Computer Graphics OpenGL Coursework

James King

February 22, 2013

1 Application Overview

The application models a lone motor boat traversing an infinite and empty ocean, and simulates some simple water dynamics such as the wake left behind the boat as it moves. The boat may be controlled by the user, and a virtual camera may be oriented to look in any direction. It is also possible to toggle between a first-person view from within the boat to a third-person chase camera.

The implementation is written in the C♯ programming language on the .NET Framework v4.5, using the freely available OpenTK library to interface with OpenGL. I have had some prior experience with OpenGL, so my main intention was to explore some graphical techniques that I hadn't previously used. These features include parsing .obj files, water shaders, environment mapping, and frame buffer objects. Also, the implementation would aim to include no deprecated OpenGL calls from older version of the library, and would be entirely using the shader pipeline.

2 Assessment Criteria Catchment

2.1 Model Structures

2.1.1 Ship Model

The ship model was created using the free 3D modelling application Blender, and exported as a Wavefront OBJ (.obj) file which could then be imported into my application using a parser written by myself. The vertex normals have been smoothed by the modelling application so that they are the average normals of all faces connected to that vertex, to improve light shading effects.

The model is split into several groups, which I have called 'Face Groups' within the application. The purpose of the groups is to allow different parts of the ship to have different material effects; and in the case of the outboard motor and propeller, to have a different transformation applied before rendering. For example, the groups that the ship has been divided into are inner hull, outer hull, trim, motor, tiller and propeller. The inner and outer hull are the same mesh, but with the normals flipped and vertex winding inverted so that the two surfaces could be textured independently and have the correct normals for both sides. There is also one additional face group in the model; a plane covering the top of the ship. This 'depth clip plane' is used in a mechanism that stops water from being drawn inside the ship, which will be explained later. All the face group vertex data is then stored in a single vertex buffer object, with pointers to the start index and length of each group recorded so that they may be rendered individually.

2.1.2 Sea Model

Unlike the ship model, the sea model is generated procedurally when the application launches. My implementation for drawing the sea uses a vertex shader looking up a displacement map to determine the height of the sea at each point on the surface, so all the model needs to be is a flat grid of quads. Because the details of the sea are encoded in a series of textures, the sea model itself can remain almost stationary in eye-space, with the exception being that it must pitch when the camera looks up or down in order to remain on a fixed plane in world-space. While generating the mesh, I have implemented a level of detail (LOD) reduction effect for quads further from the camera, and also culled quads that would be outside the camera's field of view.

2.2 Program Implementation

2.2.1 Graphics Canvas and Light Settings

I have made an effort to cleanly abstract most low-level OpenGL calls in such a way as to minimise code duplication and improve readability. For example, I have written wrapper classes for frame buffer objects, vertex buffer objects, textures and shader programs. Light is implemented in the shader which draws the boat through a diffuse and specular model, using a global light direction contained within a structure holding information about the scene. The water shader also has a slight diffuse effect, but the main material effect is a reflection of the sky environment map. The sky is a simple environment mapped cube surrounding the camera in a fixed position in eye-space.

2.2.2 Drawing of the Ship Model

The ship is drawn with two different textures for different parts, and uses the global light direction for a diffuse and specular lighting effect which also varies in magnitude for different parts. The outboard motor and tiller pivot on the spot relative to the ship's position, and the propeller spins independently with a fixed position relative to the rudder. A previous problem was that water would be drawn inside the ship when the main water plane was rendered, which has been fixed by drawing an invisible plane across the top of the ship that still writes to the depth buffer, therefore hiding any water drawn behind it.

2.2.3 Drawing of the Sea Model

The sea is the most complex part of the application. Along with the static surface quad mesh, the sea uses three textures to encode wave height, wave velocity, and the amount of spray on the surface. The sea is modelled as a large grid of columns, with virtual springs between the top of each column. When some columns are depressed (for example by the ship traversing across them) they accelerate upwards to try and match the average height of their neighbours, and the neighbours will accelerate downwards also. This, when simulated every update, causes ripples in the surface of the water. The spray texture slowly dissipates any high concentrations of spray, and will increase the amount of spray at a point if the velocity texture has a very high magnitude. The simulation is implemented using a series of shaders outputting to frame buffer objects, which write to the three water textures. Originally this was all done on the CPU without frame buffer objects, but offsetting the work to the GPU has allowed far higher detailed water textures to be used.

When rendering, the height texture is used as a displacement map in the water vertex shader to offset the vertical position of each vertex in the mesh. The height texture is also used in the fragment shader to find the surface normal at each fragment, which is then used in the sky reflection effect. The spray is drawn when the spray texture value at each fragment exceeds a threshold, and is modulated by a small noise texture that is generated when the program starts.