Defenders of Cthedra

2st Submission Documentation

UE Computergraphik

STUDENTS:
Robert Janca (0926443)
Ernad Sehic (1227865)

Wien, Juni 2015.

1. Description

DoC is a shoot'em up game where the player controls a combat aircraft. The player has to destroy enemy aircrafts and prevent the final boss from destroying the city.

2. Story Board

The planet Cthedra is under attack by an enemy force. A giant mothership is on its way to the biggest city and wave after wave of invaders are descending upon the surface. Defend the city against the invading army and destroy the mothership.

3. Gameplay

The player engages in aerial combat with the enemy aircrafts and must destroy a certain number of ships while not receiving critical damage. Once the mothership is in range, the player must destroy its main cannon to prevent the destruction of the city.

4. Controls

Key	**Effect**
W	Up
A	Left
S	Down
D	Right
ESC	Quit
ENTER	Start game
SPACE	Fire
O	Enter win state
P	Enter lose state

5. Technical Details

5.1. Types of objects in the game

- Static meshes (Platform, Terrain, Wall, Laser)
- Complex Objects (Mothership, Pyramide, Player, Enemy)
- Animated Objects (friendly ship with rotating ring)
- Transparency (Debug overlay can be set to transparent)
- Light sources (bullets)

5.2. Camera

- 3rd person camera behind the aircraft, following its movements.

5.3. Illumination

- Directional light source

5.4. Effects (for evaluation)

- Vignette with noise-based blur (0.5)
- Bloom (1.0)
- Lens flares (0.5)
- Shadow maps with PCF (0.5)
- Lightshafts (1.0)
- Cel Shading (0.5)

5.5. Additional Effects

- Deferred renderer
- HDR with fixed tonemapping
- FXAA
- Omni-directional shadow mapping
- Gauss-blur based depth-of-field
- Fog (disabled)

6. Requirements Description

6.1. Freely movable camera

The camera is not controllable by the player itself, he cannot perform geometric changes to the camera such as rotation or translation (3rd person camera). The Camera follows the player‘s ship from behind, and it cannot change until the player perform an action on his ship.

6.2. Moving objects

The game provides object movement through animations, which are defined in .json files. Different animation controllers provide the functionality and reliability for our animated objects, for example, in the Title Stage the floating of the player’s ship and the rotating platform under it.

6.3. Texture Mapping

The game has an own texture mapping system, and all of our objects are textured.

6.4. Simple lighting and Materials

The game has an own material system, so for each object a material has been assigned, and all objects are textured. The light source, as you can see in the Title Stage is a directional light source. The normal vectors are handled properly.

6.5. Controls

User inputs are handled by an input provider, and for now this are the working controls:

Key	Effect
W	Up
A	Left
S	Down
D	Right
ESC	Quit
ENTER	Start game
SPACE	Fire
O	Enter win state
P	Enter lose state

6.6. Gameplay

In this version of the game the basic gameplay contains just ship movements by the user, and the camera which follows the ship. With the SPACE button the play can fire. After three seconds enemy ships being spawned. Till now we have not implemented a collision system, so we can’t hit them, so till now we don’t have a Win State. If the player don’t destroy all enemy ships till the mothership arrived (for now cca. 3 min) he loses -> Lose State. If you shoot all enemies before the mothership arrives -> Win State.

6.7. Effects

- **Vignette with noise-based blur (0.5)**
  The scene texture is slightly distorted at the edges to simulate camera lense imperfections.
- **Bloom (1.0)**
  Bright areas are extracted with a brightpass shader, blurred and additively blended on the scene texture.
- **Lens flares (0.5)**
  Selecting the subset of the brightest pixels in the source image to participate in the lens flare, generating ghosts pivoting around the image centre.
- **Shadow maps with PCF (0.5)**
  PCF has been enabled for the existing shadow maps.
- **Lightshafts (1.0)**
  Screen space light shafts are simulated by extracting bright areas from the scene, performing a radial blur and blending the result additively on the scene texture.
- **Cel Shading (0.5)**
  First we find the color channel with the highest value. This value is discretized by the toonify function and saved as a factor. Finally, the original color is multiplied by this factor and saved as the shader result.

6.8. Complex Objects

Our ships were modeled by ourself in Blender. We have a .obj loader librarie (tinyobj) which loads our objects, so we can use them in our game.

6.9. Animated Objects

We provide animated objects at the Load Screen with rotating rings, and at the Play Stage where a ring rotates around the friendly ship (can’t be destroyed).

6.10. View-Frustum-Culling

View frustum culling has been implemented by providing bounding spheres for all geometric objects and culling these bounding volumes against the view frustum.

6.11. Transparency

Due to the defferred shader, the only transparent object is the debug overlay (enable with F2). The transparency can be toggled on/off with F9.

6.12. Experimenting with OpenGL

All textures are generated with mip maps as default setting. The renderer supports two draw modes based on the data in the mesh (indexed if a index buffer is present, non indexed otherwise).

7. “Features” of the game

- Easy change from 3rd person camera to movable camera (can be done through the config file, but not provided for the user).
- By pressing F2 on the keyboard you can see the debug overlay (while in debug mode you can press F9 to turn on transparency).
- By pressing F3 on the keyboard you can turn on/off Wire Frame.
- By pressing F8 you can turn on/off frustum culling.
- By pressing 1 on the keyboard you can see the mouse capture.
- By pressing 2 on the keyboard you can change to deferred renderer (default).
- By pressing 3 on the keyboard you can change to forward renderer.
- By pressing ENTER on the keyboard you can enter the Game State from the Title State.
- By pressing ESC on the keyboard you can exit the game.
- By Pressing O you can switch from Play State to Win State.
- By Pressing P you can switch from Play State to Lose State.
- By Pressing K/L in Play State you can turn on/off Cel Shading.

8. Additional Libraries

- FlextGL - an OpenGL extension loader generator.
  Source: https://github.com/ginkgo/flextGL
- GLFW - library for creating windows with OpenGL contexts and receiving input and events.
  Source: http://www.glfw.org/
- JsonCPP - a C++ library that allows manipulating JSON values.
  Source: https://github.com/open-source-parsers/jsoncpp
- LodePNG - a PNG image decoder and encoder.
  Source: http://lodev.org/lodepng/
- Tinyobj - .obj loader.
  Source: http://syoyo.github.io/tinyobjloader/
- irrKlang - sound library for C++ (not used yet)
  Source: http://www.ambiera.com/irrklang/

9. Effect references

- Lens Flare - http://john-chapman-graphics.blogspot.co.at/2013/02/pseudo-lens-flare.html
- Lightshifts - https://http.developer.nvidia.com/GPUGems3/gpugems3_ch13.html
- Vignette - https://github.com/SFML/SFML/wiki/Source:-HeatHazeShader
- Bloom - http://prideout.net/archive/bloom/
- Cel Shading - https://kbalentertainment.wordpress.com/2013/11/27/tutorial-cel-shading-with-libgdx-and-opengl-es-2-0-using-post-processing/
- Other: OpenGL Superbible 5th &6th Edition