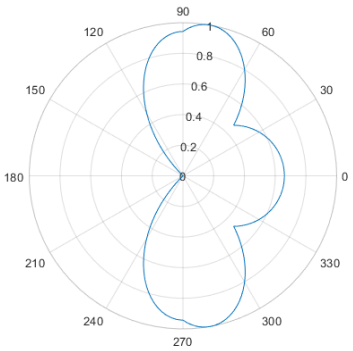
Ship Project

A tiny demo game made while learning Unreal Engine 4

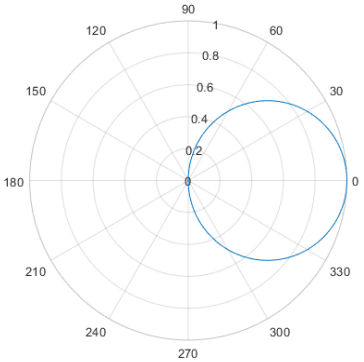
The idea of the game is simple: a minimalistic sailing simulator. In theory the game may have a lot of potential for development, but for the sake of this project a small number of core mechanics were selected to be implemented. The main focus was put on the buoyancy mechanic and the basic ship controls.

1. Ships Controls

The physics behind sailing are complex and involve a lot of different physical phenomena contributing to the resulting ship movement all at once, in this project a significantly simplified model is used for brevity. In the game the player controls two main parameters of the ship: the rudder angle and the sails orientation. In real world ships the sail rig's construction is adaptive and may react differently to different points of sail (i.e. the relative angle between the ship's direction of movement and the wind direction). A polar graph of possible sails effectiveness for different points of sail may look like this:



The angle on the graph corresponds to the point of sail and the graph itself plots out the normalized effectiveness of the sail. The effectiveness function is a sum of the contributions of different sail types, each of which is best suited to catch the wind for a specific point of sail. For brevity a simple model of the rig was used including only the spinnaker (the square sail best suited for running the wind):



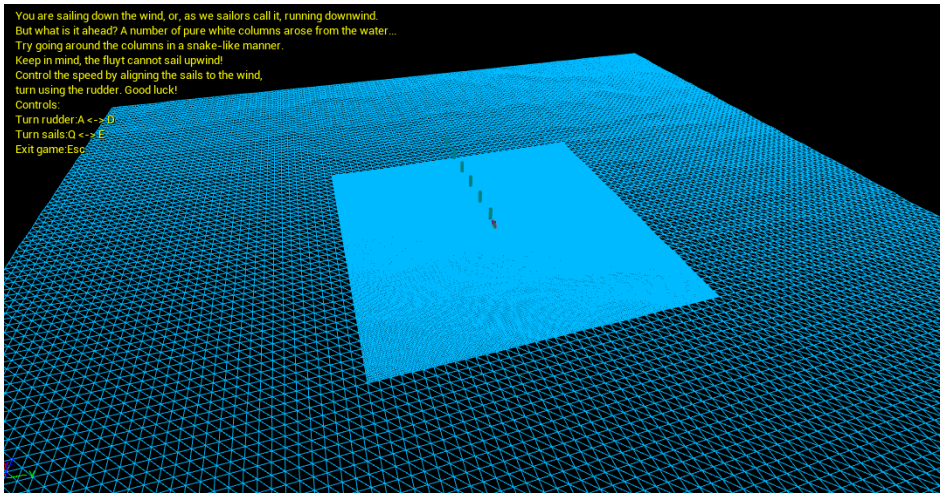
Thanks to the construction of the ship's hull, the force of the wind acting on the sail is only applied in the ship's forward direction allowing her to travel in directions not strictly aligned with the wind. The ship's orientation can be changed by changing the rudder angle. In real ships a complex system of hydrodynamical forces rotates the ship and the rotation of the rudder serves only as a trigger. In this project the entire system is simplified to a torque applied to the entire ship in proportion to the rudder angle and the ship's speed.

2. Sea and Buoyancy

The ship buoyancy mechanic can be implemented in a number of ways. The latest version of UE4 comes packaged with a built-in implementation and a number of other features for simulations of bodies of water. For educational purposes the sea and buoyancy mechanics were implemented from scratch.

The sea surface is generated on the fly using the native procedural mesh component available in Unreal Engine. The performance of the module is not optimal and there are other independent solutions such as the runtime mesh component plugin which perform better in general. The geometry of the sea surface is dictated by a height map, which is produced by running a Perlin noise texture through a sequence of sine functions and translation transforms. Unlike a random noise texture, this approach can output an endless pattern that is consistent in time (i.e. individual waves behave realistically).

The sea mesh consists of 9 tiles arranged in a 3-by-3 square. The middle piece has the highest detail while the rest are less detailed. The world position of the sea mesh is bound to the position of the ship controlled by the player, so that the ship is never able to reach the rim of the sea. This means that the sea may appear endless while using a constant number of polygons.



The ship's buoyancy is simulated via a number of forces applied at different locations of the ship's hull in proportion to their position above or below the water. If the control point on the hull is below the water line, a force acts on it pushing up in proportion to the depth of the dip.

3. Known Issues and Future Work

Currently the ship's buoyancy mechanic relies on the procedural mesh to generate hit events and uses them to determine the position of the ship's control points relative to the water. This can be bypassed if the wave height map function is used directly. Similarly the sea surface generation can be delegated to the GPU if the equivalent wave function is replicated in the material and is then used as the displacement guide for a simple tessellated plane. This has the potential of increasing the maximum detail level of the sea mesh without the computational overhead. This would also make the placement of other buoyant objects in the scene possible without them falling off the edge of the sea mesh once the player gets too far away.