Advanced Rendering: Ocean War

Ioannis Tsiompikas

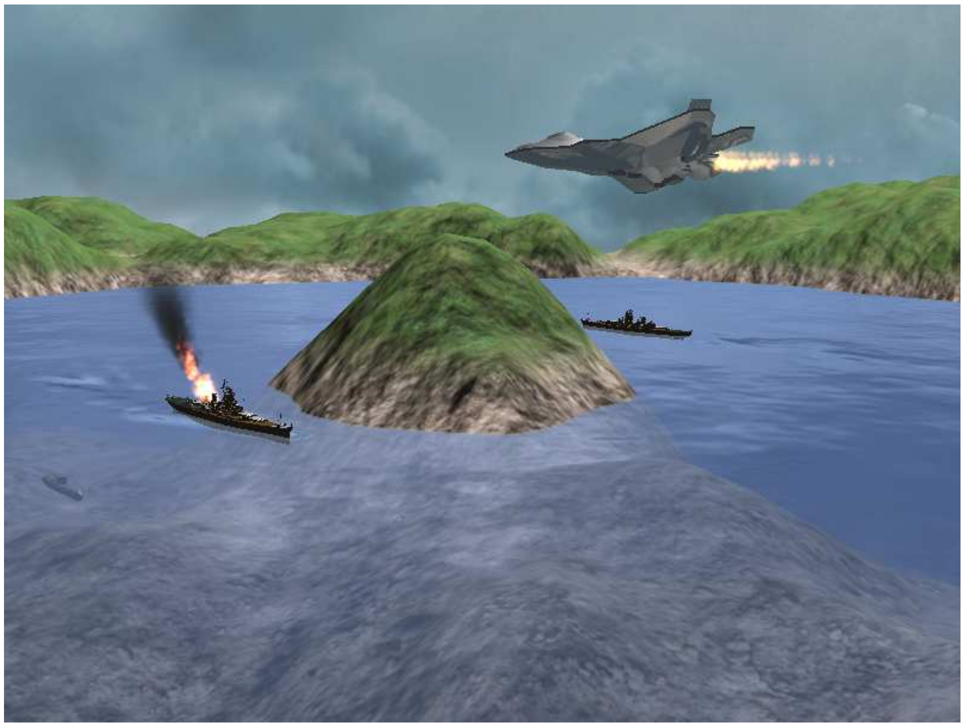
Figure 1: Full ocean battle screenshot.

1 Introduction

The "ocean war" scene was implemented as a standalone C++ program, using GLSL shaders for rendering all objects and effects. The "henge" 3D engine was used, which I developed during the first semester for the graphics coursework, with some modifications to make the particle systems shader-based.

2 Effects

Next, I'm going to briefly describe the various effects implemented for the ocean war scene.

2.1 Textured Battleships

The ship model was loaded from a Wavefront OBJ file, and duplicated to create the two ships. Since it's made up from multiple meshes some of which are textured and some are not, there are two fragment shaders responsible for rendering the ship parts: `blinn.p.glsl` and `blinn_tex.p.glsl`, which implement the blinn/phong reflectance model per pixel with or without texture mapping. A single vertex shader is used for both cases: `blinn.v.glsl`.

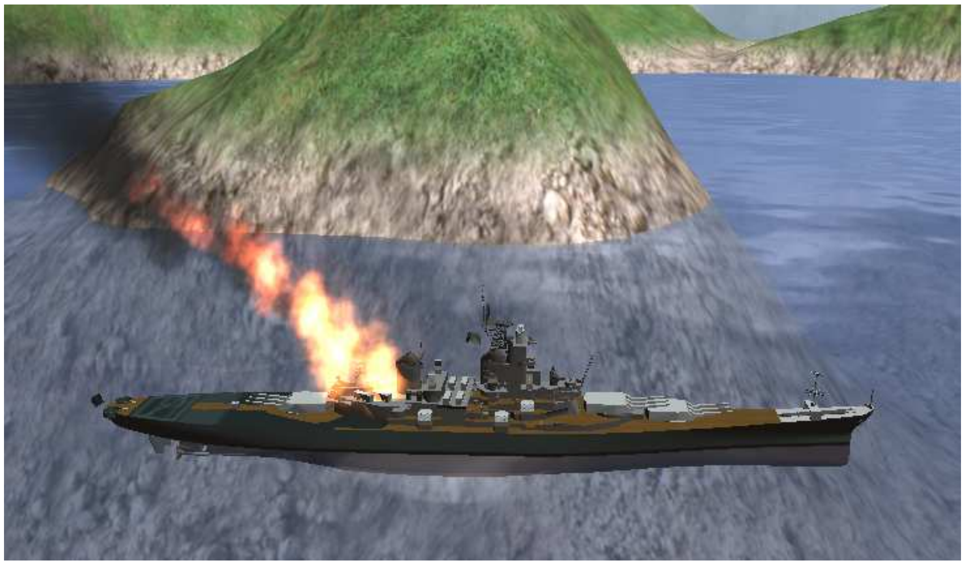

Figure 2: Battleship with animated flag, and fire & smoke particles.

2.2 Smoke and Fire

The henge particle systems were modified to offload billboarding to the GPU. The motion of the particles is still calculated in the CPU since otherwise a much more complicated render-to-texture simulation shader pass would be needed to perform integration and update particle positions. The billboarding particle shaders are: `part.v.glsl` and `part.p.glsl`.

2.3 Animated Flag

The topmost (white) battleship flag was assigned a vertex shader that waves the cloth a flag little along the y axis: `flag.v.glsl`. To do that, time was passed as a unifrom, and the local coordinate system vertex position was calculated through a sinusoidal function of time an offset along the x axis.

2.4 Sea Surface

The sea surface animation was performed per pixel through bump mapping. The "elevation" of the sea surface at each fragment is evaluated through a fractal sum of perlin noise at different octaves (fBm), which is then used to calculate the fragment normal using the partial derivatives of the heightfield and constructing tangent and binormal vectors. The fragment normal is then used to calculate the view reflection vector in order to access the enviroment cube map. (see shaders: `sea.v.glsl` and `sea.p.glsl`).

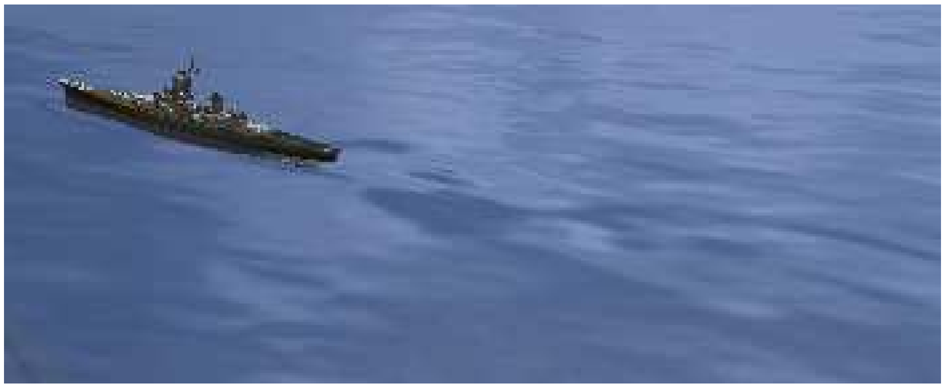

Figure 3: Ocean waves and ship wake.

2.5 Wakes due to Ship Motion

Since the motion of the battleships is a fixed counter-clockwise rotation around the center of the sea surface quad, a fixed "wake heightfield" was created in an image processing program. That wake texture is rotated appropriately by the sea fragment shader, and its contribution is added to the overall heightfield of the sea waves before calculating the fragment normal for the cubemap lookup.

2.6 Bumpy Island

A single mesh that makes up the little island in the middle, the mountains in the distance and the underwater seabed, is calculated during loading time by displacing the vertices of a tesselated quad using a low frequency, perlin noise based fBm. Also at loading time, a normal map is calculated for the that terrain mesh using the high-frequency part of the fBm. The terrain shaders (`land.v.glsl` and `land.p.glsl`) perform per-fragment illumination using the aforementioned normal map. Finally, two color textures are

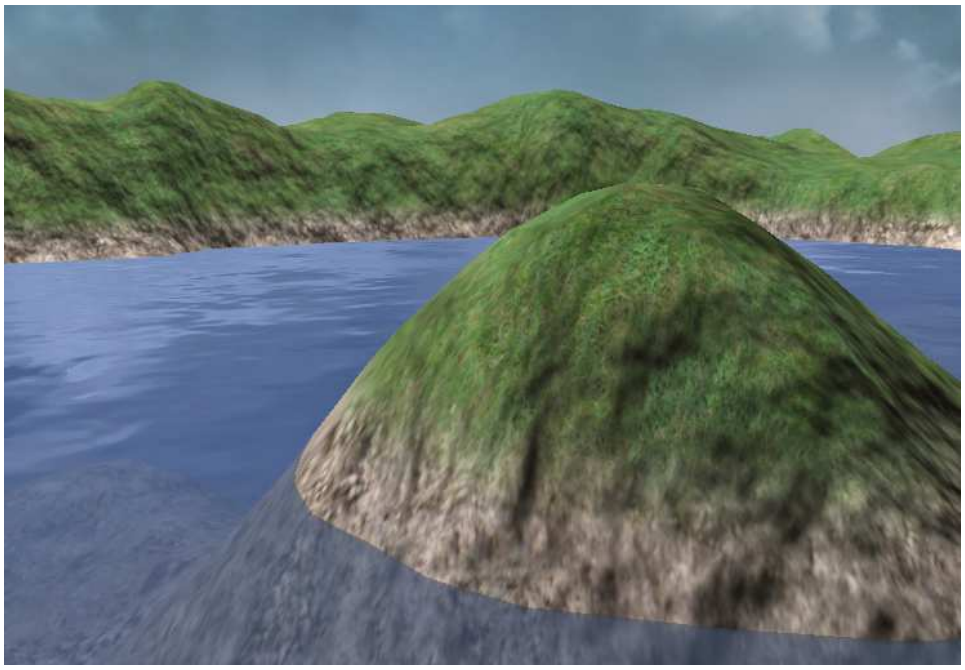

Figure 4: Normal-mapped island and mountains.

used, blended together as a function of height, to produce the effect of grass growing a few meters after the shoreline.

2.7 Underwater Fog

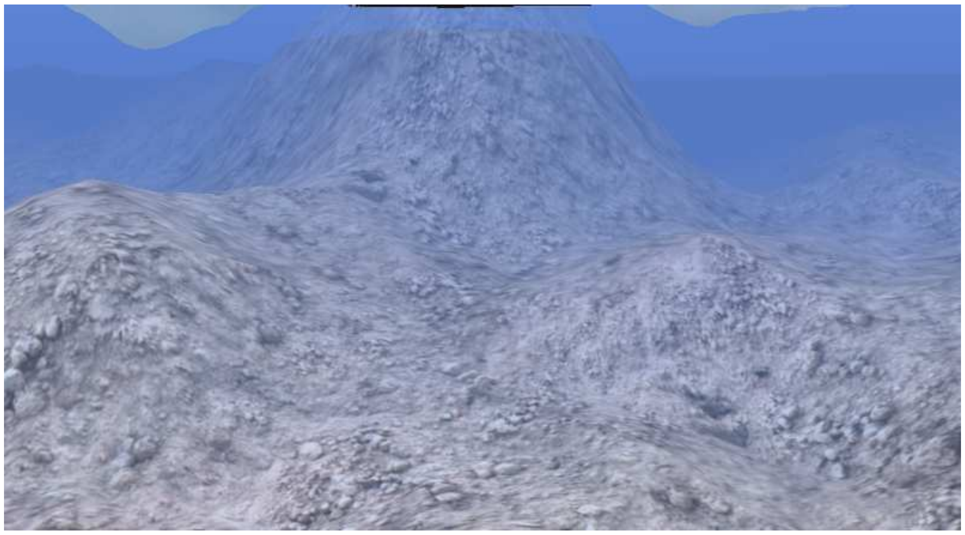

Figure 5: Underwater fog effect.

The same land shader mentioned above is also responsible for attenuating the color of the underwater parts of the terrain as a function of distance from the viewer, to produce an underwater fog effect (see figure 5).

2.8 Sky

The sky is a simple cubemap found on the internet, and mapped onto a large sphere that encloses the scene. The sky shaders (`sky.v.glsl` and `sky.p.glsl`) simply look up the cubemap color using the normal vectors of the sphere.

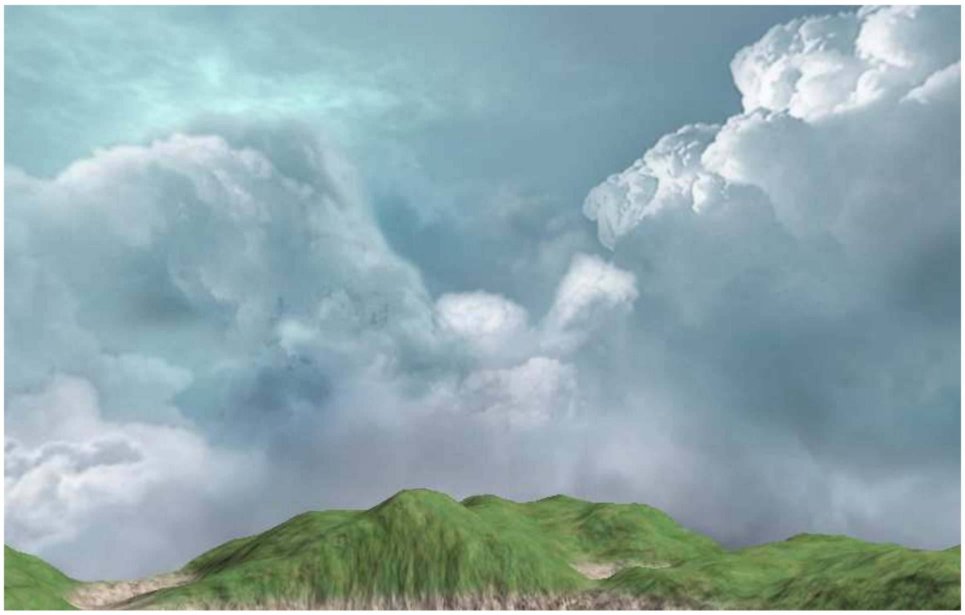

Figure 6: Sky cubemap.

2.9 Airplane

The shinny airplane is rendered with a per-fragment blinn shader that also uses the cubemap to make it look shinier (`plane.v.glsl` & `plane.p.glsl`). The afterburner jet of flame from the back of the plane is another instance of the GPU-billboarded particle system mentioned above.

2.10 Submarine

The sumbarine is rendered with the regular per-fragment blinn shader, modified to attenuate the color of the submarine as a function of distance, to match the fog of the underwater terrain shader. Also the same particle system was used to render bubbles at the back of the submarine.

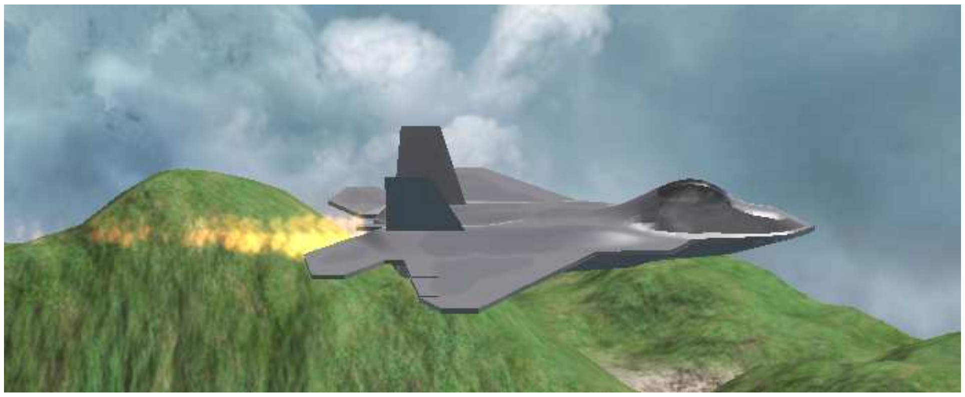

Figure 7: Airplane with afterburner particles.

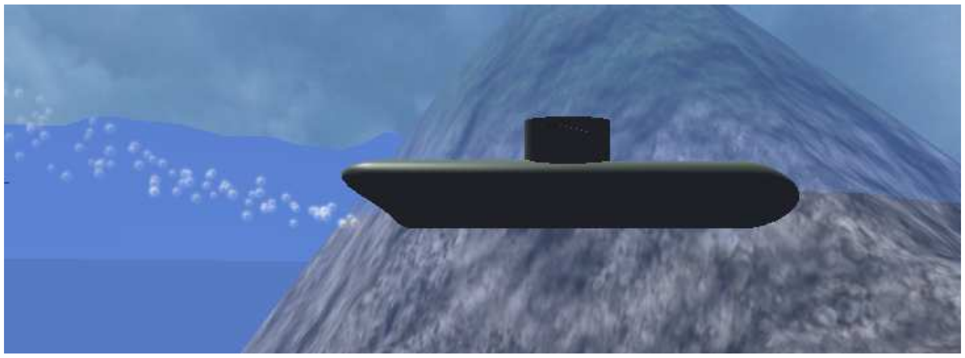

Figure 8: Submarine and its bubbles.