Submission 2 - Documentation

June 19, 2017

Contents

1 Brief description of the implementation

We have created the following classes for our implementation:

BoundingBox.h : Used for calculation of the bounding boxes (simple cubes) and storing their values, like position, collision detection matrices, center and size of the object - this is also used for drawing a bounding box

Camera.h : Used for a creation of a camera with its values, like its position, its front, up and right vector, for keyboard callbacks and its name

ColObject.h : Used for storing values for the collision detection - just a simple container

Font.cpp & Font.h : Used for the creation of a font and text rendering - has only an initializer and a render text method

Mesh.h : Used for storing the values of a loaded mesh, like its vertices and its textures and for drawing it

Model.h : Used for loading a model and storing its values - contains also its bounding box (see BoundingBox)

Particle.cpp & Particle.h : Used for a creation of a particle, storing its values like lifetime, position, speed, etc.

Shader.h & Shader.cpp : Used for the creation of a shader - takes as a first parameter the location of a vertexshader, as second the location of a fragmentshader and as an optional third parameter the location of a geometry shader

main.cpp: Contains the main logic of the game, contains the view, contains the game loop, contains the effects, contains keyboard callbacks, etc., separated into several regions (#pragma region)

We have used several directories for further separation:

Controller: Actually only a helper directory for classes used for texture loading (texture.h & texture.cpp) and print functions (print.h)

Persistence: Used for static files. It has following subdirectories:

Fonts: Stores several fonts (currently has only one font: arial.ttf)

Objects: Stores .obj and .mtl files (created with Blender)

Textures: Stores texture files (stores the 'woodbox.png' file we use for texturing a cube

Shader: Used for storing our different shaders

2 Features of the game

Nightmode: Switch between night and day

Flying over water: You can move the player over the water without dying

Freely movable camera (world camera)

Sun and Moon: Only orbits, rotating around the world (they have no specific task)

Levitating objects: flying, rotating diamonds and hearts, who have to be collected in order to win

3 Illuminated or textured objects (description of light sources)

Illuminated:

Every object, that is under the light source, is illuminated

Textured

Textured Cube: you can see it on the left bottom corner (if you move the camera in that direction) - feature, because it couldn't be seen in the first assignment

Textured text: you can see it in daymode on the upper right corner: it tells you how many diamonds / hearts there are left until you win

4 Additional libraries

Assimp for model loading (http://assimp.org/)

SOIL for texture loading (http://lonesock.net/soil.html)

FreeType for font and text loading (https://www.freetype.org/)

5 Effects

Shadow Mapping

Omnidirectional Shadow Mapping

GPU Particle System

Bloom

6 Implementation of effects

Shadow Mapping:

Implementation details: We create two images: a depth map as seen by the light source with a specific viewport (SHADOW_WIDTH and SHADOW_HEIGHT), which will be bound in the main method and a normal image, which is rendered with a shadow shader, which renders the shadows in the correct perspective.

Reference: https://learnopengl.com/#!Advanced-Lighting/Shadows/Shadow-Mapping

Omnidirectional Mapping:

Implementation details: This is an advancement to the simple shadow mapping - It creates 6 depth maps for a depth cubemap. This will then be used for creating a better shadow.

Reference: https://learnopengl.com/#!Advanced-Lighting/Shadows/Point-Shadows

GPU Particle System:

Implementation details: Used the Particle class. Used for storing lifetime, velocity, xyz-Coordinates. The particles for the collectible objects (hearts and diamonds) and the actual objects are separated in different vectors. On creation we translate the particles to the position of the player and on rendering we calculate different x,y and z coordinates for every particle.

Reference: https://learnopengl.com/#!In-Practice/2D-Game/Particles

Bloom:

Implementation details: We create two render images: once the normal scene, the second one is a threshold image (only the bright objects will be rendered). We blur the threshold image, combine it with the normally rendered image and get a bloomed image.

Reference: https://learnopengl.com/#!Advanced-Lighting/Bloom

Note: We tested our game at the AMAROK PC with the NVIDIA Graphics Card and mostly everything worked. Except the simple shadow mapping, which also won't work on the AMD Graphics Card. There will be a screenshot provided, which proves, that it works on Thomas's laptop.

7 Other special features: Controls

Help on F1

8 Tools used for model creation

Only Blender