Antarctic Tale: 3. Abgabe

1 Story

„Once upon a time... Tux wakes up after the snowstorm and his family is gone... Help him to find his lost family...“

2 Technical Overview

The entry point to Antarctic Tale is in main.cpp, which is responsible for handling input from peripherals, initializing libraries and initializing a GameState object. The GameState object, in turn, is responsible primarily for maintaining the game's state, including dynamic and static game objects, key bindings, images, etc. The GameState class populates its state variables from XML 'level' files stored in the data directory (cf. data\levels*.xml).

All game objects are instantiations of children of the GameObj class. The GameObj class provides basic functionality common to all game objects, including position and orientation, model and texture, and bounding sphere. The children of GameObj are DynamicGameObj and StaticGameObj. The former's children (for now) are Tux and AIGameObj; the latter has no children. The HeightMap class is responsible for the terrain, which it generates based on a grayscale image.

3 Libraries

- DevIL: loading images from files (cf. image.cpp and main.cpp)
- GLFW: windowing and peripheral input handling (cf. main.cpp)
- GLEW: used for shader support (cf. main.cpp)
- GLUT: used strictly to draw text to the screen (cf. main.cpp)
- TinyXML: XML parsing (cf. gamestate.cpp)
- bmp.cpp: used for some texture loading

4 Tools

4.1 Objects

Objects were obtained from the web and at most lightly modified using a trial version of 3D Studio Max.

5 Requirements

5.1 Effects

- Environment mapping [1]: there is a cubemap [2] applied on the secret fish in the first level and also in the last level there is a cubemap on the cube. This is dynamic and renders every frame.
- Bloom [3] effect: this effect is visible on the snow, but also on the pinguins head in the 2. level and on the mysterious cubes in the last level
- Per-pixel lighting in fragment shader: based on the tutorial in a link. This shader is applied on the objects.

[1] http://www.ozone3d.net/tutorials/glsl_texturing_p04.php
[2] http://www.sulaco.co.za/opengl_dynamic_cube_mapping.htm
[3] http://www.online-tutorials.net/grafik-effekte/bloom/tutorials-t-63-114.html
[4] http://www.lighthouse3d.com/opengl/glsl/index.php?dirlightpix

5.2 Animated Objects

Animation was implemented in the keyframe manner (cf. dynamicgameobj.cpp). Models were exported to the Collada format.

5.3 Frustum Culling

Implemented (cf. display() in gamestate.cpp, under „if (toggleFrustumCulling)“). Observe that the viewing frustum is intentionally scaled by 1.3 in order to widen it and make it less likely that objects appear to vanish when neer the sides of the viewport.

5.4 Immediate Mode/VBO/Display Lists

Not implemented, except for immediate mode.

5.5 Wireframe Mode/Framerate and Other Debug Info

The following F-keys are fully supported according to project requirements: F1 (toggles text box), F2 (toggles displaying the framerate on the screen), F3 (toggles wireframe mode), F4 (toggles texture filtering for objects), F5 (toggles mipmapping modes for the terrain), F8 (toggles view frustum culling), F9 (toggles transparency for the surface). F10 toggles shaders, which is useful for checking mipmapping and texturing details.

Note that since is it difficult to see mipmapping on our snow surface, we suggest

opening one of the xml level files (e.g., data\level1.xml) and changing

```
<HeightMap imagePath="data\heightmaps\maze.tga"
texturePath="data\textures\snow_tiny.jpg" x=0 y=0 z=0/>
```

to

```
<HeightMap imagePath="data\heightmaps\maze.tga"
texturePath="data\textures\mur_ambiant.jpg" x=0 y=0 z=0/>
```

5.6 Functioning Control

Tux can be moved using either the arrow or WASD keys. Diagonal movement is supported. Note that control of Tux is intuitive in spite of camera position; e.g., the up key will always move Tux upwards with respect to the viewport, rather than with respect to the Z direction on the terrain terrain. This is achieved by performing a rotation matrix transformation on the displacement vector (cf. tux.cpp).

Camera yaw and roll is controlled by the mouse. The zoom factor is controlled by the mouse wheel.