Es war einmal ein Virus

This project is created for the course "Computergraphik" at Vienna University of Technology.

The story

It's the year 2030 and all efforts to stop the virus that shall not be named have failed and most scientists have been killed by the tinfoil gang. You’re the last surviving virologist and have developed a micro-submarine that can move in the bloodstream and shoot the virus.
Go save humanity!

Game description

The game consists of navigating inside the vein with the submarine and destroying the virus particles that have infected the human body. To navigate inside the vein, the player can move up/down/left/right with the keyboard keys WSAD and change the view with the mouse movement. The power of your engine can be controlled with the mouse wheel. The task of the player is to move the submarine without hitting any obstacles (blood cells) and annihilate the virus by shooting at them. To stay alive, the player can dodge virus attacks and collect artifacts he encounters on his way to increase his life.

Goal

The player wins if she/he kills all the virus particles and manages to stay alive until the end of the level.

How to start

Run EsWarEinmalEinVirus.exe

How to play

Controls summary

- W, S - move the submarine up/down
- A, D - move the submarine left/right
- LMB - fire weapons
- Mouse - move view
- Mouse wheel - change speed
- Backspace - stop the submarine
- Ä - toggle console (for commands see section "console" or assets/functions.lua)
- 1 - switch to torpedo weapon
- 2 - switch to particle cannon weapon

Others

- F1 - Toggle wire-frame mode
- F2 - Toggle back-face culling
- F8 - Toggle View-Frustrum Culling (not in demo 1)
- F9 - Toggle HUD
- F10 - Toggle Bloom
- Esc - Exit the game

Content

A first level will be automatically loaded and put the player in control of the submarine. The default weapon (torpedo) can immediately be used to shoot viruses and tinfoil hats. Ammo for a second weapon (Particle Cannon, switch with 2 button) can be found later in a crate. Upon losing enough health from bumping into things or getting shot the player can die. Player health can be boosted by collecting a health sphere artifact. The game will end upon reaching zero health (lose) or upon killing all viruses.

Status of project goals

Mandatory gameplay

- 3D geometry: done. Implemented geometries can be found in the "models" folders, especially loading from an external file in ExternalMesh.cpp; ModelFactory.cpp provides an api for geometry creation and a cache for performance reasons
- Playable: done
- Advanced gameplay: done

- Min. 60 FPS and Framerate Independence: code is theoretically fps independent (physx will be updated every 1/60 s, inputs are scaled by elapsed time)
- Win/Lose Condition: done. win: kill all viruses, lose: health reaches zero
- Intuitive Controls: done. sub controls are sluggish by design; interaction with the player character is implemented in PlayerControls.cpp
- Intuitive Camera: done, Camera follows the player and can be rotated with the mouse, code can be found in Camera.cpp
- Illumination Model: basic phong model with or without textures implemented (assets/*.glsl), objects have materials (Material.cpp) and normals (in the various geometry classes). Lights can be specified in the level definition and will be rendered accordingly, the sub has it's own flashlight See LightTypes.h or level.schema.json for possible lighting options
- Textures: done, see Material.cpp
- Moving objects: done. all objects' movements are controlled by physx and rendered accordingly. the submarine can be influenced by the player
- Documentation: this file
- Adjustable parameters: done. a config file (assets/settings.ini) is present using the ECG framework

Optional gameplay

- Collision detection: done. physx raises handles the effects of collisions on movement and raises additional callbacks that are appropriately handled (e.g. to apply damage). See PhysxCore.cpp, PlayerControls.cpp, Scene.cpp
- Advanced physics: done. see above
- Heads up display: done, see Hud.cpp
- Scripting language integration: done, see section "Console"
- View frustrum culling: done, see Frustum.cpp

Effects

- Lighting: Shadow Map with PCF: done, the sub's lights throw shadows
- Advanced Modelling: GPU Particle System using Compute Shader: done, used for explosion (Explosion.cpp), particle cannon (ParticleCannon.cpp) and some fireworks if you win
- Post Processing: Bloom/Glow: done (with HDR), for particles and directly lit objects

Console

The console can be toggled with the key specified above in section "How to play" and supports the following commands

- setHealth(newHealth): sets the player's health to the integer value of newHealth, example: setHealth(2000)
- destroy(name): destroys the entity with the name specified in the level file, name must be quoted, example: destroy("Player")
- loadLevel(name): loads the level with the given name, example: loadLevel ("dummy") will load dummy.json (and destroy the entire current level first)
- addWeaponAmmo(weaponID, amount): (shorthand aWA) adds ammo for the selected weapon, IDs are the same as the keys, example: aWA(2, 10) adds 10 shots with the particle cannon weapon

Used libraries

- conan (https://conan.io/) - Package management
- entt (https://github.com/skypjack/entt) - Organization of game data
- nvidia physx (https://developer.nvidia.com/gameworks-physx-overview) - Physics engine
- assimp (https://github.com/assimp/assimp) - Model loading
- rapidjson (https://rapidjson.org/) - Level loading
- lua (http://www.lua.org/) - Debug console
- sol2 (https://github.com/Rapptz/sol/) - Lua integration
- entt-meets-sol2 (https://github.com/skaarj1989/entt-meets-sol2/) - helpers for using Lua with entt

License

- Tinfoil Hat by tamzidfarhan is licensed under Creative Commons Attribution-ShareAlike (http://creativecommons.org/licenses/by-sa/4.0/)\
- 45-36AN by United4192 is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/)

Authors:

- Marcus Alberer - malberer
- Giulia Gallico - codeAllien