

Documentation

Group/Game Name

Unmapped (Daniel Hiebeler, Mathias Johannsen)

Brief description of implementation

The game is using OpenGL and we tested it on a Nvidia graphics card.

The window size and fullscreen mode can be modified in assets/settings/window.ini

Additional libraries

- Jolt physics (<https://github.com/jrouwe/joltphysics>)
- Assimp (<https://www.assimp.org/>)
- FreeType (<https://freetype.org/>)
- stb_image (<https://github.com/nothings/stb>)
- nlohmann/json (<https://github.com/nlohmann/json>)

Mandatory Gameplay

3D Geometry

The map contains non trivial game objects like trees and houses that we created in Blender our own.

Playable

The game is playable. It allows the user to drive around a map with the car. By pressing the key P, the camera can be switched to flying mode, which lets the player fly around the map. This is more or less a debug mode.

Min. 60 FPS and Framerate Independence

The game should run with more than 60 FPS and we are using delta times between frames, to make it framerate independence.

Win/Lose Condition

The player has to find and reach the drop off point before the time runs out. The drop off point is a yellow box with yellow smoke coming out of it. If the player reaches the point in time, the player wins. If not, the player loses. In the pause menu (ESC key) the game can

be restarted. Optionally by pressing key T, the timer can be disabled to just explore the map.

Intuitive Controls

The car is controlled using arrow-keys. By pressing ESC, the pause menu can be opened, where all keybindings are explained.

Intuitive Camera

There are two camera options. In normal game mode, the camera follows the car. In flying mode the camera can be freely moved and rotated.

Illumination Model

The only light source in our game is the sun and each model in our map has a material. The normal maps vectors are provided by the blender models.

Textures

Every object in our map has a texture.

Moving Objects

Our moving object is the car.

Adjustable Parameters

Screen dimensions and fullscreen mode can be set via the window.ini config file.

Optional Gameplay

Advanced Gameplay

The game has complete gameplay. After finishing the game, there is a button to play again. Depending on the positions of spawn and drop-off point the game is really challenging. There is a timer which gives the player one minute to reach this drop-off point.

Collision Detection

The collision detection is implemented between the car and the world objects.

Advanced Physics

The car is implemented using Jolt physics

View-Frustum Culling

The map is stored as KD-tree. View-Frustum Culling can be toggled using the F8 key. When enabled it also displays the number of currently rendered objects. The view frustum culling is always done for the camera perspective of the car, so by switching to flying mode (press key P), it can be seen that objects behind the car are not rendered. By pressing B, the bounding boxes of rendered objects are shown in green and not rendered objects in red.

Heads-up Display

The Heads-up Display shows game instructions, speed, pause menu, winning/loosing menu and other information. In the pause menu the background is darkened using a semi-transparent rectangle. The pause menu includes a clickable button to restart the game. The HUD class provides intuitive functions and parameter to display font, rectangles and sprites. Color, size and position of text can be specified. Text can optionally be centered around the position coordinates.

Effects

Lighting: Shadow Map with PCF

All game objects in the map and also the car itself cast shadows. Shadow mapping is implemented with PCF. Artifacts like shadow acne and peter panning are prevented.

Advanced Modeling: CPU Particle System

The CPU Particle System is used for smoke out of chimneys of all houses. Also the drop-off point simulates smoke using the same particle system class with different parameters (colors, size, time to live, ...).

Animation: Hierarchical Animation

The animated wheels of the car.

Texturing: Procedural Texture

The texture of all streets is generated procedurally. Since the street texture repeats itself, we generated it so that it repeats with seamless tiling.

Advanced Data Structures: KD-Tree

All entities are stored in a KD-Tree which is used for View-Frustum Culling.

Additional Informationen

Map

The map is defined in the JSON file `assets/maps/map.json`. The JSON file contains an array with every object within the map. The entry for a single object has the following structure:

```
{
  "name": "House",
  "model": "../assets/models/house.obj",
  "type": "static_mesh",
  "material": "standard",
  "position": [6.0, 0.0, 21.0],
  "rotation": [0.0, 0.0, 0.0],
  "scale": [1.0, 1.0, 1.0]
},
```

The type defines if physics engine treats it as dynamic or static object. The JSON file is parsed using the `nlohmann/json` library.

Physics

The game uses the Jolt physics engine to handle car physics and collisions between objects. It also handles gravity for dynamic objects.

Model loading

The Assimp library is used to import the 3D models.

Image loading

The `stb_image` library is used to import the images for the skybox.

Font rendering

The FreeType library is used to render fonts in the head up display.