

Swarm

Submission 2 – Documentation

Configuration

The game starts in 1024x768 in windowed mode by default. To use different settings commandline-arguments are required

e.g. 'Swarm.exe 1920 1080 1' will run the game in high-resolution fullscreen mode.

Implementation

Gameplay

The player controls a swarm of fish and has to pass check points(green pyramids) before reaching the goal (green cube). One can adjust direction and speed of the swarm via mouse movement and scrolling, respectively. The enemy sharks kill members of the swarm on contact and have a patrol and attack mode. After entering the proximity of a shark, it stops patrolling and moves towards the player for a certain time. After that there is a short cool down state such that the shark returns to the patrol path and ignores the player. In addition the water surface level will slowly lower over time, to make it more difficult to maneuver past the sharks (passing the water surface also kills a fish).

The swarm movement is based on real swarm logic in nature (boids framework). Each fish keeps track of its nearest neighbors and tries to stay in a feel-good distance to them. On the start of the game the fish try to find their optimal position starting from a given initial setup. Note that it can happen that the swarm loses cohesion when one accelerates too fast in the beginning. In this case, one has to slow down a little bit to allow the fish to rearrange.

Controls

The movement direction of the swarm is controlled via mouse movement. The speed is adjusted with the mouse wheel. Hitting escape leads to leaving the game. The F1-9 buttons can be used to adapt the visualization according key-mapping chart from the lecture.

In addition F6 toggles the framerate-limit (default is 120FPS) and F7 shows the number of rendered objects (to visualize the effects of view-frustum-culling).

Text Status messages are shown on the screen when a F-button is pressed.

Lighting

The Scene is illuminated by a single directional light source. All objects have distinct specular light-settings and 'react' slightly different to the incoming light. Shadows are cast in the light-source's direction.

Effects

- **Water surface tessellation LOD** (2 effect points) : The surface of the sea is modelled by a

tessellated mesh. The functional wave information is given by a sum of sine functions where each wave moves in a certain direction (based on http://http.developer.nvidia.com/GPUGems/gpugems_ch01.html). There are three tessellation levels (the levels can be seen very good in wireframe mode when you move close to the surface). The implementation is done with tessellation control and evaluation shaders such that the vertex shader just passes on the positions/normals/uvs, the control shader computes view to position distances and sets the tessellation levels which are used in the evaluation shader to compute the sum of sines.

- **Shadow Maps (with PCF)** (1.5 effect points): The scene's depth information is rendered in lower resolution into a framebuffer using a different projection matrix from the sun's direction. This resulting texture is then used in a second render-pass alongside this projection-matrix. In this pass the scene is rendered like normal, but the shadow map (and the calculated coordinates) are used to darken the affected regions of the scene. Percentage close filtering is used to create smoother shadows.

Our implementation mainly follows the slides from the lecture as well as the tutorial at <http://www.opengl-tutorial.org>

- **Bloom** (1 effect point): The scene is rendered in lower resolution into a framebuffer (using the regular view and projection) using a threshold filter in the fragment-shader to create a greyscale texture with increased contrast to separate the bright parts of the image. This texture is then rendered another two times into a framebuffer using a shader that applies a blur-effect. (fractions of neighboring pixel's colors are added to each pixel/fragment) The resulting blurred glow-map is then (after the whole scene was rendered normally) used as texture on a screen-sized quad that is rendered on top of the scene using a shader that only shows the bright areas. The effect can be seen on the sun as well as most specular highlights. (The best way to show this effect is to move the swarm under the sun and look towards it).

Our implementation is based on the guidelines from the lecture as well as the tutorial at <http://prideout.net/archive/bloom/>

Complex Objects

All Objects in the game have UV coordinates and textures. All complex models (Shark, Fish, Rocks) were hand created in 3DSmax (including UV coordinates) and are loaded when the game starts using AssetImporter. Custom (hand-created) textures are used for the Sharks and Fish to match the UV unwrap.

The ground surface is loaded from a heightmap image, and transformed into a mesh at startup by a custom algorithm.

Animated Objects

The sharks are composed of separate meshes: the 3 fins are 'attached' to the main body, and slightly rotate in constant intervals. This should be clearly visible when approaching a shark.

View-Frustum-Culling

All objects in the scene (excluding some fixed level elements) are only rendered when inside the view frustum. For this, each frame the planes of the frustum are calculated and all objects are checked (by distance/radius) if they are outside those planes. The view-frustum-culling can be toggled on and off via the F8-key

Transparency

Transparency is used in various forms in this game. The sun (a view-aligned quad rendered in the sky) uses an alpha level calculated from its texture-color values inside the fragment shader. The checkpoints and the goal area have a 'global' transparency value, that is additionally multiplied with a time-based sinus function to create a pulse effect.

The water surface uses a fixed alpha value.

The Bloom effect renders a textured quad on top of the scene, which has its alpha values based on the texture's brightness.

Transparency can be toggled with the F9-key. This also disables bloom to prevent the whole screen to be covered from a non-transparent glow map.

Sound

All sounds are handled by the Irrklang library. We play a looping 2D ambient sound, some triggered 2D sound effects (checkpoint; shark attack) and a 3D music that 'follows' the sharks around.

Collision Detection

Collision Detections is handled by our own code and supports bounding boxes.

Resources

All used 3D-models and their textures are self-created. The ground texture is from the Total Textures repository and was slightly edited, the water texture is from another image library. Models are in .OBJ format and loaded with AssetImporter. The used textures are bitmap-files that are loaded via an image library (FreeImage).

The sounds are mixed together using sound-files from various online-libraries. The music in use is from the movie Jaws. Sounds are loaded and played via the Irrklang library.

Textures were created/edited in Adobe Photoshop, 3D-models were created in Autodesk 3DSmax, sounds were mixed in Audacity.

Additional Libraries

The following libraries are used:

GLFW - <http://www.glfw.org/> - Open GL window functions

GLEW - <http://glew.sourceforge.net/> - Open GL extension

GLM - <http://glm.g-truc.net/> - Mathematics

FreeImage - <http://freeimage.sourceforge.net/> - Image Loading

Assimp - <http://assimp.sourceforge.net/> - Model Loading

IrrKlang – <http://www.ambiera.com/irrklang/> - Sound

OpenGLUT - <http://openglut.sourceforge.net/> - Text Rendering