InitialT Submission 2

Christian Stippel (11778254)
Martin Rupp (11709466)

May 2019

1 Introduction

This document is meant to display the contents of our project, "Initial T". It contains information about the Gameplay, how to play the game and details of our Implementation.

2 Gameplay

In the game you are able to drive around in a tank on a map. You can interact with other game objects by shooting them, or touching them. You can get more ammunition by drifting. Every level can contain targets you have to shoot and flags you have to collect. Once you collected all flags and shoot all targets you can go trough the goal to jump to the next level. Your progress and the current tasks are displayed on the top right of your screen.
The goal is to complete all levels.

3 Controls

Key	**Effect**
W,A,S,D	Controls the tank movement
W,S	Rotate in air forwards and backwards
A,D	Rotate in air Around Z-axis
E,Q	Rotate in air Left and Right
Space	Jump (Usefull in combination with WASD when stuck)
Hold Right	Look around (Arcball camera centered above the tank)
Click Left	Shoot if you have ammunition
Hold LShift	Hand brake (use it to start drifting)
Mouse-Wheel	Zooming in and out of the scene
F-1	Toggles Wireframe-Mode
F-2	Toggles Backface-Culling
F-3	Toggles Vsync
F-4	Toggles Mouse-locking to window
F-5	Show debug information about fps etc.
F-6	Toggle HUD
F-11	Toggles Fullscreen
Key-Up	Higher the brightness
Key-Down	Lower the brightness
Esc	Exits the game

Additionally the initial width and the height can be passed as command line arguments. However the game can also change it's Viewport dynamically if you rescale the window or toggle the fullscreen.

4 Features

- Playable
  The program hopefully runs on the Lab-PCs without crashing.
- 3D Geometry
  The tank, the shots, the goal, the boxes and the streets model are loaded from model files.
- Win/Loose Condition
  You win the game by finishing all levels
- Intuitive Controls
  WASD-Controls like in almost all racing-games and WASDQE for rotating in the air.
- Intuitive Camera
  A arcball camera was implemented that follows the tank. The camera manages itself but if you hold down the right mouse button. Than you can adjust the target pipe of the tank.
- Textures
  Our tank, the targets, the street, the goal, the flag and the textured boxes use textures.
- Moving Objects
  The tank and the boxes moves around.
- Documentation
  You are reading it right now.
- Adjustable Parameters
  The Screenresolution can be changed by adjusting the window. The Fullscreen can be toggled by F11, v-sync can be toggled by pressing F3, the brightness can be changed by pressing the up/down keys. Additionally the Size can be passed with command line arguments.
- Physics-Engine
  We are using PhysX[1] for rigid-body-simulation.
- Heads-Up Display
  We are using the freetype library[2] to render text on the screen. It can be toggled F-6.

5 Effects

- Shadow maps with PCF
  Everything throws a shadow.

- GPU Particle System using Compute Shader
  The explosion is made out of particles.
- Hierachical Animation
  The tanks target pipe can be moved relatively to the tank body when holding down the right mouse button.
- Video textures
  The goal is video of a turtle that is running on a treadmill
- Environment map
  We use a Cubemap for the background
- Cel Shading
  Everything textured is shaded by our Cel Shader.
- Contours via edge detection
  We draw the edges black by applying the sobel filter on a normal and a depth framebuffer.

6 Libraries and Sources

Library/Source name	**Version**	**Usage**
Assimp[3]	4.1.0	To load our models
Freetype[2]	2.10.0	To load "*.ttf" files for text rendering
Glew[4]	2.1.0	OpenGL loader
Glm[5]	3.2.1	General maths
Glfw[6]	0.9.9.4	Window management
PhysX[1]	4.1	Physics calculations
Stb_image[7]	-	Texture loading
ECS-Tutorial[8]	-	ECS-System for our game logic
Car-Physics[9]	-	Arcade-style car logic

7 Tools

We used Blender to model our Objects, MS-Paint and GIMP for the textures.

8 Implementation

We structured our Program in classes. Each class has certain reponsibilities.

- Manager
  Contains references to all important classes.
- Window
  A wrapper for the window management. It also wraps the functionality for the callBacks for mouse and keys, aspectRatio, swapBuffers etc.

- Input
  Wraps the input once more. The callbacks of Window call the static methods in input. In input all events are saved for later use.
- Camera
  Contains logic for the camera.
- GameLoop
  Contains functionality for an independent framerate.
- Renderer
  Contains all functionality for rendering, such as setting correct vaos and drawing the meshes.
- GameLogic
  Creates “GameObjects“ and calls their update method. GameObjects are implemented using the ECS Design Pattern.
- Physics
  Wrapper for the physx library. It has wrapper methods for creating, adding and updating dynamic and static Meshes.
- LevelLoader
  We implemented a LevelLoader to load new Levels and keep track of the gathered flags and targets.
- ECS
  We implemented an Entity Component System to structure all our interactions.

9 Building/Compiling

Since we have built all our library in static-linking mode (No dll's), some "*.lib" files that were created became very large, expecially the "assimp.lib" file. Because git has a file-size limit, which was exceeded by this library file, we had to compress it and put it inside the "InitialTLibs/assimp_4.1.0/debugLib" folder. Before building our program in debug mode, the "assimp-vc140-mt.zip" file has to be decompressed.

The visual studio project settings are only set up for 32bit compilation. 64bit should also work, but the libraries are not compiled for 64bit.

References

[1] `https://developer.nvidia.com/physx-sdk`.

[2] `https://www.freetype.org/`.

[3] http://www.assimp.org/.

[4] http://glew.sourceforge.net/.

[5] https://glm.g-truc.net/0.9.9/index.html.

[6] https://www.glfw.org/.

[7] https://github.com/nothings/stb.

[8] https://www.youtube.com/playlist?list=
PLEETnX-uPtBUrfzE3Dxy3PWyApnW6YEMm.

[9] https://www.youtube.com/watch?v=LG1CtlFRmpU.