

The final result is a jump and fly game with various visual effects such as motion blur, lens flare effects and dynamic lighting. The game is runnable on Linux and Windows PCs as well as on Mac OS.

A special feature of the game is the procedural generation of geometry and textures and the fact that we passed on almost any external resources. Thus, every member of the team was involved in the entire project and gained insight into the development of a standalone video game. This included programming, of course, but also techniques in software engineering (drafts and concepts) and content creation as well as learning to work in a team. At the end, we delivered a nice looking and properly running jump and fly game.