A PRIME ON WEB-BASED SIMULATION

Ícaro A. Fonseca and Henrique M. Gaspar
Department of Ocean Operations and Civil Engineering
Norwegian University of Science and Technology
Larsgårdsvegen 2, 6009, Ålesund, Norway
E-mail: henrique.gaspar@ntnu.no

KEYWORDS
Web-based Simulation, Virtual Prototype, JavaScript Library for Engineering, Digital Twin.

ABSTRACT

A web-based approach gives advantages regarding sharing, compatibility, open source development and interactivity of engineering simulations. This work introduces the approach with a simple damped harmonic oscillator, presenting mathematical formulation, numeric solution method and visualization of the results both in graphical and in 3D. Vessel.js, a more advanced JavaScript library, is used to exemplify web-simulations of complex marine engineering problems, such as vessel motion with six degrees of freedom in calm water and vessel motion with mooring lines. A gallery of open source simulation for conceptual ship design and marine engineering is presented, with detailing of three other examples previously developed: motion response of multiple vessels in regular wave, simulation of pendulum motion of a lifted load and simulation of fuel consumption in transit. A next step of the work intends to focus on further development of time-domain models for maritime and subsea operations in general. The paper closes with a call for future digital twin applications to be developed in open web-based platforms.

WHY WEB-BASED SIMULATIONS?

Web-based simulations provide advantages in terms of sharing, compatibility, open source development and creation of graphic user interfaces, or GUIs (Gaspar, 2018).

The fact that it is possible to share web simulations with anyone who has access to an internet connection makes the approach convenient to give distributed users access to the same model. This centralization also allows unified support of the application, as once the developer deploys a new version of the source code online, all users instantly obtain access to it.

Such applications are compatible with any modern device with a web browser. While at this point most consumers take it for granted that a web service is going to work seamlessly on their devices, this advantage, in the context of engineering simulations, overcomes common compatibility issues related to operational system type or version, dependence on frameworks or on installation of programming languages.

A solid open culture supports web development and web technologies. In fact, a set of open standards, establishes the basis of the web's current ubiquity by allowing websites to use the same conventions without obstacles imposed by proprietary restrictions. There are multiple benefits in adopting this openness in the development of engineering simulation. From the point of view of scientific research, the openness assures that other researchers can verify and validate the simulation and its results. From the point of view of development, it facilitates the creation of simulations by providing access to libraries and frameworks for diverse purposes. Conversely, by publishing a simulation as an open web app, one allows others to reuse any element of the simulation, from 3D models to source code.

Compared to traditional engineering programming environments, web technologies provide more options and freedom for the creation of sophisticated user interfaces. The developer of a web application may use sliders, text fields and buttons to gather inputs from the user. Results can be presented as formatted text, tables, plots or interactive visualizations, either 2D or 3D. Multiple textual and graphic elements can be combined in dashboards to present a cohesive experience to the user, allowing them to vary inputs and observe the effects of the variation on the results in real-time.

Web development typically relies on three main languages: Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript. The first two define the content of web pages and the presentation style of the content, respectively. Focus is given on the rest of the paper on the later, and we propose that JavaScript language as key success to develop open and collaborative software in engineering since. As expressed by Gaspar (2017), the language excels in speed, compatibility among different platforms, user interface capabilities and large use by the community.

Given the aforementioned advantages of JavScript and the web-based approach, it is natural that developers pursue ways to bridge other programming languages to the web. At this time, there are already applications that interpret and execute source code from other

programming languages, for instance Python (https://jupyter.org/), or C (https://webassembly.org/) on web environments. However, for being a native programming language of the web environment, JavaScript allows a more complete realization of the approach, and thus is employed in the examples presented in this work.

FUNDAMENTALS

This section applies the principles presented in the introduction to a simple example of a dynamic problem. The problem is formulated mathematically, implemented as a web simulation and then visualized with different approaches.

Formulating the Problem

We formulate a simple problem of a block vibrating in one direction while attached to a spring and damper, i.e., a damped harmonic oscillator. The block is subject to an external force, as shown in Figure 1 below.

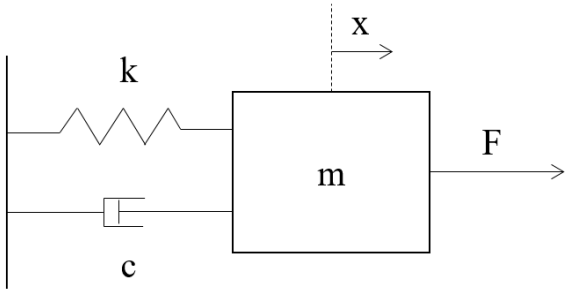

Figure 1: Scheme of Modelled Oscillator

Equation (1) below expresses the motion of the block over time.

$$F = m\ddot{x} + c\dot{x} + kx \quad (1)$$

This equation can be solved in JavaScript using the Runge-Kutta Dormand-Prince solver (RKDP), implemented in the Numeric.js library (Loisel, 2019). To do so, we need to provide the solver the initial time, final time, initial conditions (position and velocity) and a function containing the equation to be solved. To model the oscillator, we start defining the constant parameters:

```
var m = 30; // kg
var k = 20; // N/m
var c = 2.9; // (N*s)/m
```

Now the function representing the differential equation needs to be defined. To do that, Equation (1) is rearranged in a system with Equations (2) and (3).

$$d(x)/dt = \dot{x} \quad (2)$$
$$d(\dot{x})/dt = \ddot{x} = (F - c\dot{x} - kx)/m \quad (3)$$

The function will receive as input an array with the position and velocity, respectively x and $\dot{x}$ (the arguments of the derivatives in the left side of the equations) and output the evaluated derivatives (as in the right side). For a free oscillator, it will be:

```
function freeMotion(t, x_arr) { // x_arr = [x, xdot]
    var F = 0; // N
    var xdotdot = (F - c * x_arr[1] - k *
x_arr[0])/m;
    return [x_arr[1], xdotdot];
};
```

Finally, the following lines solve the equation with initial conditions x = 1 m and $\dot{x}$ = 0 m/s from time 0 to 100 seconds. The RKDP method uses adaptive time-step, so no standard time-step size is specified. If necessary, the default tolerance can be tuned.

```
var tmax = 100; // in seconds
var sol_free = numeric.dopri(0, tmax, [1, 0],
freeMotion);
```

This creates an object (`sol_free`) with the all the positions, velocities and accelerations calculated by the solver.

Graphical Visualization

Besides inspecting the simulation results in the programming console or printing them as formatted text, it is also possible to create visualizations of relevant simulation variables. The plots presented in this section use Flot, a JavaScript library that offers good compatibility with Numeric.js. Other useful visualization libraries are D3.js (https://d3js.org) and plotly.js (https://plot.ly/). Figure 2 shows a plot of the block position x in relation to the neutral position over time as calculated for the free oscillator.

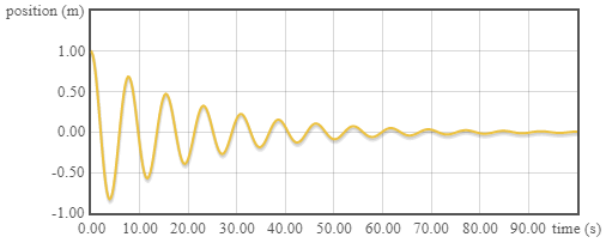

Figure 2 Block Position (X) for Free Oscillator

It is also possible to simulate and visualize the motion for a forced oscillator. Equation (4) defines the force as a sinusoidal.

$$F = 10\sin(t/5) \quad (4)$$

By making both the initial position and velocity equal to zero and solving the problem again with the new force, we obtain the plots for the force and position in Figures 3 and 4, respectively.

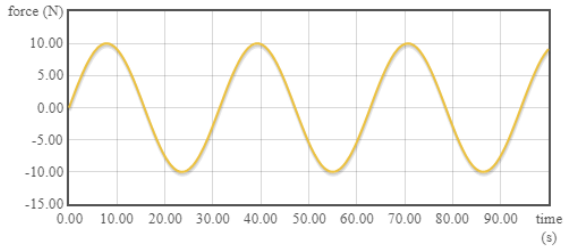

Figure 3 Oscillating Force

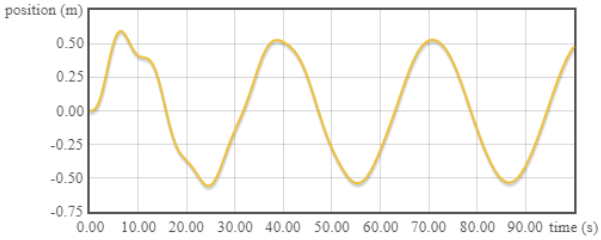

Figure 4 Block Position for Forced Oscillator

3D Visualization

The web-based approach supports complex 3D graphics with textures. The WebGL library allows rendering of 2D and 3D graphics in any compatible browser. In this work, the Three.js (https://threejs.org/) library is used to aid the creation and manipulation of 3D objects with WebGL, reducing the effort necessary to build a meaningful visualization.

After a 3D scene is created and configured in the browser with Three.js, a cuboid (or, as it is called in the library, a "block geometry") can be added to it with the following excerpt:

```
var geometry = new THREE.BoxGeometry(1, 1, 1);
var material = new THREE.MeshBasicMaterial({
    color: "blue"
});
var cube = new THREE.Mesh(geometry, material);
scene.add(cube);
```

Which creates a box in the scene, as shown in Figure 5 below.

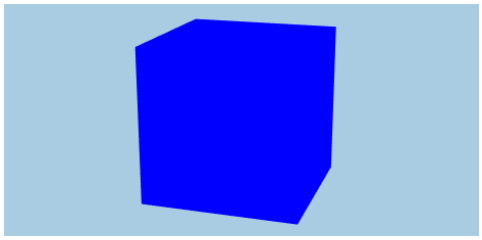

Figure 5 Block in a 3D Scene

Continuing to model the visualization, two cylinders are created to represent the spring and damper; and a thin cuboid is created to represent the base. The geometries are arranged in space with operations of rotation and translation to assemble the 3D model of the oscillator seen in Figure 6.

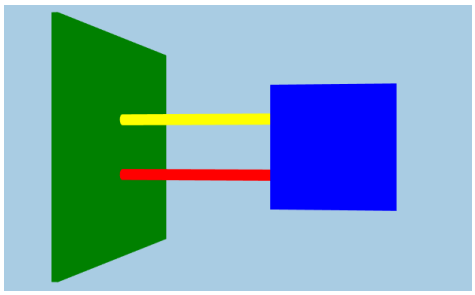

Figure 6: 3D Model of the Oscillator

Figures 2, 3, 4, 5 and 6 are available online at https://shiplab.github.io/vesseljs/examples/ECMS_2019.html

Besides aiding the creation of 3D models from scratch, the Three.js library also allows loading of 3D models stored as external files. It supports formats such as STL, a format of triangulated surfaces originally developed for 3D printing; and COLLADA, a file format for exchange of 3D models. This allows the user to create complex engineering models using dedicated CAD software and then convert them to a convenient format that can be directly loaded to a web scene.

Linking Formulas to the 3D Visualization

The 3D model can be linked to the mathematical formulation of the problem to visualize the oscillator's motion. This is done with some adaptations to the methods presented so far. The numeric solver is configured to calculate the shift in position of the block at the refresh rate of the animation, so it is possible to render the motion as it happens in real-time. At each refresh of the animation, the algorithm calculates the position of the block with the solver and updates the corresponding x coordinate in the 3D visualization. The algorithm also moves and scales the spring and damper cylinders to make them accompany the motion of the block. Figure 7 shows a sequence of screenshots of the final motion animation.

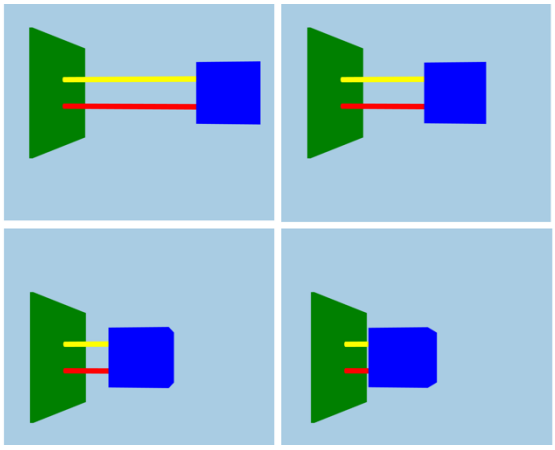

Figure 7 Different Positions of the harmonic oscillator for differents times.

SIMULATING ENGINEERING PROBLEMS

This section applies the web-based approach to the simulation of some problems in marine engineering. The simulations are developed with Vessel.js (hhtp:// http://vesseljs.org), an open and collaborative object-oriented library for conceptual ship design (Gaspar, 2018).

Multiple Blocks – The Case of a Ship

In Vessel.js, a ship is defined as a structure comprised of hull, decks and bulkheads, and a collection of "blocks" representing different tanks and compartments, as visualized in Figure 8.

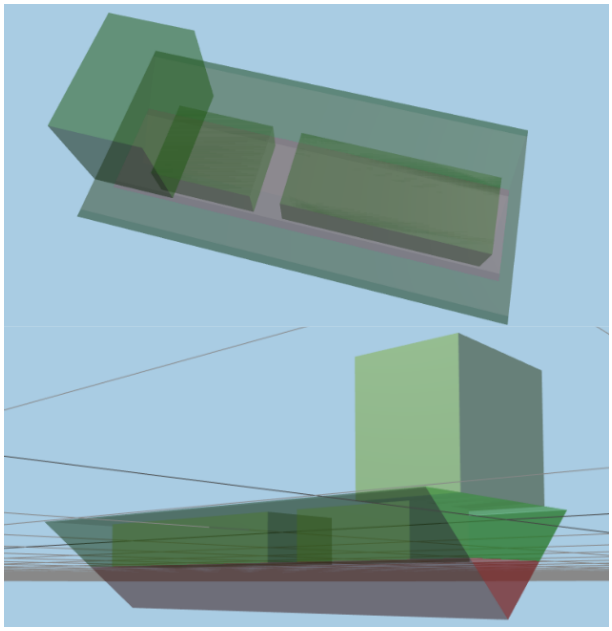

Figure 8: Visualization of a Simple Prismatic Barge with Three Compartments (Gaspar, 2018)

Each block has the same starting point as the mass from the previous harmonic oscillator example, with relevant attributes added according to the object that is modeled, such as weight, which is the sum of its own lightweight and the weight of its content. A "states" object collects the respective positions and filling ratios of every block inside the vessel. The user can modify these states to represent different weight configurations and/or different loading conditions of the vessel, for instance a fuel or cargo tank full or empty. The Vessel.js library is able to combine the weights of the individual blocks and of the structural components to calculate the total displacement of the vessel and its center of gravity numerically. These results can be used to further calculate the floating condition of the vessel, i.e. draft, small angles of trim, hydrostatic and stability coefficients.

The web-app in Figure 9 illustrates the calculation of such parameters. In it, the user can edit the prismatic barge by adding, moving or deleting blocks and modifying their weights. The application gathers user inputs in text fields arranged as tables where each row accounts for one block. The application display results as a summary list of weights and stability coefficients calculated with Vessel.js. Every time the user changes the inputs, the application recalculates the result, then updates the list and the draft of the 3D barge model in the visualization.

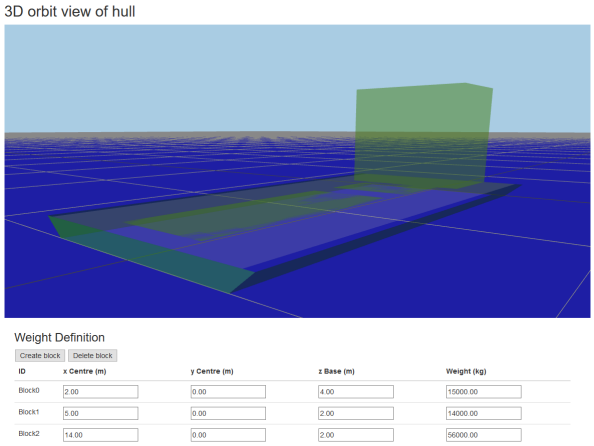

Figure 9: 3D Visualization and Input Pane of the Ship Stability Web-App

6 DOF Ship Motion

Just as the oscillator motion was solved and visualized with a web-application, the vessel motion is also simulated using the libraries presented so far. To accomplish that, an algorithm including a system of equations accounting for the six degrees of freedom of the vessel's motion was developed (Oliveira, 2019). For simplification of the mathematical model, a square barge is considered so the motion coefficients can be more easily established for small amplitudes, and only the condition in calm water is simulated, so no wave forces along the hull need to be considered (Figure 10). In that formulation, the vessel motion occurs only as response to the initial conditions set by the user with sliders in the GUI.

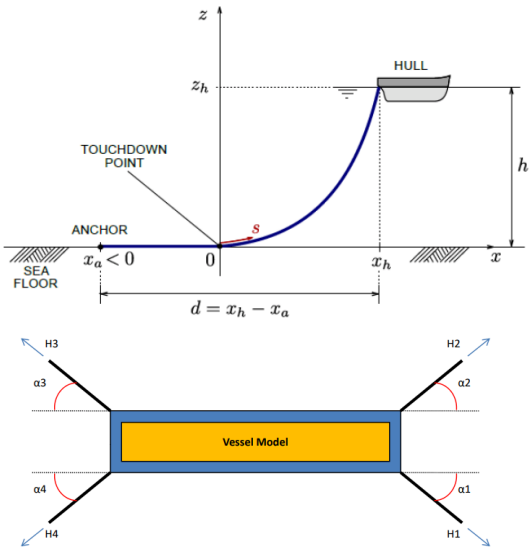

Figure 10: Simplified mathematical model for a moored square barge (Oliveira, 2019)

To make the visualization more realistic, a sky and ocean were added to the scene using a shader for Three.js. Figure 11 shows the 3D visualization of the barge in calm water, with no mooring. The sliding controllers on the top-right corner were created with the dat.GUI library.

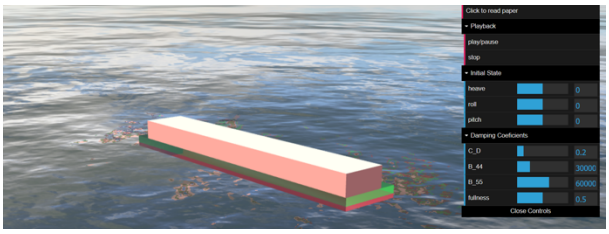

Figure 11: 6 DOF Simulator of Vessel Motion in Calm Water without Mooring

Despite the fact that the simulation accounts for the six degrees of freedom of the barge, the sliders only allow the user to set the motion modes which have restoring components, so the response oscillation can be observed. Those motion modes are heave, pitch and roll. They also allow configuration of viscous damping coefficients for different motion modes of the vessel, as it is difficult to estimate these coefficients before having access to experimental data. Furthermore, it is also possible to vary the filling ratio of the barge's compartment in order to observe how the cargo loading influences the motion response.

As soon as the user drags the slider, the algorithm starts calculating the barge motion response and oscillating it in the 3D visualization, as shown in Figure 12. Gradually, the damping dissipates the motion and the barge converges to a stationary position.

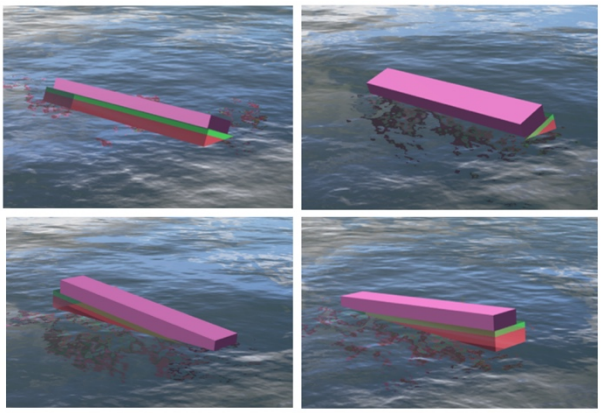

Figure 12: Barge Subject to Different Motion Modes in the Simulator

Mooring Lines

The vessel motion simulator can be used as a basis upon which to expand the simulation model to account for new effect, such as the barge moored to the seabed with four lines (Figure 10).

The mathematical model accounting for the effects of the mooring lines in the barge's motion receives as inputs the physical properties and geometric configuration of the problem, i.e., the length, hanging point and density of the lines, the radial distance from the anchoring points and the seabed depth. The simulator calculates the horizontal and vertical forces applied to the barge following an iterative procedure (Andrade *et al.,* 2016). The results are used to calculate the mooring forces and moments applied to the barge, which are included in the equations of motion solved during the simulation.

When modelling the 3D visualization of the simulation, the "line" geometry provided by Three.js allowed the creation of the mooring lines attached to the barge. Once created, the algorithm moved the geometries to follow the motion of the barge during the simulation. Figure 13 shows a screenshot of the 3D visualization in a stationary position.

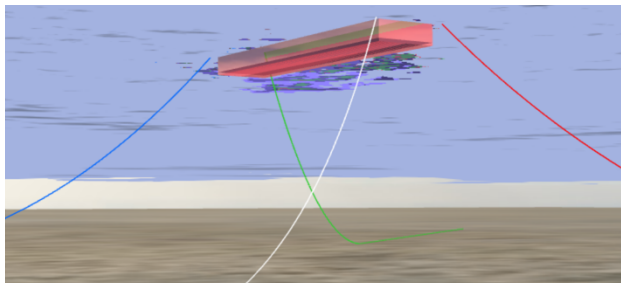

Figure 13: Lower Perspective of the Moored Barge Showing the Hull, Mooring Lines and Seabed

VESSEL.JS EXAMPLES GALLERY

The Vessel.js website presents a gallery with various open source examples of simulations focusing on topics related to ship design and marine engineering in general. This section presents three of them. As they are now, the applications are not intended to aid real life design decisions, but mostly to demonstrate the potential of the web-based approach to simulations and establish a base upon which the development of such simulations can continue.

Multiple Hulls in Regular Ocean

The application simulates the motion response for heave, pitch and roll of multiple vessels in an ocean created from a single regular wave, as shown by the screenshot in Figure 14. Differently from the previous motion simulations, which solved differential equations of motion, this app calculates the response amplitudes of the vessels with closed-form expressions (Jensen *et al.*, 2004). Then, it converts the response amplitude to a time series, synchronizing the motion of the 3D models of the vessels with the wave animation over time. As the motion is calculated with closed-form expressions, it does not take into account interaction among the motions of the vessels or their interference on the wave.

The application allows the user to control wave characteristics, i.e., amplitude, period, direction and phase with sliders. The wave length is automatically

derived from the wave period following the dispersion relation for deep waters. The user is also able to adjust the number of ships floating simultaneously in the simulation with a slider, and to add model of new ships by uploading a JSON file.

Figure 14: Simulation of Multiple Vessels in Regular Ocean

Vessel with A-frame in Regular Ocean

This application simulates a ship with a suspended load moving in a regular wave. The vessel motion response to the regular wave induces a pendulum motion to the suspended load, as shown in the screenshots in Figure 15.

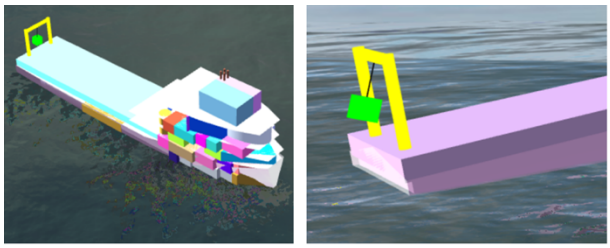

Figure 15: Simulation of Load Pendulum in Regular Ocean with the Load Pendulum in Detail

The vessel motion response is calculated with the same closed-form expressions as the previous example with multiple hulls. The pendulum motion is solved with the Lagrangian equations of motion for a spherical pendulum with a moving pivot (Myhre, 2019). The equations are solved in real time as the application renders the 3D visualization of the motion on the web browser. The user is able to reconfigure wave characteristics with the GUI sliders and visualize the motion response for different incident waves.

This simulation reconciles several types of vessel states: loading condition, floating condition, hull response to waves and pendulum motion. The Vessel.js library organizes these states in one object with two main groups: continuous and discrete, as shown in Figure 16. Discrete states are can be stored for longer spans of the simulation, while continuous states are updated in real-time and linked to the 3D visualization.

Fuel Consumption in Transit

This simulation uses semi-empirical expressions to perform estimations of fuel consumption in transit (Fonseca et al., 2018). It aims to provide metrics for the evaluation of the suitability of different vessel designs for a given operating region based on their respective consumptions.

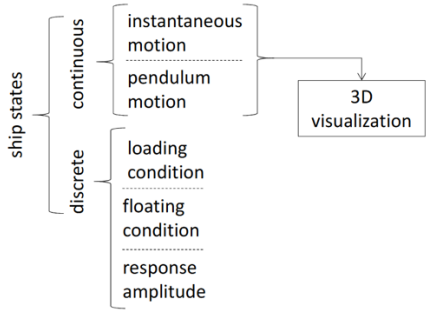

Figure 16 Organization of the States in the Pendulum Simulation

The application generates the different vessel designs from a combination of parent ship specification, list of power plant specifications and list of propeller specifications. A scatter diagram listing the annual wave occurrences describes the intended operating region of the vessel.

When performing the simulation, the algorithm creates new ship specifications by linearly scaling the main dimensions of the original parent ship specification, as illustrated in Figure 17. The scaling is constrained to keep the gross volume constant. Then, the algorithm combines the scaled specification with a power plant specification and a propeller specification to establish a complete design for the simulation. The algorithm simulates that final design in a trip with all the sea states listed in the scatter diagram, deriving an estimation of fuel consumption for the given combination. The process is repeated for all combinations of scaled ship, propeller and power plant specifications, allowing a comparison among the performances of different design options.

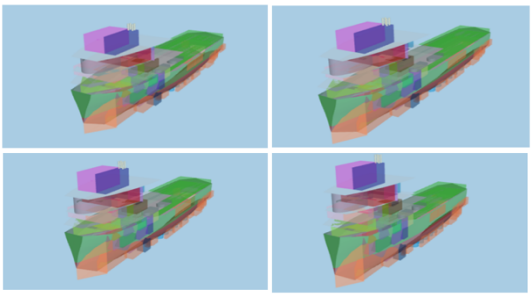

Figure 17: Four Examples of Scaled Vessels Evaluated in the Fuel Consumption Simulation

.

BENEFITS AND A CALL FOR WEB-BASED DIGITAL TWIN APPLICATIONS

As demonstrated through this paper, the web-based approach is already a viable solution for the development of several engineering simulations and presents advantages in terms of speed, compatibility, development tools and user interface. Such advantages reflect on the

characteristics of the various examples presented in the previous sections here summarized:

- Platform-independent and can be used without prior set-up or installation, working on modern computers, tablet or mobile.
- Extensive use of available open libraries, e.g., for solution of mathematical models, creation of 2D plots and rendering of 3D scenes.
- Interactive GUIs, allowing the simulations to convey meaning easily to users, including those who do not have an engineering background.

Speed is a high-point: a regular consumer laptop, or even a smartphone, is capable of executing the applications in real-time, solving the mathematical models and rendering the 3D scenes. Given the scope of the examples, this includes simultaneous solution of differential equations, manipulation of 3D geometries and rendering of textures in the web browser. For such reasons, it becomes apparent that the approach still provides unexplored potential for further work.

In this context, we conclude this paper with a call for more web-based approach to simulations, given that it has the potential to become an important enabler for the implementation of digital twins. That is, the same advantages regarding, e.g., compatibility and usage of web simulations during the design stage of engineering products can be extended to the operational stage. In that context, the simulations can be used to collect data about behavior of the physical product (feedback) and control it during operation (feedforward). The aggregation of asset behavior in the digital twin also opens the potential to reuse the operational data to aid design decisions of later products, thus "closing the loop" of data use in the product's life cycle.

Future work will focus on the case of virtual prototyping more complex maritime operations, exploring the application of web methods and standards to the interactions between physical systems and web simulations. As a starting point, a proof of concept will be established where a web interface tracks the motion of a scaled ship model and controls the operation of one its components, e.g., an engine. Such a concept will allow study of the role of web standards on the collection of data from physical systems (feedback), on the control of systems (feedforward) and on the linking of the collected data to the simulations used during the ship life cycle (the final "closing the loop" aspect). Once that role is established, it will be possible to move the research towards advanced applications of the digital twin.

ACKNOWLEDGEMENTS

The simulations presented in this work are available on https://vesseljs.org/. This research is connected to the Ship Design and Operation Lab at NTNU in Ålesund (https://www.ntnu.edu/ihb/ship-lab). The research is partly supported by the EDIS project, in cooperation with Ulstein International AS (Norway) and the Research Council of Norway, and by the INTPART Subsea project in cooperation with the University of São Paulo (USP) and the Research Council of Norway.

REFERENCES

Andrade, B., H. Brinati, O. Augusto and M. Conti. 2016. "Mooring Lines." In *Applied Topics in Marine Hydrodynamics*, G. Assi, H. Britani, M. Conti and M. Szajnbok (Eds.). Escola Politécnica da Universidade de São Paulo.

Fonseca, Í. A., Gaspar, H. M., Ryan, C. F. and Thomas, G. A. 2018. "An Open and Collaborative Object-Oriented Taxonomy for Simulation of Marine Operations". In *Proceedings of the 17th Conference on Computer and IT Applications in the Maritime Industries* (Pavone, IT). 412-425.

Gaspar, H. M., 2017. "JavaScript Applied to Maritime Design and Engineering." In *Proceedings of the 16th Conference on Computer and IT Applications in the Maritime Industries* (Cardiff, UK). 253-269.

Gaspar, H. M., 2018. "Vessel.js: An Open and Collaborative Ship Design Object-Oriented Library" In *Proceedings of the 13th International Marine Design Conference* (Helsinki, Finland, Jun 10-14).

Jensen, J. J., A. E. Mansour and A. S. Olsen. 2004. "Estimation of Ship Motions Using Closed-Form Expressions." *Ocean Engineering*, Vol. 31, 61-85.

Loisel, S. 2019. "Numeric.js." Available athttps://github.com/sloisel/numeric.

Myhre, T. A. 2019. "Spherical Pendulum Dynamics." Available athttps://www.torsteinmyhre.name/snippets/spherical_pendulum.html.

Oliveira, F. F. 2019. "Implementation of Open Source Code for 6 Degrees of Freedom Simulations in Maritime Applications." Research Report, Ship Design and Operation Lab, NTNU.

AUTHOR BIOGRAPHIES

ÍCARO A. FONSECA is a PhD candidate in engineering at NTNU in Ålesund, researching standards for ship digital twins. MSc in Ship Design at the same university with master thesis developed in collaboration with the Marine Research Group at UCL. Engineering degree in Naval Architecture and Marine Engineering at Federal University of Pernambuco (UFPE), Brazil. Brief industrial experience as a naval architecture intern in the Vard group.

HENRIQUE M. GASPAR is an Associate Professor at the Department of Ocean Operations and Civil Engineering, Norwegian University of Science and Technology (NTNU). The professorship is connected to the Ship Design Chair at the Maritime Knowledge Hub, sponsored by Ulstein Group. Education consists of a PhD degree in Marine Engineering at the NTNU, with research collaboration at UCL (UK) and MIT (USA). Previous professional experience as Senior Consultant at Det Norske Veritas (Norway) and in Oil & Gas in Brazil.