TTK4550 Specialization Project

Tobias Bergkvist

December 17th, 2019

Contents

Introduction

The Heave Problem

When drilling from a floating rig or drilling ship, the heaving motion of the floater causes major pressure fluctuations in the well when the drill pipe is in slips during connections. Pressure fluctuations in the order of 10-20 bars have been observed in practice, sometimes giving an unacceptable risk of mud loss or kick. The only remedy for the problem is to wait for wind and waves to subside. There is a potential for saving time and cost by obtaining accurate information about downhole conditions on which to base the decision to wait or drill forward. HeaveLock has developed a simulator that predicts downhole pressure fluctuations based on weather information, response amplitude operator of the rig, well geometry, fluid properties etc. The simulator produces a large amount of data and the data needs to be visualized to the user in an easy way. In this project work, the objective is to visualize simulator data for a chosen well by developing a web-based interface.

The HeaveSIM output data

Picking the right tools

Selection criteria

Accessibility

How easy is it to get access to the system from any device?

Performance

Is the experience smooth (high FPS that matches the monitor), without consuming a lot of system resources?

Extensibility

How easy is it to add additional controls, features, or UI elements in the future?

Alternatives for 3D rendering

OpenGL with C++

- Performance: very good

A desktop application requires the user to install software or an app. (bad for accessibility)

Unity WebGL with WebAssembly

- Internet Explorer does not support WebAssembly (https://caniuse.com/#feat=wasm)
- Unity WebGL is not supported on mobile browsers (https://docs.unity3d.com/Manual/webgl-gettingstarted.html)

THREE.js (WebGL) with JavaScript and GLSL

In this case, performance could become a challenge, but having witnessed complex and performant applications created in this way - it must be obtainable.

Accessibility in this case is superior. (https://threejs.org/docs/#manual/en/introduction/Browser-support) This could work in Internet Explorer 11, Microsoft Edge, Google Chrome, Firefox, Opera, Safari, as well as modern mobile browsers.

This is likely the alternative which has the biggest challenges when it comes to performance - as JavaScript is an interpreted language. OpenGL Shading Language (GLSL) allows for writing code that is compiled on the gpu. In theory, having most of the work-intensive code written here could help deal with performance issues.

The ability to write code in a highly abstracted scripting language is great for extensibility, as the time investment of extending the system with UI components is low.

Implementation

Algorithms

The image idea

Final Architecture

Optimizations

Cross-Browser compatibility

User Guide

Further Work

References