

Mandatory Gameplay Features:

Non-Trivial 3D-Geometry (6 Points)

A complex model for the player ship is loaded from an external file. For this, the library ASSIMP is used. Currently, the gltf format is used to import the player ship. We also created a model for an enemy drone in Blender for which the .dae (Colada) format was used.

Playable (3 Points)

The game is playable. The player can fly the ship and dodge obstacles.

Advanced Gameplay (3 Points)

There are multiple gameplay mechanics such as flying to dodge obstacles, destructible obstacles, shooting down enemies, a health bar, shields and the special ability to tilt the ship. It has a complete level with a goal at the end which ends the game. (to do: show "Game Over" or "Mission Accomplished" on the screen for game over or winning.

Min. 60 FPS and Framerate Independence (3 Points)

Game is updated with delta time. Window is set to 60 FPS in ini file.

Win/Lose Condition (3 Points)

Game Over if player takes too much damage. Win if player reaches the end of the level.

Intuitive Controls (2 Points)

Player ship can be controlled with keyboard (Arrow Keys). Implemented with the InputHandler class using key callbacks and polling for input.

Intuitive Camera (2 Points)

Camera follows behind the player. With keyboard button "1", the camera can be switched to a freely moveable debug camera. (if developer tools are enabled in the settings.ini file) With button "2" the game can be paused.

Illumination Model (2 Points)

Phong shading is used together with textures. Each object has a material.

Textures (2 Points)

Textures are used.

Moving Objects (2 Points)

Player Ship, Projectiles, Enemy Drone and destructible tiles are moving.

Documentation (1 Point)

This.

Adjustable Parameters (1 Point)

- Screen Resolution (Width/Height) -> can be set in ini file
- Fullscreen-Mode (On/Off) -> can be set in ini file
- Refresh-Rate (glfwWindowHint(GLFW_REFRESH_RATE, ...)) -> can be set in ini file
- Brightness (Projectors are usually very dark, this parameter should control how bright your total scene is, e.g., an illumination multiplier)
- Bloom – enables the bloom effect
- Lightmaps – enables the use of lightmaps for all static objects
- Developer Tools – enables the option to switch to a mode that shows the physX actor shapes (toggled with 'F3' key) and also debug camera (toggles with '1' key).

Optional Gameplay Features:

Collision Detection (4 Points)

Implemented collision detection with Nvidia physX engine using physX callbacks.

Advanced Physics (6 Points)

Projectiles can be shot. They collide with enemies and can also be used to destroy a wall consisting of dynamic rigid body bricks. The player ship also collides with static objects and enemy projectiles. The ship automatically slowly aligns back to its normal position after a collision with a static object or projectile.

View-Frustum Culling (6 Points)

The level consists of 9 segments. Each of these reference a bounding box (a physX Trigger shape). If culling is activated (can be toggled on and off with F8) the section will only be rendered if the ship overlaps with the corresponding bounding box.

The number of drawn and culled objects is printed in the console if culling is activated.

Heads-Up Display (4 Points)

An HP bar is rendered as a HUD element in the top left corner. The HP bar color and length is a representation of the players health. Text is also rendered in the HUD to indicate the number of shields the player has. The player starts with 3 shields and each hit destroys a shield. If no shields are left, the next hit will lower the players HP. The text is rendered using a font texture that was generated with CBF (Codehead's Bitmap Font Generator). With the Glyph class, the letters are mapped with coordinates and drawn onto quads.

Total Gameplay Points: 50

Effects

Lighting

Lightmap using Separate Textures (8 Points)

The lightmap textures are generated in Blender using the tool 'Baketool'. This tool makes it easy to generate separate lightmap textures for all objects in the scene. Since the lightmap files are named after the objects-node's name, and this name is included in the gltf model file, our asset importer automatically checks if a corresponding lightmap exists and adds it to the imported mesh's material. Unfortunately, due to the limited file size of the submission, lightmaps we had to use very low resolution lightmap files.

For objects that have light maps, the lightmap is added on top of the base color and all static lights are ignored since their illumination is already included in the lightmap. Dynamic lights (like projectiles) are still applied normally.

Lightmaps can be toggled on and off in the settings.ini file

Advanced Modelling

CPU Particle System (8 Points)

A Particle System was implemented to simulate a trail of energy coming out of the engines on the back of the ship. The tutorial from <https://learnopengl.com/In-Practice/2D-Game/Particles> was used as a reference and adapted to make use of a geometry shader. The particles are created and managed by the ParticleEmitter class. The point coordinates of the particles are loaded into a buffer and passed to the geometry shader. The geometry shader then creates little quads around these points that are facing the camera ("billboards"). The tutorial from <https://ogldev.org/www/tutorial27/tutorial27.html> was used as a reference.

Special Features:

- **Moving and shooting enemies** – alien drones are patrolling the level and can be shot down
- **Health upgrades** – they are big blue spheres that can be collected to restore 50% health
- **Shield** – the player starts with a shield that absorbs 3 hits.
- **Projectiles** – projectiles have a light source attached to them to give them more realism.

Animation

Hierarchical Animation (4 Points)

The enemy drone has a propeller on top which is spinning while the drone itself is moving. The transformation matrix of the propeller model is multiplied with an animation matrix after being multiplied with the transformation matrix of the parent.

Texturing

Environment Map (4 Points)

Since our scene already uses a cubemap as a background, it was very simple to also add environment mapping to simulate reflective objects. The only reflective objects are the stilts under an elevated tunnel. For this we use a shader that calculates the reflection vector and then uses it to sample the cubemap texture.

Shading

Simple Normal Mapping (4 Points)

For models that have a normal map this normal map texture is automatically imported. The tangent space is calculated by assimp using the 'aiProcess_CalcTangentSpace' post-processing option. In the vertex shader the TBN Matrix is calculated using Tangent, Bitangent and the vertice's normal. Then, in the fragment shader the normal is sampled from the normal map texture and transformed using the TBN matrix. This effect is best noticable on concrete walls on the side when lightmaps are deactivated in the ini file.

Advanced Data Structures

Bloom/Glow (8 Points)

The bloom effect was implemented following this tutorial. <https://learnopengl.com/Advanced-Lighting/Bloom>. A Framebuffer class was created to manage the additional framebuffers which are needed. Each frame is divided into 2 images: one containing only the very bright parts of the image and the other one the normal image. The bright part is then blurred multiple times using a gaussian blur filter (inside a shader, using ping-pong framebuffers) and then the blurred bright image is added on top of the normal image to achieve the bloom/glow effect.

Total Effect Points: 36

State of the Game:

Controls:

Move: move left (left arrow), right(right arrow), up(down arrow), down(up arrow).

Shoot: enter key

Fully tilt ship: space

Toggle wire-frame mode: F1

Toggle backface culling: F2

Toggle frustum culling: F8

Switch to debug camera: 1 (only if Developer Tools are enabled in settings.ini)

move debug camera: WASD

turn debug camera: mouse

Pause game: 2

physX debug view: F3 (only if Developer Tools are enabled in settings.ini)

Level:

The Level geometry has been created in Blender and is imported using assimp. Additional level information such as enemy spawn locations are described in the LevelDesc struct. This data structure is used to generate the Scene. In the level there are multiple static obstacles, some of which are placed in a way that required the tilt function of the ship (activated with space bar) to pass them. Enemy drones will shoot at the player if he is in front of them, they can be shot down by the player. There is a brick wall which fully block the way but can be shot apart by the player, otherwise the wall will be destroyed when the player crashes into it, which would harm the player. Two health orbs (blue spheres) have been placed in the level, they can restore up to 50% of the player's health.

Used Libraries:

GLFW

GLM

ASSIMP - <https://www.assimp.org/>

Library used for importing 3D assets. Only used for loading player ship as of now.

Nvidia PhysX - <https://www.nvidia.com/en-us/drivers/physx/physx-9-19-0218-driver/>

Physics library used for player movement, simulation and collision detection.

DevIL – Library used for loading texture images.