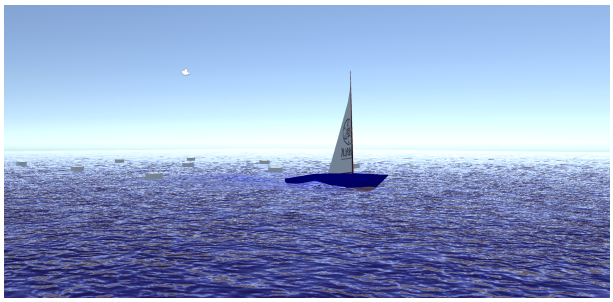
# Exploring the Unity3D rendering pipeline and normal extrusions on static meshes on the GPU, in the context of a sailboat

Ingemar Markström

December 5, 2017

# Contents





# 1 Introduction

## 1.1 Background

Understanding the rendering pipeline of a game engine is fundamental for accomplishing anything more interesting than colored triangles on screen at high frame rates in modern real time applications such as computer games. Although modern game engines like Unity3D, Unreal Engine och the Crysis Engine have made great efforts optimizing both their rendering performance along side physics engines, the need for understanding how the low level rendering is done can still make or break the total performance of an application. The first part of this project will study the rendering pipeline at a broader perspective, creating the static hull of the boat, while the second part of the project investigates two different applications of real time normal extrusion modifications on static meshes.

Realtime modification of meshes is a key part of creating bent surfaces in a 3D environment. Doing these calculations on the CPU quickly becomes slow when dealing with larger and higher resolution meshes, so offloading these modifications to the GPU is the next step in improving rendering speed. Considering a sailboat we have a sail curvature depending on wind speed, bending further the more wind there is, and an object moving through water will create waves propagating over the water surface. Both these phenomena is subject to mesh modifications and is studied in this project.

## 1.2 Purpose

The purpose of this project is to deepen my creative skills in shader programming by create a prefab of a sailboat for the game engine Unity3D with sails bending in the wind and waves propagating on the water surface away from the boat, efficiently modified on the GPU. First we investigate the rendering pipeline by creating the hull, efficiently making sure the water stays outside the boat. We then investigate the use of normal extrusion in the vertex shading step when rendering the sail curvature and as the last step then extend the demo with the effect of bow and stern waves.

## 1.3 Related work

### 1.3.1 Part 1: No water in the boat

In previous project work [6] at the Royal Institute of Technology, an augmented reality application on a phone gives the user the ability to see through

a paper box resembling a small scale model of a real sarcophagus. This is done by modifying the order of which the different parts of the 3D scene is rendered in the phone, and the goal of this project is to adapt this solution to empty the hull of a boat from water in the shading step, instead of cutting and slicing a big water plane into pieces.

### 1.3.2 Part 2: Normal extrusions

Working with normals and normal extrusion is nothing new in computer graphics [1], and one common usage today is for example extruding bricks out of textures with normal maps used in bump or parallax mapping. In this project however, the underlying structure for the normal map will be continuously changing depending on the surrounding environmental effects on the boat, in contrast to to the general idea of having a fixed normal map texture.

Although the general idea of modifying a flat mesh into a bumpy surface with a real time updated model instead of a static normal map is the common denominator here, the way the calculations for the amount of extrusion for the sail and bow waves are done will differ. The goal with both is however to be as lightweight as possible, but still look pleasing,

**Sail curvature:** The sails on a boat are sewn and designed to deform following the wind, and when achieving the optimal reeling, an airfoil is formed. This curvature can be seen as a second order bezier curve as described in [4]. Using many vertices in the sail, and interpolating the offset using UV position on the sail mesh texture, this Bezier curve can be calculated.

**Bow waves:** Following a similar approach with the waves, the problem here is to represent the underlying structure and waves mechanism in a realistic way. Interesting work has been done in the field of realtime wave particle simulation [9] of which this project part was heavily inspired from, and which seems to be a reasonable method for rendering real time waves. Handling waves as particles emmitted at the bow and stern of the boat in the direction of the wave at a regular time interval and then mapping this collection of particles down on a height matrix, gives the underlying structure sent to the GPU. This resembles using a static height map texture, but the matrix is updated each round of the physics engine.

## 1.4 Limitations

As this projects focus on normal extrusions of sails and water, no excess time will be spent on creating the most beatuiful boat or the most realistic water shader. Controls for sailing the boat along with a simple physics will be implemented, but actual playability or pure game development aspects are not prioritized. At the same time the calculations for realistic sail drag and efficiency is out of scope for this program, but a simplified mathematical sailing model will be implemented. We also limit the attempts to only generate waves on an initally flat ocean surface.

## 1.5 Method

All models used will be created using Blender 3D, with texture editing done in GIMP, and the complete asset and scenery will be assembled and created in Unity3D. The main coding language of Unity3D is C#, and although shaders can be written both specifically for OpenGL (GLSL) or DirectX (HLSL) the shaders created for this project will be written in the declarative ShaderLab syntax, which is the common way of writing Unity3D shaders. The work is to be continuously documented on the development blog [7], and this report covers the key aspects of the project work progress.

## 1.6 Perceptual study of sea and cybersickness

### 1.6.1 Background

As of work prior to this project in CoCar [2], by reading [3] when implementing the driving experience of CoCar we started investigating the amount of cybersickness that follows from driving a car at high speeds in virtual reality. Previous work has been done in the area of rotations in VR, and while the study in [5] concluded that all types of rotations are problematic but no one was worse than the other, the developement of CoCar pointed to rolling being much worse than pitch and yaw movement. This was solved by completely eliminating roll in the game, and making the horizontal line a fixed reference to the driver. An interesting perceptual study would be continued work and investigations regarding ways to give the user an enjoyable experience of sailing a boat in virtual reality with the random motions that occur in the boat. In the conclusion of [8] they state that fast and sharp turns might be a reason for the high rates of cybersickness, which is an interesting entry point for future investigation as sailboats are generally slow.

### 1.6.2 The study

When developing this project further, one idea is to implement controls for sailing in virtual reality. Because a boat is constantly swinging back and forth when moving, the perceptual study would be to complement our previous studies of CoCar (where rolling was investigated and later locked), by researching the effects on cybersickness of vertical motion when the boat goes through waves and if the speed of vertical motion is relevant. This would be done using three test groups:

1. The user is to manouver a small dinghy that react rapidly on the water around it, which create fast seemingly random up and down movement where the user is located

2. A larger, heavier boat, that reacts slower and more predictible in the water, much to the opposite of the small dinghey.

3. A control group, with a boat that is not affected by the surrounding water, and is extremely predictable to the user.

### 1.6.3 Test and Evaluation

The participants taking the test should be sailors, or at least have prior knowledge how to sail a boat, and they would fill out a questionaire prior to the test with information about their prior experience of sea sickness, and experience with virtual reality. Because of good research ethics, there should also be a checkbox that ensures the test person that because this is a test in a boat virtual reality, the effect might be severe nausea. The test should be to sail the boat as fast as possible around a small path of buoys for 15 minutes, and each 5 minutes the degree of nausea should be checked. The test would be repeated a fixed number of times, each with increasingly bad waves.

### 1.6.4 Expected outcome

The hypothesis would be that the more waves there is, the more ill the test person will feel afterwards because of the uncontrollable movement up and down. The small boat will be the worst, and the heavier boat will be better but not free from nausea. The control group should set the bias level of nausea for the basic non wave experience.

# 2 Part 1: No water in the boat

## 2.1 Creating the boat

The the boat was created in iterative steps, where the rudder, mast, boom, and inner and outer hull were separated, for reasons soon to be explained. The boat model were kept low poly because expected prototyping could possibly have the boat being scrapped or heavily reworked. The final version of the boat resembled the first prototypes regarding the separate parts, which gave reason to believe the first approach were on the right track. The first outline of the boat captured from blender can be seen in Figure 1.

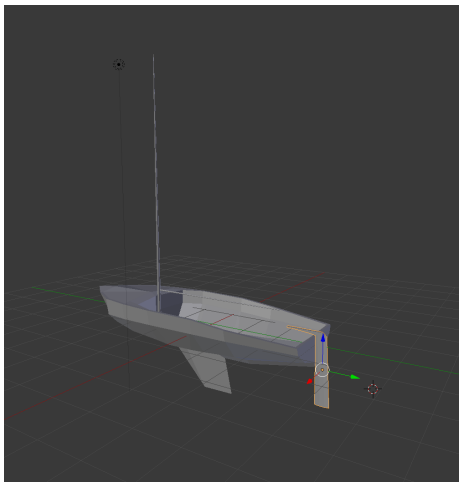


Figure 1: The boat

## 2.2 Drawing the ocean

The idea behind having the inner and outer hulls separated is a key in the solution for efficiently drawing the ocean. The problem we are dealing with is best illustrated in Figure 2. The approach taken here is to draw the water as one big blue quad, but as the boat is leveled to the waterline, we see water both inside and outside of the hull.

This is where the use of two separate hulls comes in. Taking the approach from Sarkofag [6], we can use a modified order of rendering and a custom inner hull shader (removal of the depth information check which is generally called the Z-buffer). This forces the renderer to draw the inner hull over whatever was already there, efficiently replacing the water inside the boat

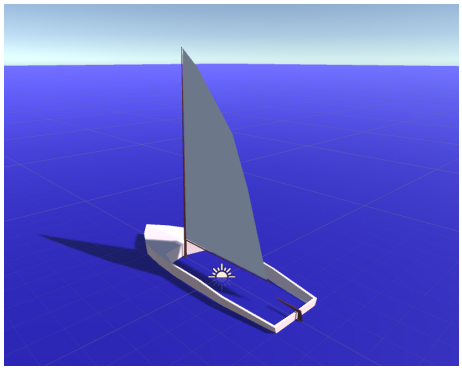


Figure 2: The problem with water

with the inner hull. In fact, this also does not hinder us having transparent water, but writing an actual water shader is far out of this project, as stated earlier in the introduction. With a simple transperent shader for the water plane, we see the results in Figure 3.

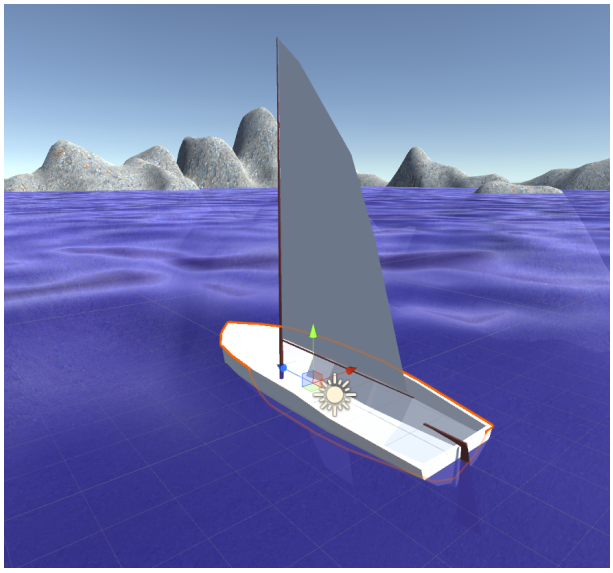


Figure 3: No water in the boat

### 2.3 Movement and floating physics

Although the boat was now rendered in water it still did not actually behave like a floating object, and as the built in physics engine in Unity3D is extremely powerful but overkill for this task, the boat physics does not resolve around the concept of Archimedes principle and gravity, but instead leveling out at sea level with a damped oscillation. This also made some movement swings back and forth and banking possible by applying forces through the regular physics engine, because it always tries to level out naturally. When the wind pushes into the sails, the forward force is determined by the angle between the sail and the wind direction, the sail area and wind speed. Realistic sailing physics is unfortunately out of scope for this project, but the model implemented gave a convincing apperence allowing further work of normal extrusions. Another pleasing addition of the implemented physics engine is the keel effect, which can be tuned for boat drift when making sharp turns. Although the physics implemented is minimal, it covers the basics of a moving boat floating in the sea.

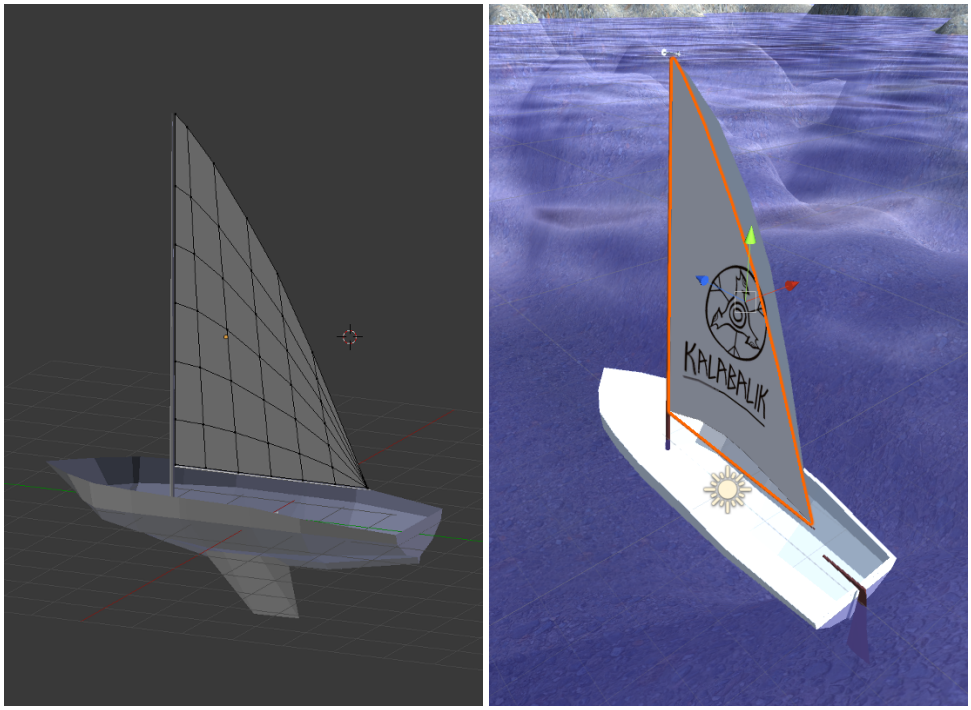
## 3 Part 2: Normal extrusions

### 3.1 Main sail

The sail is made out of the mesh that can be seen in Figure 4a, and as stated in [4] the sail can be approximated with a second order Bezier curve. Using the UV coordinates from the sail vertices, we can calculate the normal extrusion amount $E$ for each of them using the x position for $UV$ in second order Bezier formula:

$$E = (1-x)^3 + 3(1-x)^2(x) + 3(1-x)x^2 + x^3$$

When taking the wind force into consideration this resulted in the vertex pass in the custom sail shader seen in Figure 5 and the effect can be seen in Figure 4b.




(a) Sail mesh (b) Sail with a bend

Figure 4: Sail elements

```
...
void vert (inout appdata_full v){
  float vUV = v.texcoord.x/_SailLength;
  float uvOff = 3*(1-vUV)*(1-vUV)*vUV+3*(1-vUV)*vUV*vUV*0.2;
  v.vertex.xyz += v.normal*_SailForce * uvOff;
}
...
```

Figure 5: Excerpt of the vertex shader from the sail shader

## 3.2 Approaches for simulating waves

### 3.2.1 Height map textures or wave particle simulation

The next useage of normal extrusions was in creating waves around the bow and stern when the boat moves. Here, two main ideas were investigated. The

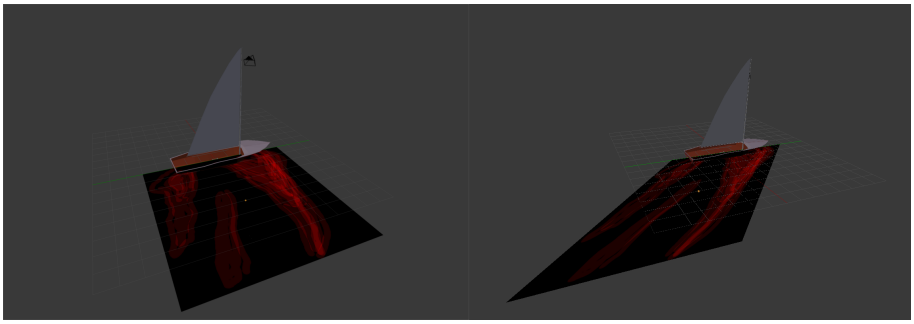


Figure 6: The height map approach to waves

first was based on using a height map texture and the second with an actual particle simulation on an underlying matrix model, but only the particle solution was implemented. Because the height map would be connected, skewed and rotated in relation to the boat, the situation where the boat steers into its own waves would rather look like it crumbs around itself, and the tip of the outer wave would be moving faster than the inner wave. On the other hand, one positive aspect of the height map would be the fast rendering time, essentially only drawing a high resolution quad with a custom bump mapping. But, even though this project was not nessecarily a simulations project, it could be argued that the end result would surely look more realistic using a wave particle approach, making it the implementation of choice.

### 3.2.2 Boat movement and underlying matrix implementation

Implementing wave particles was done on a 30 by 30 matrix with the boat centered in the middle. The upper limitation of matrix size was determined by the largest uniform that can be updated and sent to the shader each frame, which for the used system (MacBook Pro 2014 high performance specification) at implementation was 1023. Efficiently having a matrix of 900 height points gave room for additional uniforms if needed. The particles were kept in a fixed size array, and new particles are created by cycling forward in the array, eventually updating and exchanging old particles that is no longer in use. Because all particles are created at startup and no particles are destroyed, the time for rendering the wave particles to the matrix each frame is constant. All $n_p$ particles needs to be updated, then written to the wave matrix, which is a linear time operation $O(n_p)$. Creating new $n_{np}$ particles is also done in linear time $\mathrm{O}(\mathrm{n_{np}})$. The wave matrix is then

translated into an array of $m_x * m_y$ float values, which in our case is 900. Thus, the simulation is rather lightweight. As seen in Figure 7 waves of

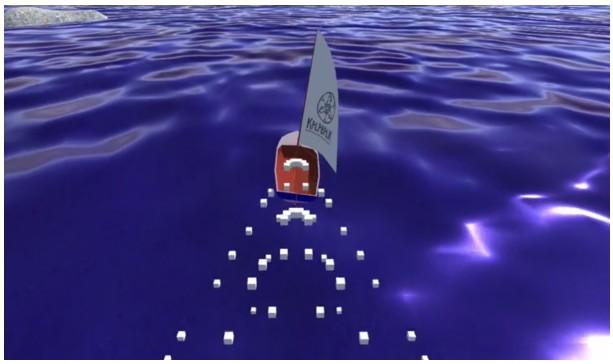
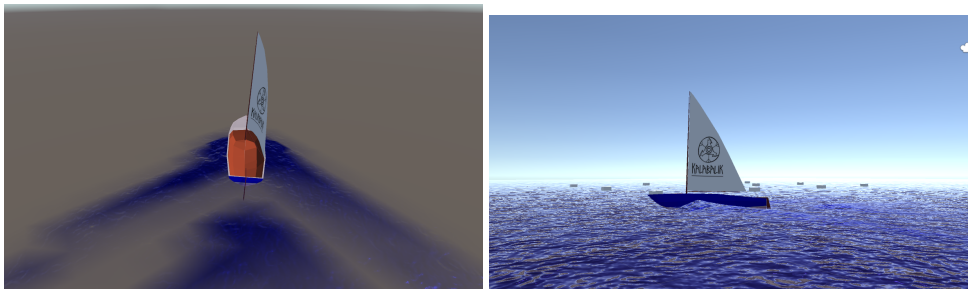


Figure 7: Wave particles as boxes

a fixed number of particles (here simulated as white boxes for clarity) are spawned with a fixed period. Each particle contains both information about the initial direction and speed when it was created, world position, but also the boat speed at which it was created which is seen as the size of each cube. For each rendered frame, we go through all the particles, and translate them onto the high resolution quad around the boat. As the size of each particle fades over time, the corresponding height of the wave shrinks, and the height is determined by summing all particles that end up on the same position in the matrix. This allows many wave particles to influence the same position which makes colliding waves create higer waves. This solves the questionable realism issues with the height map mentioned earlier in 3.2.1.

### 3.2.3 Integrating waves to the ocean plane

At this state we have the boat with a roughly two boat lengths of wave effects surrounding it as seen in Figure 8a. The next challenge was to have the visible effect of waves integrate with the low resolution ocean plane. This was done by first matching the color of the waves to be somewhat different from the ocean, and then implementing alpha blending depending on wave height. Higher waves gets higher alpha values, which in turn makes the color change more to the color of the waves. If the height is zero, only the ocean will be seen, but as soon as there are wave particles in motion, those will impact the perceived color of the water. The result can be seen in Figure 8b.



(a) The wave particles plane (b) Waves on the ocean

Figure 8: Waves

# 4 Results and future work

Using the simple concept of moving vertices along their normals, we now have a sailboat with a sail that takes on the shape of a airfoil with respect to the apparent wind like its real counterpart, and when the boat gains momentum, waves arise at the bow and stern, further adding to the illusion of speed and realistic boat behaviour. The physics engine additions implemented further increase the feel of mass by the adjustable skidding factor implemented as the keel factor.

One obvious flaw of the Bezier interpolation of the sail in itself is the lack of correlation between sail bulge and the actual length of the sail itself. If the effect were to be exaggerated, it would be obvious that the sail acts like a balloon and is made of rubber. But, as long as the effect is used in a realistic manner, it is not noticable. To compensate for this we also have the posibility to modify the length of the sail along the boom if needed.

Although the effect of the waves are limited to the relatively small waves following and surrounding the boat, future work could be done to find more efficient ways of sending the height data to the renderer, or to investigate tesselation to get more vertices in between the ones already present, and thus increased detail level and area. But, the aim of this project was to make an efficiently computed water waves effect and not a completely accurate water simualation.

Also missing in the boat at the moment is the character sailing it. The boat is now viewed from above and behind, but future work could be the implementation of virtual reality controls, for example using the HTC Vive, where you can walk around in a larger boat, control the reeling and steering, and research the potential factors of sea and cybersickness in VR.

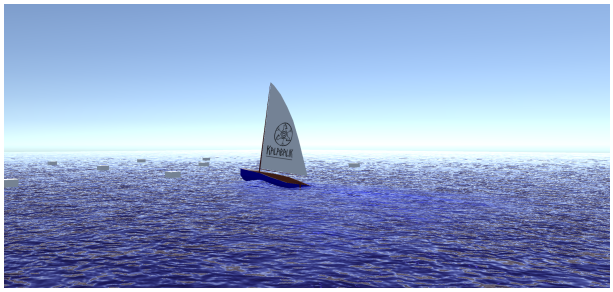


Figure 9: Final result

## References


[1] Blinn, J. F. Simulation of wrinkled surfaces. In *ACM SIGGRAPH computer graphics* (1978), vol. 12, ACM, pp. 286–292.

[2] CoCar. `https://www.cocar.se`, 2016. Project reference.

[3] Davis, S., Nesbitt, K., and Nalivaiko, E. Comparing the onset of cybersickness using the oculus rift and two virtual roller coasters. In *Proceedings of the 11th Australasian Conference on Interactive Entertainment (IE 2015)* (2015), vol. 27, p. 30.

[4] Leone, A., and Teresi, L. Numerical study of aeroelasticity of sails. In *COMSOL Users Conference* (2006).

[5] Lo, W., and So, R. H. Cybersickness in the presence of scene rotational movements along different axes. *Applied ergonomics 32*, 1 (2001), 1–14.

[6] Markström, I. Sarkofag webpage. `http://www.sarkofag.se`, 2016. Making of Sarkofag.

[7] Markström, I. imarkstrom development blog. `https://imarkstrom.blogspot.se/search/label/Sailboat`, 2017.

[8] Vinson, N. G., Lapointe, J.-F., Parush, A., and Roberts, S. Cybersickness induced by desktop virtual reality. In *Proceedings of Graphics Interface 2012* (Toronto, Ont., Canada, Canada, 2012), GI '12, Canadian Information Processing Society, pp. 69–75.

[9] Yuksel, C., House, D. H., and Keyser, J. Wave particles. In *ACM Transactions on Graphics (TOG)* (2007), vol. 26, ACM, p. 99.