

Submission 2

BoozeWars

Patrick Link (11728332)

Carlos Rodriguez (1635311)

Implementation

In general we tried to implement our game object oriented and well structured. With object for each element in our gameplay, which saves its attributes, behaviours and model. The main file is responsible of the game logic. The scene and its entities are controlled in a separate class.

Free Movable Camera

We provide a free movable camera which is essential for our gameplay. The camera can be moved at all time in the game. The rotation matrix is derived from quaternions.

Moving Objects

We provide a wave of enemies which start to move once the game has been started. They move from the start of the street to the end. The wave can be easily parametrized, by giving the number of enemies, the time delay between enemy spawns and the delay between waves. One can even define multiple waves. The enemies are controlled by the physx engine, as described below.

The barrels rotate and translate, so it looks like they are rolling. Once their life time is over, they scale till they disappear.

Texture Mapping

Each of our object has its own texture. The sky and the map have customized textures. We use mipmaps and bilinear filters for the enemy and building objects. We choose the nearest mipmap level and bilinear interpolation of the texture.

For the skybox with used cubemap textures.

Text Loading

We use freetype to load and display text in the view.

Object Loading

By using assimp we load the models and its meshes and save the template parts for each. When calling draw we iterate through the meshes and display them. The meshes are loaded

from a .obj file while the meshes and the textures are connected by a .mtl file. The textures are images.

Every Object is loaded in the ObjectFactory which instantiates once in a programs life time (Singleton). This Factory creates all objects in its constructor and hands it to the game if required.

Loaded objects are:

- Buildings (saloon)
- Weapons (Barrels)
- School
- Street Light
- Enemies (bottle and beer glass)

Shadow Mapping

We implemented shadow mapping for the directional light. The scene is first rendered from the point of view of the light, we take the middle of the map and use orthonormal projection in direction of the light. The depth values are then saved in a texture. This texture is then used in the next pass along with the coordinates with respect to the light to determine if a fragment is in a shadow or not. We sample 4 points and take the average so that the shadow has a smoother appearance. The shadow pass is done with front face culling to reduce shadow acne.

Physx Engine

The enemies are controlled by the physx engine. A driving force pulls the enemy in direction of the goal. Moreover one can place walls in the way, which are implemented as static rigid bodies. One type of enemy casts three rays and uses the collision information to find a path, the second type casts one ray, and if it collides with an obstacle it jumps over it. This is all implemented with forces and impulses.

Effects

CPU-Particle System

We generate 500 particles for each weapon (barrel), which spread randomly in all direction, once the barrel destroys, which creates a hierarchical animation.. The direction is determined by a randomly chosen point on the sphere, parametrized by standard spherical coordinates. Their opacity gets less depending on the time and they die once the opacity is 0 or their time to life is over. We used instanced drawing to render the particles.

Light mapping

We implemented light mapping using a separate texture holding the light information, which was calculated using blender. Unfortunately as our scene is very dynamic, we can't do light mapping for many objects. In fact only the school building uses light mapping. We used several point lights and area lights to get a nice atmosphere.

Procedural textures

We used procedural textures for the ground plane. We used perlin noise n , from which we took the quantity $f = 1 - n * n$. This has nice ridge like features. Moreover to make the texture fit to the visual style of the rest of the game, we then used a step function to get discrete values for f . The so calculated f is then used to mix a foreground color corresponding to grass and a background color representing the dirt. For the street, we just took $1-f$ and the same colors. If the current fragment belongs to the street or to the world is sampled from a black and white texture.

Cel Shading

In our current state the lightning comes from the sky which is directional lighting. We have also one point light. The lightning calculations is done with cel shading. We implemented our own cel shader. It calculates the dot product of the normal and light direction and then bins it with the formula $x \rightarrow x \% 4$ to get discrete shades.

Contours (edge detection)

Outlines of our objects are generated in two ways. Firstly a sobol filter finds discontinuities in the depth buffer, secondly we use a LoG filter to find places where the surface normal changes. If these values are above a certain threshold, we paint a black pixel there. This is done as a postprocessing effect. This means that the the screen is first rendered to a texture, where the color, the normals and the depth value are rendered into different textures. These information is then used to calculate the outlines, as described above.

Controls

The main controls are for moving the camera. With W, A, S and D you are able to move camera to the right, left, front and backwards. By pressing R or F the camera can be moved up and down. With Esc the game can be quit and with Enter the wave can be started. With R the building indicator can be hidden and with E the building is rotated. With the mouse you can move the view from the current position. With left click buildings can be placed. A transparent rectangle indicates the size and position of the building that will be placed, with the key Q you can hide this rectangle. Also walls can be placed at the street. With the number keys 1 and 2 you are able to switch between the buildings and the wall.

Other interesting controls:

F1 - Help

F2 - Frame Time on/off

F3 - Wire Frame on/off

F4 - Shadows on/off

F8 - View-frustum Culling on/off

Basic Gameplay

The goal is to survive a wave of enemies. First you are able to place buildings all around the street. There is no need of placing all buildings before starting. After placing min. one building you are able to start the game by pressing enter. During the wave you are still able to place buildings and every 10 enemies you get to place one additional building. If you have placed enough buildings, you will survive the wave. You win the game, if you survive all waves. If an enemy reaches the end of the street, the player loses 5 to 10 hp of the initial 100. You lose if your life turns zero.

To help and make the game more interesting one can also place walls on the street.

Features

Collision checks

When placing buildings we check if at the current position is the street or if it is colliding with another building.

Fighting system

Each building has 5 barrels, which are the weapons of the buildings. The barrels roll out one after another and implodes after the range of the building is reached. We detect collisions between the barrels and the enemies and once they hit an enemy, they implode. The enemies get hit and destroy once their life is too low.

Libraries

FreeType for the text on the screen. <https://www.freetype.org/>

FreelImage to import images. <http://freeimage.sourceforge.net/>

Assimp to import models. <http://www.assimp.org/>

Physx for physics calculations. <https://developer.nvidia.com/gameworks-physx-overview>