The application of computational fluid dynamics to the modelling and design of high speed boats

Jack Townsend

Swansea University
Prifysgol Abertawe

College of Engineering

Swansea University

2020

Submitted to Swansea University in fulfilment of the requirements for the Degree of Doctor of Engineering.

Abstract

Computational fluid dynamics solvers were applied to the field of high-speed boat design. The lattice Boltzmann method was used to assess the water-phase of the flow around a number of high-speed hullform geometries, and was validated against empirical industry and literature data. A heave dynamics capability was developed to assess the heave equilibrium position of a high speed boat, showing close agreement with industry data. A mesh movement and evolutionary optimisation software was applied to the aerodynamic optimisation of a high-speed catamaran using a Reynolds-averaged Navier-Stokes solver for modelling of the air phase of the flow.

Dedication

I dedicate this thesis to my granddad James Sharp, my best and earliest inspiration for engineering (even if I didn't know it).

Declarations

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

11/05/2021

Signature Date

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

11/05/2021

Signature Date

I hereby give my consent for my work, if relevant and accepted, to be available for photocopying and for inter-library loans after expiry of a bar on access approved by the University.

11/05/2021

Signature Date

Acknowledgements

I would like to acknowledge my supervisor Dr Ben Evans for the endless supply of opportunities, insight, knowledge, and support.

I would also like to acknowledge Dr Ian Mabbett for nautical advice and contacts, Dr David Naumann for help with the use of his AerOpt programme, Swetha Lakshmy for assistance in data gathering for the VWT catamaran simulations, Mark Watkins and Mike Smith for experimental assistance, Dr Mark Dawson and the wider SA2C community for programming advice, and finally all my friends and family in and out of Swansea who have generally put up with me for 4 years.

I would like to acknowledge the support of the Supercomputing Wales project, which is part-funded by the European Regional Development Fund via the Welsh Government, for providing the necessary computational resources and advice to complete this work.

Contents

List of Figures

List of Tables

Nomenclature

β	Deadrise angle, $^\circ$
γ	Ratio of specific heats
λ	Wetted length to beam ratio
λ'	Wetted length to beam ratio neglecting local wave rise
μ	Dynamic viscosity, $Pa - s$
ν	Kinematic viscosity, m^2/s
ϕ	Design parameter
ρ	Density of fluid, kg/m^3
σ	Surface tension coefficient
τ	Trim angle, $^\circ$
$\mathbf{e}$	Particle velocity vector in LBM context, $\mathbf{e} = (e_x, e_y, e_z)$
$\mathbf{u}$	Macroscopic flow velocity in LBM context, m/s^2
$\mathbf{x}$	Particle position vector in LBM context, $\mathbf{x} = (x, y, z)$
b	Beam (width) of boat
C_Δ	Load coefficient, $C_\Delta = \frac{v}{\sqrt{\rho b^3}}$
C_D	3-dimensional drag coefficient, $C_D = \frac{D}{\frac{1}{2}\rho S v^2}$
C_d	2-dimensional drag coefficient, $C_d = \frac{D}{\frac{1}{2}\rho l v^2}$
C_f	Schoenherr friction coefficient

C_L	3-dimensional lift coefficient, $C_L = \frac{L}{\frac{1}{2}\rho S v^2}$
C_l	2-dimensional lift coefficient, $C_l = \frac{L}{\frac{1}{2}\rho l v^2}$
C_v	Speed coefficient, equivalent to Froude number
D	Drag or resistance force
d	Draft (depth) of boat
f	LBM velocity distribution function
f_{CN}	Degrees of freedom per control node
Fr	Froude number, $Fr = \frac{v}{\sqrt{gL}}$
g	Acceleration due to gravity, m/s^2
L	3-dimensional reference length, m
L	Lift or heave force
l	2-dimensional reference length, m
L_c	Wetted length of the chine
L_k	Wetted length of the keel
m	Mass, kg
Ma	Mach number
N_{CN}	Number of control nodes
p	Pressure, $Pa(kg/m^2)$
R	Universal gas constant, $8.314J/(Kmol)$
Re	Reynolds number, $Re = \frac{\rho v L}{\mu} = \frac{vL}{\nu}$
S	Reference area, m^2
T	Temperature, K
t	Time, s

u_{LB} Lattice velocity in Palabos context

u_{ref} Reference velocity in Palabos context

V Volume, m^2

v Velocity, m/s

Chapter 1

Introduction

1.1 Thesis motivation and structure

This work was motivated initially by industrial interest in developing tools to predict high speed planing hull behaviour with the long-term goal of a water speed record. The industry goals morphed throughout this project, changing with the involvement of a secondary industry partner, Norson Design, to a more general application of CFD to hullform design. Along with this second partnership came interest in the topic of aerodynamic optimisation of a high speed catamaran.

Throughout these directional shifts, the underlying goal remained: the development of a modelling capability for high-speed boats in a manner that would be useful to an industrial entity. This took the form of the novel application of the Lattice Boltzmann Method (LBM) and High Performance Computing (HPC) to the problem of planing hulls to overcome difficulties associated with modelling high-speed hulls that were noted in the literature.

The scope of the physics experienced by the hullforms studied is limited to calm water conditions as a result of the water speed record origins, in which operation would only be conducted in such conditions. Largely because of the calm water environment, interaction between air and water is assumed to be negligible and the modelling complexity is reduced to a single-phase solution (water only) with free surface modelling to capture the fluid interface between air and water. The action of air on the fluid interface is captured by a constant

atmospheric pressure term in the free surface modelling.

As Norson Design joined the project, interest in modelling the water phase continued, and an additional work package on the modelling of the aerodynamic performance of a novel catamaran design was undertaken. This work focused on the aerodynamic modelling and optimisation of the unwetted portion of a catamaran design. Novelty in this work lies in leveraging the aerodynamic lift created in the "tunnel" area bounded by the catamaran hulls, water surface, and main cabin. By modifying the cabin geometry, aerodynamic lift could be increased, in turn reducing the water-wetted area and thus hydrodynamic drag; this work is presented in a stand-alone chapter. The structure of the thesis is briefly outlined as follows:

- Chapter One – planing hulls are defined and their physics described. The process of selecting an approach to modelling them is described..
- Chapter Two – the solver and methods selected are described in detail. The solver is validated against analytical and empirical data.
- Chapter Three – the addition of dynamics capabilities to assess the heave equilibrium of a hull are described and results are presented. This capability is validated against industry data.
- Chapter Four – a supplemental project on the aerodynamic optimisation of the non-water-wetted portion of a racing catamaran is presented as a stand-alone chapter.
- Chapter Five – an overall summary and conclusion to the thesis findings is given.

1.2 The historical context for computational ship design

The design and use of ships has a long history before the advent of the computational methods used in their design today. Prior to the 18$^{\text{th}}$ century, shipbuilding was more of a craft than a science: designs were arrived at along axioms of the institutional experience of shipwrights and their apprentices rather than scientific principles and data [1]. Such a design philosophy was not conducive to radical innovation as new ships were typically built immediately at full scale, making any changes outside of the traditional configurations very risky.

In 1857 – prompted by concerns over the *Great Eastern*, by far the largest ship built to date – William Froude was tasked with investigating ship stability in rolling [2]. Froude presented a landmark paper on the stability of ships at sea to the Institution of Naval Architects in 1861 [3] and was subsequently tasked with finding the most efficient hull shape. To do this, Froude empirically tested a series of scale models and postulated the dimensionless Froude number to relate scale test results to their full scale counterparts. Two models, *Raven* and *Swan*, were assessed (see Figure 1.1). Although the sharper bow of *Raven* conformed to John Scott Russell's waveline theory [4], the blunt bow of *Swan* produced less resistance [2]. This finding was successfully validated by full scale tests carried out by the Admiralty, leading to the construction of the first publicly financed ship testing tank [2]. The mathematical principles and empirical procedures pioneered by Froude marked a new age in the understanding of ship physics, and form the basis of the design of modern ships to this day.

At the turn of the 20$^{\text{th}}$ century, the development of smaller and more powerful motors brought about the advent of planing vessels. A planing vessel is primarily lifted – when at speed – by a hydrodynamic reaction force as opposed to a traditional displacement hull, which is held up primarily by hydrostatic force (i.e.: buoyancy). A full description of the physics of a planing hull is given in Section 1.3. By planing, vessels could reach far greater speeds

Figure 1.1: One of four original *Raven* (top) and *Swan* (bottom) hulls as used by Froude in scale testing on the River Dart in 1867 [5].

Figure 1.2: (Left) CMB-4 on display at the Imperial War Museum, Duxford. The 40ft (12.2m) wooden vessel armed with a single torpedo and pair of Lewis machine-guns sank a 7,087 ton cruiser, making it an early example of the military value of small, fast attack craft [7]. (Right) A similar CMB planing at speed during World War I [8].

than their displacement hull contemporaries, making them highly valuable military assets in the form of high speed torpedo boats that posed a danger to larger and more expensive battleships and cruisers. For example the Royal Navy's World War I era Coastal Motor Boats (CMBs) were most distinguished in 1919 when a team of CMBs (displacement of 5 tonnes each, see Figure 1.2) sank Russian cruisers *Oleg* and *Pamiat Azova* (displacement 7,087 and 6,674 tons respectively), a significant achievement for small, fast, cheap attack craft [6]. In World War II, German *schnellboote* used similar designs and tactics to harass Allied convoys in shallow waters around Britain, turning British east coast shipping routes into a perilous "E-boat alley" [6]. Fast attack craft have since taken a permanent role in the composition of navies around the world.

High-speed planing boats now make up nearly the totality of small to medium recreational boats and a large portion of military patrol boats [9]. As such, a great deal of modern ship design, including scale tow tank testing and analytical techniques, are targeted at planing vessels.

Tow tank and basin tests for the design of hulls, planing or otherwise, are typically labour and capital intensive, expensive, contribute to long lead-times, and limit the scope for modification to the hullform [10]. Modern maritime design is one of the many fields where computational techniques have revolutionised the design process. Computational methods now play some role – to varying degrees – in many ship designs. Few to no hullform designs are solely evaluated using computational models, as is often the case with CFD in general; sufficient uncertainty in the many complex input factors to a given CFD simulation mean that for critical designs CFD is backed up by experimental testing where possible. The advantage of CFD lies in the ability to interrogate the flow field in ways experimental testing cannot, high degree of repeatability, and (depending on the given case) cheaper experimentation due to less expensive equipment and/or facility requirements. Thus the ability to guide the design to a near-optimal starting point with CFD that can later be confirmed and refined with experimental testing has the scope to greatly reduce the cost and lead-time associated with physical tests [11]. Due to the hullform design often being prerequisite to further ship design work, reduction in the lead-time of hullform design is desirable from an economic perspective. Despite the wide variety of computational techniques available to the modern maritime industry, it has been suggested that the take-up of such techniques has not reached its full potential [12, 13]. Reasons for this, and discussion on the desirable features of future marine CFD capabilities are, is explored in Section 1.4.2.

1.3 Theory of planing hulls

1.3.1 Modes of travel for surface vessels

Hullforms can be broadly grouped into displacement hulls and planing hulls. Displacement hulls operate by displacing a mass of water equivalent to their weight, thus providing hydrostatic lift to keep them afloat. Displacement vessels are typically larger, slower or both: for example, container vessels and cruise ships, or a simple rowboat.

Resistance on a displacement hull underway is composed of pressure drag and viscous drag. The viscous component, D_v, is given by Equation 1.1.

$$D_v = \frac{1}{2}\rho v^2 S C_v \tag{1.1}$$

where C_v is the coefficient of viscous drag and S is wetted area. The theoretically optimal shape to maximise internal volume and minimise wetted surface area is a sphere, however this will incur high pressure drag due to large separation. The optimal shape to minimise C_v is more complex [14], but for a minimum internal volume will roughly approximate a thin, slender body. Thus to reduce overall drag, naval architects must assess the trade-off of reduction in C_v with reduction in surface area.

In the 19th century, John Scott Russell observed that by some component of ship resistance must come from the making of waves, and asserted that by using mathematically defined "wave-line" geometries this wave-making component could be partly or wholly eliminated [15]. The wave-line theory was supplanted by Sir William Froude's philosophy of ship design following the *Raven* and *Swan* scale model experiments, where *Swan* produced lower resistance despite *Raven* conforming to wave-line theory. In these tests Froude set out scaling and resistance laws and defined the speed-length ratio, given by Equation 1.2

$$\textit{speed-length ratio} = \frac{v}{\sqrt{L}} \tag{1.2}$$

where v is velocity of the ship and L is the wetted length. This was later non-

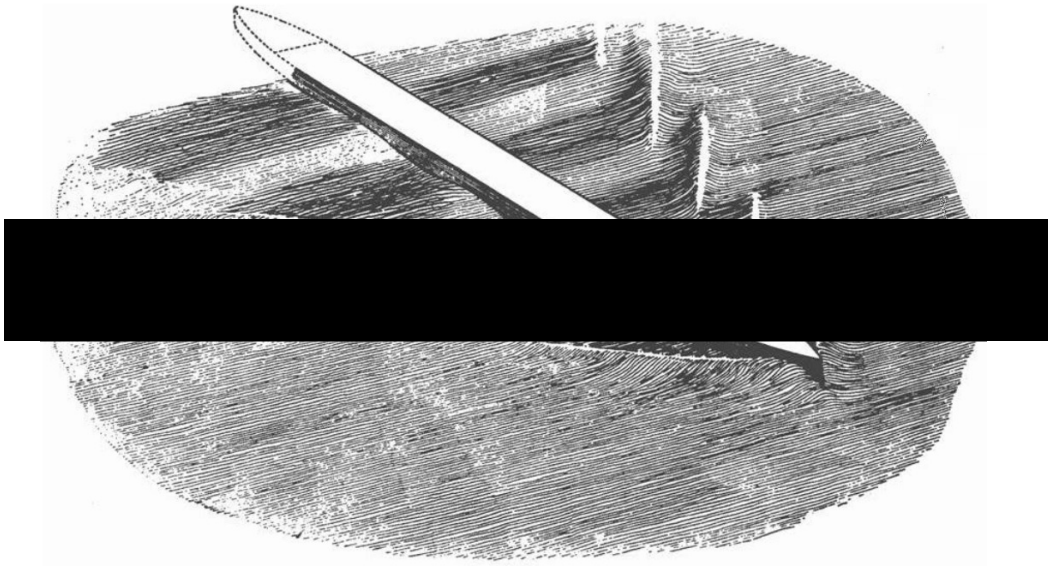


Figure 1.3: An illustration of the characteristic wave train pattern by Sir William Froude, 1877 (illustration reprinted [16]).

dimesionalised by adding a gravity term, g – thus defining the ratio between inertial and gravitational flow effects – and became known as the Froude number [2]. In this manner it is analogous to the Mach number in aerodynamics. The Froude number is given by Equation 1.3.

$$F_r = \frac{v}{\sqrt{gL}} \tag{1.3}$$

The Froude number remains to this day a useful and universal tool for naval architects in scale testing and flow regime definitions.

John Scott Russell's observation of the existence of wave-making resistance was not incorrect, wave-line theory was simply not the best way to minimise it. A characteristic wave train from a ship, as illustrated by Froude, is shown in Figure 1.3. The transverse wave starts at the bow, and can be assumed to travel at roughly the same speed as the vessel; its length will increase as speed increases, as given by the wave theory Equation 1.4.

$$\lambda = \frac{2\pi v^2}{g} \tag{1.4}$$

where λ is wavelength. As this bow wave pattern lengthens, either a peak or trough will coincide with a similar wave generated at the stern. When the wavelength of the bow wave matches the length of the vessel, the stern sits

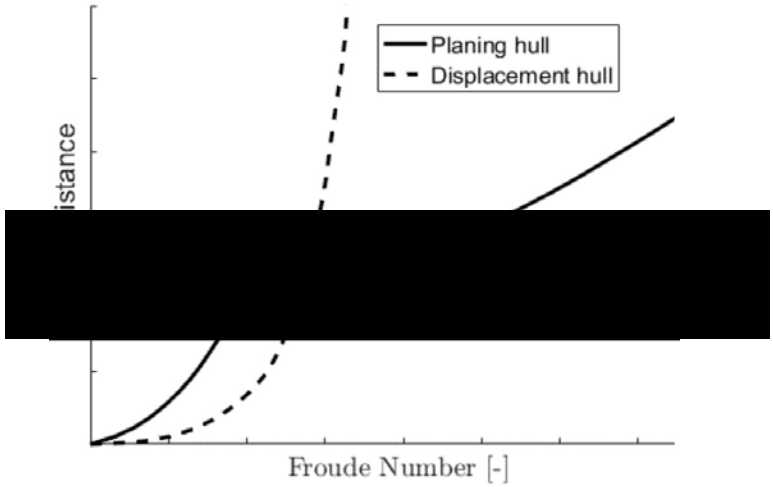


Figure 1.4: An illustration of how the characteristic difference between the design of a planing hull and a displacement hull affects resistance as Froude number increases [17].

in a large trough of the bow wave, causing large constructive interaction and sharply increasing wave-making resistance. The speed at which this occurs is referred to as "hull speed", and occurs at a Froude number of roughly 0.40. Substituting in a value of 0.4 for Froude number and $9.81 m/s^2$ for gravitational acceleration (g), hull speed is given by Equation 1.5

$$F_r = \frac{v}{\sqrt{gL}} \quad \rightarrow \quad v_{hull} \approx 0.4\sqrt{g}\sqrt{L_{WL}} \approx 1.25\sqrt{L_{WL}} \tag{1.5}$$

where L_{WL} is the length of the wetted waterline of the hull in meters and v_{hull} is the hull speed in meters per second. In Froude's era hull speed was considered effectively a "speed limit" for a hull, however with modern propulsion and design this is no longer the case in practice. Regardless, this does mean that displacement hulls become inefficient at higher Froude numbers.

Planing hulls, in contrast, can reach much higher Froude numbers relatively efficiently by rising out of the water with their lift component dominated by the hydrodynamic reaction force of the water as opposed to hydrostatic displacement. In this sense a planing hull is designed to act as a lifting surface as opposed to a displacement volume, and so the underlying physics qualitatively change. The character of the resistance acting on a planing hull versus a displacement hull as Froude number increases is illustrated in Figure 1.4 [17].

In Figure 1.4 a "hump" region is seen for the planing craft as it approaches its hull speed. As speed increases, the lifting nature of a planing hull leads to an increase in lifting hydrodynamic force to the point the boat rises from the water, reducing wetted area and displacement lift. It is at the point hydrodynamic lift exceeds hydrostatic that the boat is considered to be planing, though this transition region is often considered a distinct mode of travel in itself and gives the name to semi-planing (or semi-displacement) boats that operate predominantly in this region. In this transition region the boat will seem to be "climbing" its bow wave and trim angle will increase. As speed further increases, the boat typically rises onto the surface with a reduced trim angle and reduced wave making resistance, typically leading to a increase in efficiency as seen by the post-hump dip in Figure 1.4. At this point the boat is firmly in the planing regime, and resistance will once again increase with speed as new terms like spray drag and appendage drag increase [18]. This phenomenon has been observed in computational solutions of planing hull behaviour [19].

Resistance in planing mode is characteristically different to resistance in displacement as new terms become appreciable: spray drag caused by the fine spray generated by high speed flow fore of the spray root, and appendage drag caused by any protrusions from the smooth hullform (typically propulsion components) [18]. A breakdown of resistance for various ship types is given in Figure 1.5, where the tanker and container ship represent displacement vessels, the fishing vessel is semi-displacement, and finally a generic planing vessel [20]. Note that flat plate friction (i.e., viscous boundary layer resistance) dominates the resistance for the tanker and container ships due to their large wetted area. Some forms of resistance are exclusive to the planing hull, namely spray drag and appendage drag, while form resistance is negligible as the boat is planing on the surface. It should be noted this is only inclusive of the resistance associated with the water-wetted portion of the vessels, and aerodynamics for the air-wetted portion can be significant.

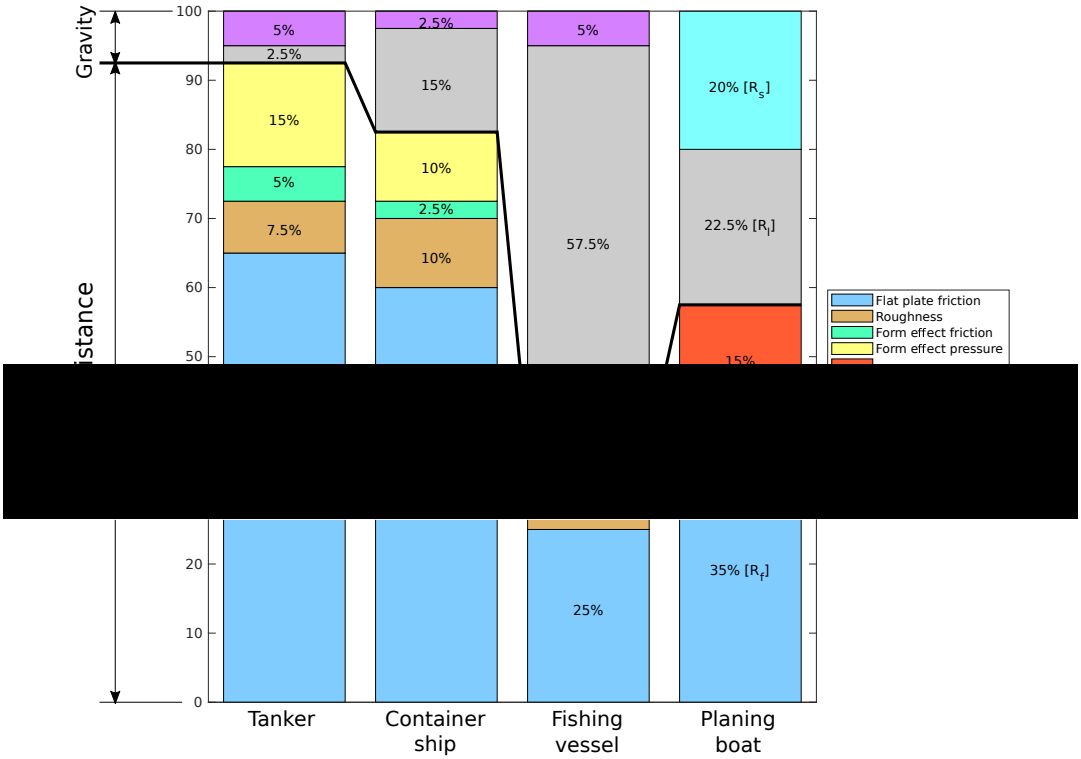


Figure 1.5: Resistance components for various hull types [20].

1.3.2 Geometry of a planing hull

Before a closer look at the physics of a planing hull, a rundown of the nomenclature attributed to planing hulls is necessary.

Figure 1.6 shows a simple planing monohull design. Monohulls are defined by having a single hull, as opposed to catamarans (twin hull), trimarans (triple hull), outriggers, or more exotic configurations. As is typical for most monohulls, the keel can be seen to run along the axis of symmetry at the lowest point, with the deadrise surface rising up to a sudden reflex that defines the chine. The hull surface then rises from the chine approximately vertically to the top of the hull. The area below the chine will be of most interest as the wetted area is typically confined here when planing. This configuration of deadrise surface from keel to a chine is typical, if not definitive, of planing hulls and is referred to as a vee-bottom hull or, when particularly aggressive, a deep-vee hull. The beam is defined as the width of the hull, and can be seen to reduce towards the front as is typical of most hulls. The transom is the

flat end at the stern (rear) where propulsion devices are typically mounted/-housed. Trim angle refers to the angle between the keel and the horizontal, and is analogous to the angle of attack of a lifting surface.

Draft is the term used to described the maximum depth of the hull below the waterline, and is used interchangeably with "sink". Due to the typical positive trim angle of planing boats, the deepest part of a planing hull will be at the intersection of the keel with the transom, and thus the depth of this point usually defines the draft or sink of the hull.

Deadrise angle often varies along the length of planing boats. Smaller deadrise angles tend to produce lower resistance, but also poor stability and seakeeping properties; thus the typical shape of a warped hull will have a steep deadrise at the bow to assist with wave-piercing and directional stability, flattening out to a shallower deadrise towards the stern to reduce resistance where stability is less affected due to the stern typically being submerged in planing, thus "biting" rather than "bouncing" off of incoming waves [21]. Extreme cases of this configuration are referred to as wave-piercing hulls.

In Figure 1.7 a more complex planing monohull is shown. In this design a double-step is employed. Steps are transverse discontinuities in the planing surface of the hull that usually run from chine to chine. At the chine the step often cuts an opening into the chine line to encourage air entrainment aft of the step (see side view in Figure 1.7). Steps have the effect of reducing the drag-to-lift ratio by reducing wetted area as flow separates off the step edge

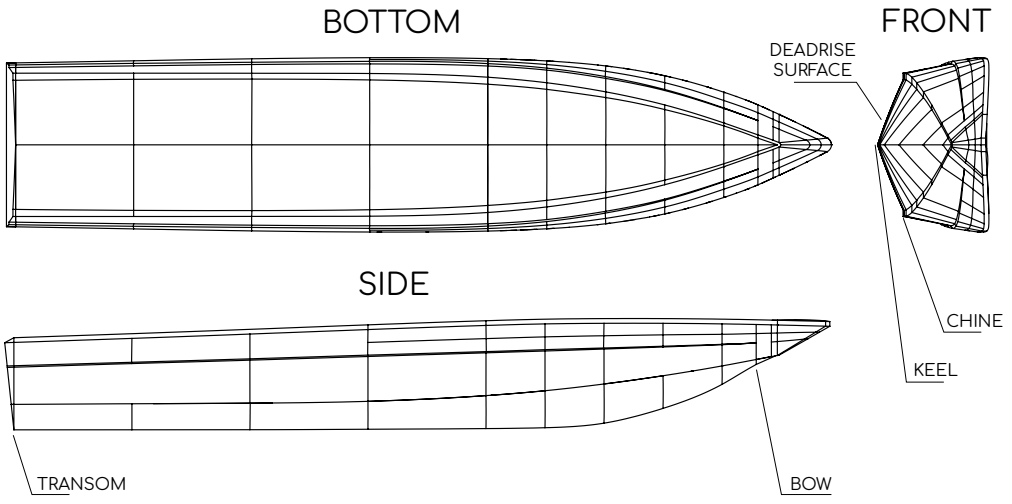


Figure 1.6: A planing monohull design is illustrated.

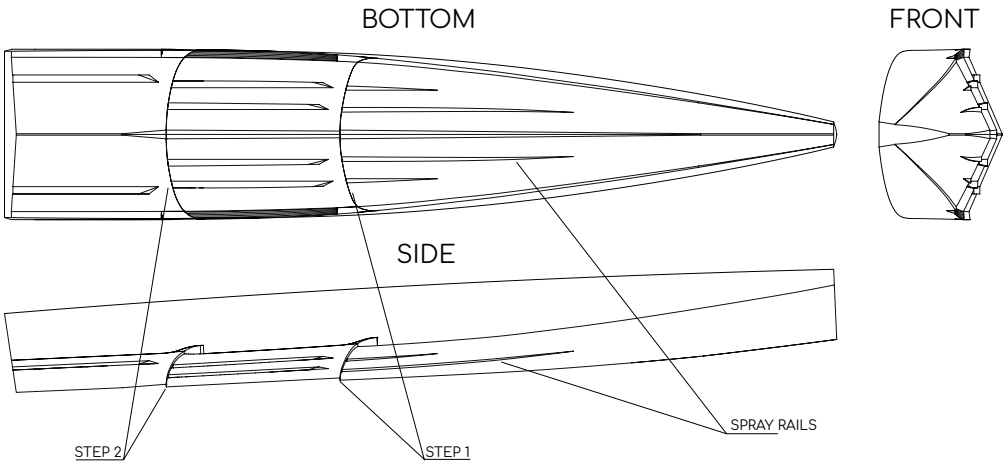


Figure 1.7: A double stepped planing monohull is illustrated.

and air is entrained via the opening at the chine [22]. On a hull with n steps, there will be $n+1$ high pressure stagnation patterns aft of the step where the flow reattaches. This has the effect of moving the overall centre of pressure aft and limiting the trim angle variability with Froude number to near-zero once planing, increasing comfort [22].

Spray rails can be seen to run longitudinally along the deadrise surfaces in Figure 1.7. Spray rails can act to deflect spray away from the hull, reducing spray drag and in some cases increasing lift, though this is dependant on many design factors such as deadrise angle and spray rail cross section shape and sizing [23].

An example of the flow separation aft of a single-stepped hull is shown in Figure 1.8, where reduction in wetted area due to air entrainment can clearly be seen. Some air entrainment along the spray rails can also be seen.

1.3.3 Analytical methods for planing hulls

There exists today a number of approaches to the analytic description of planing bodies, for example the Savitsky, Morabito, CAHI, Payne, and Shuford methods. These methods generally have a theoretical basis centred on classical mechanics, and are then modified to fit empirical data. An analytic approach to the complex flow around a planing body will necessarily require a lot of simplifications and assumptions, as a great deal of the more complex fluid behaviour cannot practically be fully captured. These assumptions may

Figure 1.8: An underwater still of a single-stepped hull, courtesy of Norson Design.

be accurate under certain conditions or in specific ranges, or combinations of the many relevant parameters. Fitting to empirical data goes some way to correcting this, but makes the method reliant on the scope and accuracy of the underlying empirical data.

The benefit of such analytic approaches is massively reduced order of complexity and computing cost when compared to a full 3D fluid simulation of the flow. They also allow designers without access to expensive tow tank testing facilities to get rough estimates of craft behaviour and tailor a hull to certain conditions. For this reason all of the above methods enjoy a degree of popularity among designers of planing craft – especially smaller designers with less access to expensive testing equipment – and serve as reasonable benchmarking methods in the field of CFD for planing hulls.

The mentioned methods have known limitations, arising from either the assumptions that underpin them, the range of results in the empirical data they are fitted to, or even the accuracy of those experiments. For this work it was determined that the Savitsky method would be used to benchmark solver performance alongside a set of empirical data. Masri et al composed a review of the mentioned methods [24], in which the following relevant properties of the Savitsky method were outlined:

- It is the most widely used analytic method in speedboat design, with a

wealth of related work in the literature.

- Applicable to steady state conditions only.
- Irrational behaviour at high deadrise angles ($\beta > 50°$).
- Lack of spray drag accounting.
- Results begin to deviate from experimental data at high trim angles ($\tau > 4°$).

As this method will be used to benchmark against numerical results of a solver that, initially, necessitates a fixed geometry, the limitation of steady-state conditions only are not of concern for preliminary benchmarking. The deadrise limit is not particularly restrictive, and selected empirical benchmarking data is for a hull of 20° deadrise, well within the limit of $\beta < 50°$. Spray drag accounting was added to the method later, as is expanded on in Section 1.3.4, and will be included in the comparisons. The trim angle restriction is not ideal, but if optimisation and modelling is targeted at a planing cruise regime then trim angle should not be excessively beyond this limit for most cases. In any case, comparisons can be made within the valid trim angle range, but this known behaviour should be borne in mind for higher trim angles.

Based on these considerations, benchmarking of the numerical methods developed in this work will be made against the Savitsky method and a set of empirical data (which is expanded on in Section 2.5.1). Due to the Savitsky method being based on the simplified physics of a planing hull, the method is expounded on in detail in the following section both to describe the method and to give understanding of the terminology and physics surrounding planing bodies.

1.3.4 The Savitsky semi-empirical method for planing hulls

Planing hulls were the focus of a number of experiments performed at the NACA towing tank in Langley and the Davidson Laboratory in New Jersey

in the mid-1900s [25, 26, 27, 28, 29]. In 1964 Daniel Savitsky published a paper on the behaviour of prismatic planing hulls, defining semi-empirical equations based on these experiments that describe lift, drag, wetted area, centre of pressure, and porpoising limits (a mode of longitudinal instability) as functions of speed, trim, deadrise angle, and loading [30]. This method has since been added to: in 1976 the findings were relayed in a manner accessible for designers [31] and Blount and Fox added power prediction [32]; in 2007 a whisker spray drag term was added to improve the overall resistance evaluation [33]; more recently new methods for warped hulls have been developed based on Savitsky's reduced method [21].

The Savitsky method begins by considering the physics of a planing flat plate, as illustrated in Figure 1.9, before moving on to hulls with a deadrise angle. In Figure 1.9, τ is the trim angle, d the draft (depth), and v the forward velocity of the plate. Following Savitsky's conventions, λ is the mean wetted length to beam ratio taking into account the local wave rise, and λ' is the mean wetted length to beam ratio based on the undisturbed far-field water surface; thus these values multiplied by beam, b, are simply the respective wetted lengths. The way in which a wave rise is generated ahead of a planing flat plate is shown in an exaggerated manner in Figure 1.9. Also depicted is how this wave rise continues up the plate forming the spray region. The contribution of this spray region is neglected for now, but is incorporated in later Savitsky analysis [31].

The corresponding pressure profile is shown in Figure 1.10, with the stagnation pressure defined from Bernoulli's principle.

The pressure on a planing hull is comprised of two component pressures, the static pressure arising from displacement, and dynamic pressure arising from the reaction of the hull against the water when moving. At low speeds, the static pressure dominates; at high speeds, the dynamics pressure dominates. Savitsky's method takes into account both of these pressures, within limits discussed later.

The fluid flow direction across the plate will be a mixture of longitudinal and transverse flow. A planing boat geometry can be thought of as a very

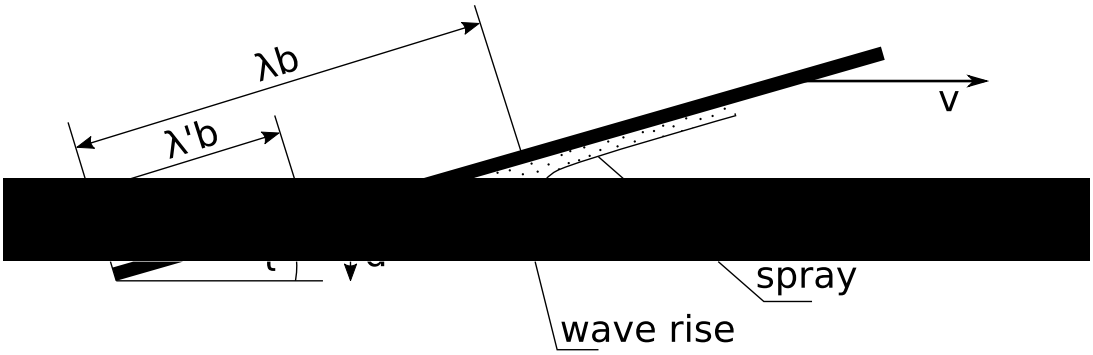


Figure 1.9: A planing flat plate as described by Savitsky [30]. v is the forward velocity, d the draft, τ the trim angle, and b the plate beam (width). λ is the wetted length to beam ratio, such that λb is simply the wetted length. The superscript $'$ denotes wetted length relative to the far-field undisturbed water level.

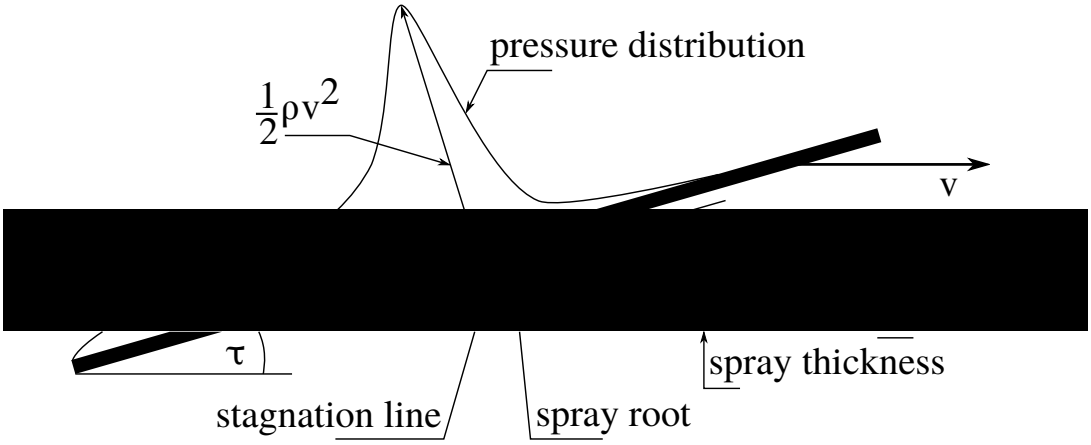


Figure 1.10: The general pressure distribution of a planing flat plate as described by Savitsky [30]

low aspect ratio wing, with large chord and small span. From aerodynamic theory, such wings have high transverse flow (i.e. parallel to direction of boat travel), and very small chordwise flow (i.e. perpendicular to direction of boat travel). For a zero aspect ratio plate of infinite length and zero span ($\lambda = \infty$, $b = 0$), Savitsky argues the lift is proportional to τ^2, and can be expressed as in Equation 1.6.

$$C_L = A\tau + B\tau^2 \tag{1.6}$$

For λ values practical for planing hulls (assumed to lie in the range $\lambda \leq 4$), empirical testing showed that Equation 1.6 can be approximated using $\tau^{1.1}$, giving the relationship in Equation 1.7.

$$\frac{C_L}{\tau^{1.1}} = f(\lambda, C_v) \tag{1.7}$$

For the dynamic term, Sottorf [25], who's method assumes negligible buoyant pressure contribution, showed that dynamic lift ($C_{L,dynamic}$) varies with $\lambda^{\frac{1}{2}}$ for a given trim angle as in Equation 1.8.

$$C_{L,dynamic} = c\lambda^{\frac{1}{2}}\tau^{1.1} \tag{1.8}$$

where c is an unknown constant to be solved for.

The buoyant lift term can then be calculated from the volume displaced by the flat plate. This is calculated from the density of the fluid displaced multiplied by the volume displaced.

$$L_{buoyant} = V\rho g \tag{1.9}$$

where V is the displaced volume. Savitsky empirically determined that $\lambda = \lambda' + 0.30$ for wetted length ratios in the range of $1 \leq \lambda \leq 4$. Rearranging this for λ' gives the following relationship for λ', which is the mean wetted length ratio neglecting wave rise.

$$\lambda' = \lambda - 0.30 \tag{1.10}$$

Thus the volume displaced by a flat plate of beam b, trim angle τ, and wetted length $\lambda' b$, is that of a right-triangular prism as expressed in Equation 1.11.

$$V = \frac{1}{2} \times \lambda' b \tan(\tau) \times \lambda' b \times b = \frac{1}{2} b^3 \lambda'^2 \tan(\tau) \tag{1.11}$$

Substituting Equation 1.11 and 1.10 into Equation 1.9 gives the equation for the buoyant lift term.

$$L_{buoyant} = \frac{1}{2}\rho g b^3 (\lambda - 0.30)^2 \tan(\tau) \tag{1.12}$$

To non-dimensionalise 1.12 as a lift coefficient to be combined with the dynamic lift coefficient, it is divided through by $\frac{1}{2}\rho v^2 b^2$, and the $(\lambda - 0.30)^2$

term is expressed as $D\lambda^n$, where K and n are constants to be determined.

$$C_{L,buoyant} = \frac{D\lambda^n}{C_v^2}\tan(\tau) \tag{1.13}$$

Assuming that, given the small angle τ, $tan(\tau)$ and $\tau^{1.1}$ are equal, this can be re-expressed as follows:

$$C_{L,buoyant} = \frac{D\lambda^n}{C_v^2}\tau^{1.1} \tag{1.14}$$

Thus a lift coefficient accounting for both buoyant and dynamic components can be expressed by combining Equations 1.8 and 1.14.

$$C_L = \tau^{1.1}\left(c\lambda^{\frac{1}{2}} + \frac{D\lambda^n}{C_v^2}\right) \tag{1.15}$$

Savitsky then evaluated the constants of c, D, and n from experimental data, giving the readily usable semi-empirical expression given in Equation 1.16 in terms of commonly used parameters in boat design.

$$C_L = \tau^{1.1}\left(0.012\lambda^{\frac{1}{2}} + \frac{0.0055\lambda^{\frac{1}{2}}}{C_v^2}\right) \tag{1.16}$$

The fitting of the empirical data to Equations 1.10 and 1.7 is valid for the regime of $0.60 \leq C_v \leq 13.00$, $2^\circ \leq \tau \leq 15^\circ$, and $\lambda \leq 4$, and so Equation 1.16 is limited to this regime.

Equation 1.16 is intended for a flat plate, which does not represent the vast majority of planing vessels. The lift coefficient for a prismatic planing hull of deadrise β was determined by Savitsky as a function of deadrise angle β in degrees and the equivalent flat plate lift coefficient (Equation 1.16) at equivalent τ, C_v, and λ values.

$$C_{L,\beta} = C_{L0} - 0.0065\beta C_{L0}^{0.60} \tag{1.17}$$

where C_{L0} is the lift coefficient of the flat plate equivalent as given by Equation 1.16. The generic prismatic hull geometry that is assumed is depicted in Figure 1.11.

Prismatic hulls are defined by a constant cross sectional area. In reality

few to no hulls are perfectly prismatic, and variable deadrise surfaces are common. Any discontinuities such as steps will also make a hull non-prismatic. Despite this, many hulls can be approximated as prismatic, making the presented Savitsky method very popular among boat designers, as a reasonable estimation of lift generation can be made at very low computational cost. Compare Figure 1.11 with Figure 1.12 to visualise the similarities between the prismatic idealisation and a "real" variable deadrise hull.

The Savitsky method is easy to use, computationally cheap, well established in the literature, and has been used in the benchmarking of computational methods [34]. In Section 2.5 a benchmarking against the Savitsky method is made for these reasons.

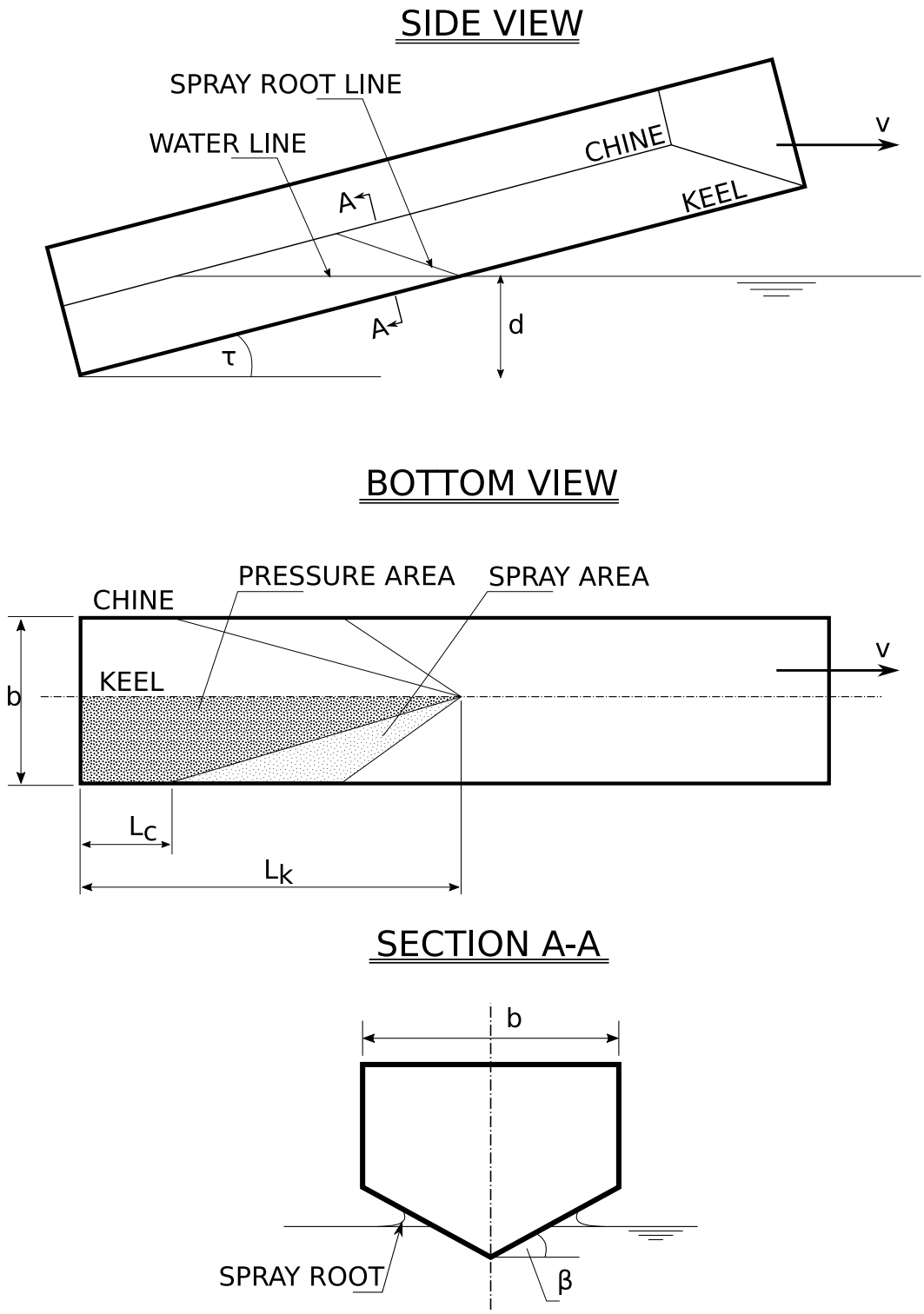


Figure 1.11: A prismatic hull of deadrise angle β is depicted. d is the draft, or the depth of the lowest point of the transom (rear of boat). v is the forward velocity of the boat. L_c and L_k are the wetted chine and keel lengths respectively. b is the beam (width) of the boat.

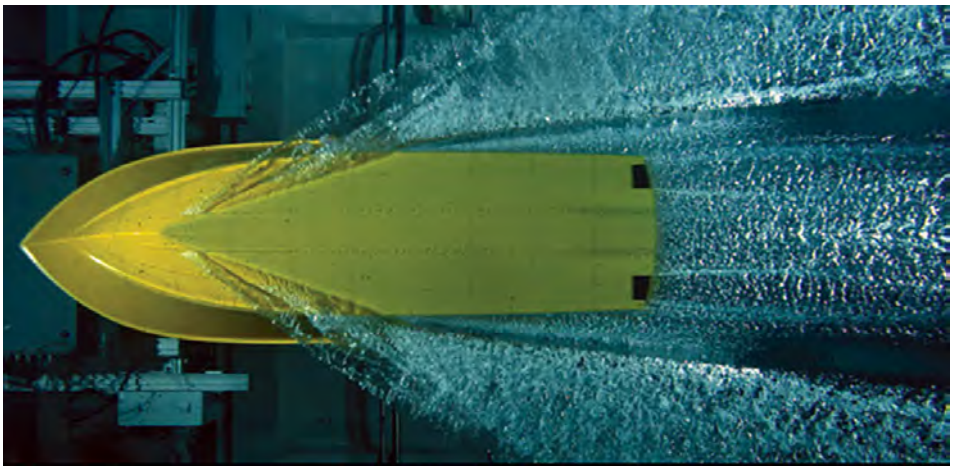


Figure 1.12: An underwater shot of a planing hull being tested at the Steven's Institute of technology.

1.4 The state of the art in planing hull CFD

It is the last ten years that CFD (Computational Fluid Dynamics) for ship hydrodynamics has made its greatest strides. This is primarily owed to advancements in the fields of free-surface tracking and capturing, turbulence modelling, 6 Degree-of-Freedom motion prediction, adaptive grid refinement, high-performance computing, and optimisation methods [35]. A comprehensive evaluation of the history and current state of CFD in hull design is given by Stern et al [35].

Since the inception of CFD in the 1960s, attempts to apply CFD techniques to ship design have been made with increasing sophistication. Integral methods were popular up until the 1980s, utilising the momentum integral equation, entrainment equation and cross-flow momentum equations [36]. This method fell out of favour due to the difficulty in extending the cross-flow velocity profiles to three-dimensions and sensitivity of the potential flow to inflections in the fairness of the hullform [37]. This prompted a move towards three-dimensional finite difference solvers for boundary layers, which were apt for thin boundary layers, but failed on thicker boundary layers, separation of flow, and regions near the free-surface boundary [38]. To improve on the consistency of the solution in these regions, partial-parabolic Reynolds approaches were developed [39]. In the 1990s, full Reynolds Averaged Navier-Stokes (RANS) solvers with viscous-inviscid non-zero Froude

number capabilities to solve hullform boundary layers and wakes [40]. Free-surface tracking [41], improved 2-equation turbulence models, and body-force propulsor modelling were subsequently developed [42].

The last ten years marked a significant increase in the sophistication of CFD in ship modelling, and significant progress towards the "virtual tow tank" has been made. The concept of the virtual tow tank is to create a software suite that can accurately carry out experiments as one would perform in a conventional tow tank. This would include free-surface fluid modelling with dynamic interaction between a rigid hullform and the water, with various wave patterns and conditions. Carrica et al [43] made a significant step towards this goal in 2007, enabling wave breaking and ship pitch and heave motion to be modelled concurrently using level-set free surface tracking, dynamic overset grids, and inertial reference frames. This involved two reference frames, one earth-based inertial frame on which the fluid flow was solved, and another ship-based reference frame is used to compute the rigid body motion of the ship, with interpolation occurring between the two at each timestep.

Numerically modelling the full (or even partial) physics of a ship is a highly complex task which requires a great deal of computational power. To meet this large computational requirement High Performance Computing (HPC) clusters are employed. HPC clusters consist of a multitude of dedicated computers linked through a LAN (Local Area Network) system, which can be used to process computationally intensive tasks. The effect is one of having more resources available to split the processing of calculations between, rather than a more "powerful" processing capability, and so there is a reliance on the ability to distribute and communicate the task among the distributed processors. Distribution and coordination of tasks is handled using a Message Passing Interface (MPI), for which many open-source versions exists. Details on the HPC system used are found in Section 1.6.

1.4.1 Existing software

There are a number of computational hydrodynamic software packages aimed at ship design and optimisation currently in continuous development. The

tools are grouped here into three categories.

The first are continuum free surface single-phase solvers. These are mesh-based continuum solvers that utilise some form of free surface modelling for a single phase (the water). These solvers neglect the air phase, which is acceptable in some applications due to the sometimes negligible impact on ship performance by the air phase relative to that of the water phase. The air phase is, however, very important to model for full understanding of phenomena such as bubbly wake, air entrainment, cavitation, propeller modelling, wave breaking, and wind-related seakeeping studies. Due to this, several of the software packages in this category are expanding capabilities to model multi-phase flow.

The second category are continuum multi-phase solvers. These are solvers that are still mesh based continuum solvers, but are capable of modelling multiple phases concurrently (air and water in the case of boat design). This style of modelling is closer to the fundamental physics of hydrodynamics, and therefore more desirable for attaining the virtual tow tank.

The final category considered here are Lattice Boltzmann Method (LBM) solvers, which is the method that was selected for the modelling presented later in this thesis. A full introduction to the LBM is given in Section 1.5, and the rationale for using the LBM is given in Section 1.5.3.

The strengths, weaknesses, and primary features of each of these tools are detailed in the following subsection. An overall assessment of the existing software is given, along with an exploration of the capabilities desired in the future.

Continuum free surface solvers

- *CFDShip-Iowa* [44].

  Developed by University of Iowa's Hydroscience and Engineering group and sponsored by the US Office of Naval Research, CFDShip-Iowa is a incompressible URANS/DES continuum solver with a variety of turbulence models. A single-phase level set method with Volume of Fluid (VoF) method [45] is used for free surface/interface modelling.

The solver is computed on a structured grid using the finite difference method. Overset grids are used to accommodate complex geometries, and dynamic overset grids allow motion of rigid bodies; it should be noted that overset grids typically incur high computational costs in the grid generation phase. Single-phase level-set approach is used for the free surface tracking. CFDShip-Iowa is parallelised for HPC processing.

CFDShip-Iowa is generalised to a number of external flow problems, but differs from STAR-CCM+ in the sense that it is specifically developed for surface-vessel performance analysis rather than multi-physics capabilities. This means it has a hydrodynamic post-processor specifically tailored for ship design criteria such as skin friction, pressure drag, wave profiles, 6-DoF seakeeping, and manoeuvring. Propeller modelling and multiple wave profiles are also included.

Capabilities include fast model-scale sinkage, slamming/water impact, trim prediction, wave breaking, wave-body interaction (for simple geometries), and bubble/droplet size and distribution prediction.

CFDShip-Iowa has been developed over twenty years, with many stages of validation performed across that time. The code has also been proven to be viable for high-speed applications.

The primary downside to CFDShip-Iowa, however, is that its usability is restricted due to its sponsorship by the US Navy ONR. Additionally, CFDShip-Iowa is a free surface, single-phase solver; initially, this limited its ability to directly model hydrodynamic phenomena where mixing of the air and water occur, such as cavitation, bubbles, spray, and droplets. Cavitation in particular - which can damage structures and reduce the efficacy of hydrofoils, propellers, and rudders - proved difficult to model initially. In CFDShip-Iowa v6.2, however, a sharp-interface cavitation model using volume of fluid method was added [46].

The latest versions of CFDShip-Iowa (v6.1-3) aim to increase accuracy, robustness, and exascale HPC capability [47]. In v6.1, level set based ghost fluid method is used for sharp interface treatment and two-

phase coupling with VoF method for tracking the interface. This was extended from Cartesian grids to orthogonal curvilinear grids with improved interface modelling in v6.2. Targets for v6.3 include fully coupled, multi-scale, multi-phase, turbulent flows around immersed structures.

- *CREATE-Ships (IHDE)* [48].

  The CREATE-Ships (Computational Research & Engineering for Acquisition Tools & Environments) IHDE (Integrated Hydrodynamic Design Environment) is a project under development by the United States Navy to deliver rapid, early-stage ship design influence capabilities to the US Navy, where empirical data, tow tank tests, and full model tests are still relied on [49]. The US Navy has a limited ability to impact early stage design, and greater abilities to do so could be economical in terms of both financial expense and time. The IHDE aspect involves streamlining workflow and minimising turn around time by integrating all the tools into a single desktop application for easy usability; this would ensure that analysis could be performed in the limited timeframe of the pre-design phase.

  CREATE-Ships has a suite of tools for resistance analysis, seakeeping, hydrodynamic loads, and operability. In terms of hydrodynamics modelling, CREATE-Ships uses: Total Ship Drag (TSD), a slender ship model for resistance prediction [50]; Das Boot, a potential flow [51] code with non-linear free surface boundary modelling for assessing sink and trim [52]; AEGIR, a 3D potential flow solver for wave resistance, sinkage, and trim [53]; as well as a number of seakeeping, hydrodynamic loads and operability tools [49]. These tools may be accurate enough when balanced against fast turn around times at the pre-design stage of projects, but they lack the full physics capabilities of other solvers. Potential flow solvers, for example, deliver an irrotational, steady solution cannot simulate complex phenomena such as spray and wave breaking [54]. Nevertheless, CREATE-Ships is still in development with targets to expand the physics models and add features such as propeller modelling.

Sponsorship by the United States Department of Defence (DoD) means that use of the system is limited for non-government bodies. A public version containing limited features is available, although this is restricted to authorised US institutions such as universities and research centres on a case-by-case basis [49].

- *FINE^{TM}/Marine* [55].

  FINE^{TM}/Marine is a suite of FINE^{TM} (**F**low **In**tegrated **E**nvironment) from the CFD and multi-physics company Numeca. FINE^{TM}/Marine is designed for free surface marine flow problems such as ships and offshore installations, with single and multi-phase capabilities through a Graphical User Interface (GUI) for usability. A number of design studies based on analysis by FINE^{TM}/Marine can be found in the literature [56, 57, **?**].

  FINE^{TM}/Marine uses a scriptable hexahedral unstructured mesh (thus enabling complex geometry capture), and includes a dedicated post-processing suite (CFView^{TM}) with marine orientated options. The solver is capable of free surface capturing and 6DoF motion for rigid bodies, with adaptive grid refinement, overlapping grids, and cavitation modelling.

Continuum multi-phase solvers

- *STAR-CCM+* [58].

  A commercially available software suite developed by CD-Adapco, STAR-CCM+ is an industry standard multi-tool for CAD that has been used in a number of studies with marine application [59, 60]. The software includes a polyhedral and hexahedral meshing algorithm with automatic surface repair that streamlines workflow. Overset meshes are utilised; this involves placing one mesh over another, allowing tailored precision for the range of motion that might be expected of a boat in chop. Processing is performed on a client-server basis, enabling low-spec laptops to connect to high performance computing (HPC) clusters (HPC servers are a service also offered by CD-Adapco) to perform computa-

tions. A number of solvers are available depending on the problem at hand, including multi-phase capabilities.

STAR-CCM+ offers a wide variety of solvers and techniques for many engineering applications; all of these solvers are fundamentally mesh based.

- *ANSYS Fluent* [61].

  Fluent is a CFD solver and design environment from the CAD company ANSYS Inc, and is among the most-used CFD solvers in the world. Fluent is well established and widely used in industry, offering a range of functionality including free surface and multi-phase flows based on volume-of-fluids models.

- *OpenFOAM* [62].

  OpenFOAM is an open source CFD solver distributed under a GPL v3 license. Due to OpenFOAM's open-source nature, it is popular in CFD research. A number of ship design studies using OpenFOAM can be found in the literature [63, 64].

Lattice Boltzmann methods

- *LaBS* [65].

  LaBS (**La**ttice **B**oltzmann **S**olver) is a commercial CFD code based on the Lattice Boltzmann Method. It is developed within the framework of a consortium including Renault, Airbus and CS Communication et Systèmes, with CS Communication et Systèmes in charge of distributing the software.

  LaBS excels at solving compressible flows to a degree of accuracy that allows study of aeroacoustic fields by directly modelling aeroacoustic behaviour [66, 67], however no applications specific to maritime flow problems or design are available.

- *LBHydra* [68].

LBHydra is a Lattice-Boltzmann simulation package capable of modelling laminar and turbulent flows, heat and mass transport, and multiphase fluids in transient flow around 2D and 3D geometries [69, 70]. LBHydra is developed primarily by the University of Minnesota and written in C++ and CUDA, allowing it to be run easily on GPUs [71].

- *Palabos* [72].

  Palabos (**Pa**rallel **La**ttice **Bo**ltzmann **S**olver) is an open-source LBM-based solver for general purpose fluid flow simulations distributed under an AGPLv3 license. Although freely available, the company FlowKit Ltd. offers professional support, training and consultation for the Palabos platform. The native library interface is written in C++ with no further external dependencies making it highly portable and easy to use.

  Palabos takes particular advantage of the LBM suitability to parallelisation, with optimisation techniques in I/O communication and memory access yielding excellent scalability [73].

  Palabos supports a range of functionalities at present, and is in ongoing development. With regard to physics models, both single and multiphase flows are implemented (Shan/Chen and He/Lee model [74]), as well as volume of fluid method for free-surface flows and a static Smagorinsky/Large Eddy Simulation (LES) model for fluid turbulence [75].

  Palabos is designed to receive input geometries in `.stl` format, which are then voxelised to establish the computational domain; this pre-processing stage is integrated into the Palabos workflow and takes place in parallel, greatly reducing the time dedicated to pre-processing. Alternatively, simple domains can be defined manually by the user.

  Post-processing capabilities include direct writing of outputs to an ASCII or binary file, or directly to GIF images or .VTK files, which can later be processed by suitable software (e.g.: Paraview, EnSight, etc.). To reap the benefits of parallelisation in post-processing as well as in pre-processing, Palabos is also capable of natively computing streamlines,

isosurfaces, and contours.

- XFlow [76].

  XFlow is a commercial LBM-based solver with suites for automotive, aerospace, manufacturing, energy, civil, and marine applications with user friendly GUI and built-in pre- and post-processing. XFlow is distributed by Next Limit Technologies, and as a commercial software package comes with an expensive license fee on a per-core basis; early investigation revealed that a license would be prohibitively expensive.

Conclusions on existing software

There exist a number of software packages with varying degrees of capability in modelling hydrodynamic studies. There is not, however, a software package that clearly excels in all capabilities without downsides. Firstly, there is the case of cost; the most sophisticated software consistently come at a very high cost that may be acceptable to larger industries, but are not conducive to research by smaller companies or in academia. Also, it should be considered that some of these packages (e.g.: CFDShip-Iowa, CREATE-Ships) are off limits to a wider user base due to their national restrictions.

Overall, prohibitively high cost (or special requirements for access) apply to all the software packages that approach the level of sophistication required to be considered virtual tow tanks. Those that are free and open, however, lack specific features and capabilities. While most have some form of post-processing for data visualisation and results, only CREATE-Ships endeavours to provide geometry creation, modelling, optimisation, and geometry alteration in a single, encompassing design environment. In this respect there is great scope for a software package that delivers such a design environment with a solver component that strives to act as a virtual tow tank.

1.4.2 Shortcomings of modern computational hull design

In a review paper on the state of CFD in maritime applications, Stern et al. call for the following advancements [35]: preparation for the onset of exascale HPC; better parallelisation and scaling for HPC systems; more efficient use of computational power; a transition to first-principle based solutions; order-of-magnitude improvement in accuracy, robustness and performance .

To fully exploit the benefits of exascale computing, current software may require large changes or even complete overhauls [35]. This is due partly to poor scalability, and the current dependence on phenomenological or empirical models. Many RANS solvers are expected to reach a limit where increased grid size no longer confers benefits to accuracy, as turbulence models remain a limiting factor. Although discarding turbulence models in favour of Direct Numerical Simulation (DNS) [77] of turbulence may solve this, it comes at incredibly high computational cost. An alternate compromise is methods such as Large Eddy Simulation (LES) [78], where the smallest spatial and temporal scales are essentially averaged out to reduce computational cost of fully resolving these lengthscales as in a DNS approach. Additionally, RANS approaches do not inherently excel at free-surface modelling, and often rely on simplifications of phenomena such as wave breaking, cavitation, and wake bubbles.

In particular, it is the desire for first-principle models, better free-surface modelling, and better parallelisation that make models such as the Lattice Boltzmann Method (LBM) attractive. The theory of LBM and case for using it in this application are made in the following section.

1.5 The lattice Boltzmann method for planing hullforms

On the balance of the findings in the previous sections, the Lattice Boltzmann Method (LBM) was selected for this project as the technique with which to approach the modelling of high speed planing hullforms. This decision was made largely on the basis of its inherent parallelisation for running on HPC clusters, ability to model free surface flows effectively, effective handling of complex geometries, and second-order accuracy in solving the Navier-Stokes equations. All these points are explained in detail in Section 1.5.3.

In Section 1.5.1, a brief introduction to LBM is given. In Section 1.5.2, the theory of LBM is then described in detail, with attention given to the particular model used for this work. Finally, in Section 1.5.3 the argument for its use in modelling planing hullforms is made.

1.5.1 An introduction to the lattice Boltzmann method

Fluids can be described in terms of properties such as velocity, density, and pressure. In a continuum modelling approach, these properties are assumed to vary in a continuous manner across a spatial field. This grants the benefit of being able to use methods based on differential equations to describe the flow; examples of such methods are potential flow methods, Euler methods, and Navier-Stokes methods. Applied with a variety of spatial and temporal discretisation approaches, these methods enjoy widespread use as industrial CFD tools.

Fluid behaviour adheres well to a continuum description on a macroscopic scale, however this breaks down as the length-scale of a flow problem approaches that of the size of the constituent molecules. On a molecular length-scale, fluids are made up of discrete molecules rather than continuous fields and are fundamentally characterised as having large intramolecular distances relative to their molecular dimensions: properties like density are zero at a point between molecules, and very large at a point lying inside a molecule. For the continuum approximation to be valid a sufficient density of particles

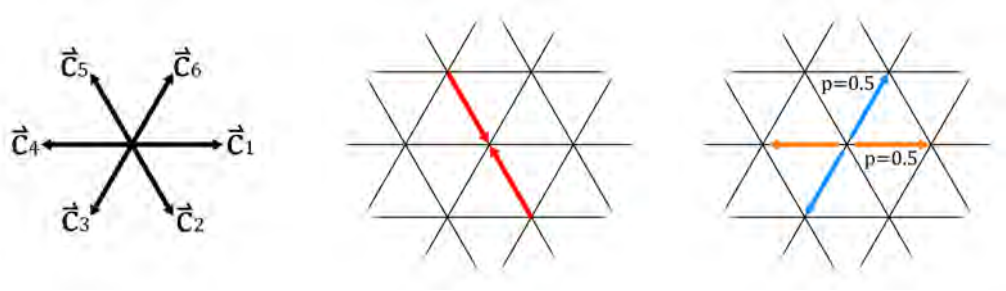
is needed; as a result, methods arising from a continuum assumption are unfit for rarefied flows (extreme low density) or flows where the particle size approaches the length-scale of interest (nanoscale applications).

Atomistic or molecular modelling approaches seek to describe fluids by directly modelling the interactions of these molecules. The prohibitive pitfall of atomistic approaches is the incredible amount of computer processing power and memory required to model the huge number of molecules in applications that do not have extremely rarefied flow.

A compromise between continuum and atomistic methods is a probabilistic one, where the probability of finding particles at a given point is sought, which is the basis of kinetic theory laid out by Boltzmann and Maxwell in the 1800s [79, 80]. A statistical model is a compromise between the macroscopic continuum approach and microscopic atomistic approach, and due to this is sometimes referred to as a mesoscopic method.

Historically, LBM is considered the successor to the Lattice Gas Automata (LGA) of the 1980s [81]. The foremost LGA model was the FHP (Frisch-Hasslacher-Pomeau) model, which was based on a discrete hexagonal grid. At intersections on this grid are nodes, which have six links extending to neighbouring nodes. Idealised gas molecules populate the grid, and can exist on any one of the six links around a node. In an iteration, an idealised particle will travel to one of the nearest neighbouring nodes along one of these links. The next step is collision, where molecules on the node are redirected to two different links on that node depending on their incoming state. This is illustrated in Figure 1.13.

Given that enough molecules populate the grid, LGA FHP is capable of fully modelling compressible fluid flow. There are, however, some serious downsides to LGA that prompted the evolution to LBM: namely, a huge number of molecules are needed for the model not to be dominated by numerical noise, and discrete state calculation was not as quickly computable as floating point arithmetic. The LBM resolved these issues by using a statistical description of the molecules rather than discrete molecular states. The theory of the LBM is described in the following section.



(a) (b) (c)

Figure 1.13: An illustration of LGA FHP process:
(a): A node on the discrete hexagonal grid has six links connecting it to neighbouring nodes. Molecules populate the grid at these nodes.
(b): During an iteration of the model, two example particles travel from their node to a neighbouring node along a link.
(c): Collision takes place, and each molecule at the node is assigned a new link depending on their initial state (either the blue or the orange state would occur, with momentum, p, being conserved).

1.5.2 Lattice Boltzmann method theory

The lattice Boltzmann method is described with a focus on the specific manner in which it is applied in the solver module used for this work [82]. A full, in-depth theory of the lattice Boltzmann method is given by Krüger et al. [83], including a full derivation of the distribution function.

While a finite volume Navier-Stokes (N-S) approach might directly discretise the N-S equations, the LBM approach consists of a first-order explicit discretisation of the Boltzmann equation in a discrete phase-space. The movement and collisions of these particles on the grid is computed, and the incompressible N-S equations are replicated to second-order accuracy. This second order accuracy is proved by Frisch et al. by way of Chapman-Enskog expansion [84], and by He and Luo by direct discretisation of the Boltzmann equation [85].

The underlying kinetic theory for LBM relies on describing the position and velocity of the particles. It is prohibitive to model every particle in the flow, and so instead a statistical description is given by way of a velocity distribution function. This function describes the probability of gas molecule being found at a given point in space with a given velocity.

In a three dimensional model, the distribution function is dependent on

seven variables: three for space, three for velocity, and one for time. This is mathematically represented as:

$$f(\mathbf{x}, \mathbf{e}, t) \tag{1.18}$$

where position and velocity vectors respectively are:

$$\mathbf{x} = (x, y, z) \qquad \text{and} \qquad \mathbf{e} = (e_x, e_y, e_z) \tag{1.19}$$

The $\mathbf{x}$ vector holds the three-dimensional position values, the $\mathbf{e}$ vector holds the three corresponding velocity values, and t denotes time.

The macroscopic variables in which flows are typically thought of can be recovered from this distribution function. Because the macroscopic variables (density, velocity, temperature) are dependent on position and time, not velocity, arriving at them from the distribution function typically requires integrating over the velocity space.

Mass density, ρ, is found from the distribution function by integrating the distribution function across the velocity space as in Equation 1.20, where m is the molecular mass.

$$\rho(\mathbf{x}, t) = m \int f(\mathbf{x}, \mathbf{e}, t) \quad d^3e \tag{1.20}$$

The velocity of the flow, $\mathbf{u}$, can be found by weighting this integral by the velocity of the molecules, $\mathbf{e}$, as in Equation 1.21.

$$\mathbf{u}(\mathbf{x}, t) = \frac{m}{\rho} \int \mathbf{e} f(\mathbf{x}, \mathbf{e}, t) \quad d^3e \tag{1.21}$$

Temperature, T, arrives from the kinetic energy of the gas. For an ideal gas the expected value of the square molecular velocity, as expressed in Equation 1.22, where R is the universal gas constant.

$$T = \frac{m}{3R\rho} \int |\mathbf{e}^2| f(\mathbf{x}, \mathbf{e}, t) \quad d^3e \tag{1.22}$$

The physical rules for the velocity distribution function come from the Boltzmann equation that gives LBM its name. This equation is expressed in

its general form in Equation 1.23.

$$\frac{df}{dt} = \left.\frac{\partial f}{\partial t}\right|_{force} + \left.\frac{\partial f}{\partial t}\right|_{diffusion} + \left.\frac{\partial f}{\partial t}\right|_{collision} \tag{1.23}$$

The force term relates to external forcing, the diffusion term relates to particles in free motion, and the collision term relates to the pairwise collision of particles. Recall that f is the velocity distribution function. The forcing and diffusion terms can be expressed such that the equation takes the form in Equation 1.24.

$$\frac{df}{dt} + \frac{\mathbf{p}}{m} \cdot \nabla f + \mathbf{F} \cdot \frac{\partial f}{\partial \mathbf{p}} = \left.\frac{\partial f}{\partial t}\right|_{collision} \tag{1.24}$$

where $\mathbf{p}$ is momentum ($\mathbf{p} = m\mathbf{e}$), $\mathbf{F}$ is an external force. The collision term is notably not expanded on here. Conceptually, the collision term describes how the momentum of colliding particles is changed based on their initial conditions. This term is usually complicated, even for a simple elastic collision between spheres, and usually takes the form of a simplified expression that achieves a replication of the macroscopic physics at reasonable computational cost. The collision term used in this work is expanded on shortly.

The Boltzmann equation – as is typical of equations describing fluid dynamics – is too complex to be solved analytically for all but the most trivial problems: instead, it must be solved numerically on a Cartesian grid of cell. Each grid cell affects only its neighbouring cell. In this work, the D3Q19 stencil [74] is used on a Cartesian grid of uniform grid cell size to discretise the velocity-space. The D3Q19 stencil is so named for featuring 19 velocity vectors in 3-dimensional space, and is a popular choice for solving incompressible flows as it sits in a middle-ground between the less accurate D3Q15 and more memory-intensive D3Q27 velocity sets [83]. The D3Q19 stencil is depicted in Figure 1.14. Each arrow represents a variable located on that grid node, and depicts the connectivity to the neighbouring grid node. There are many other types of velocity stencils including 2D variants, though the most popular for 3D cases are D3Q15, D3Q19, and D3Q27. Larger velocity sets typically offer smaller numerical error and better stability at a computational resource cost;

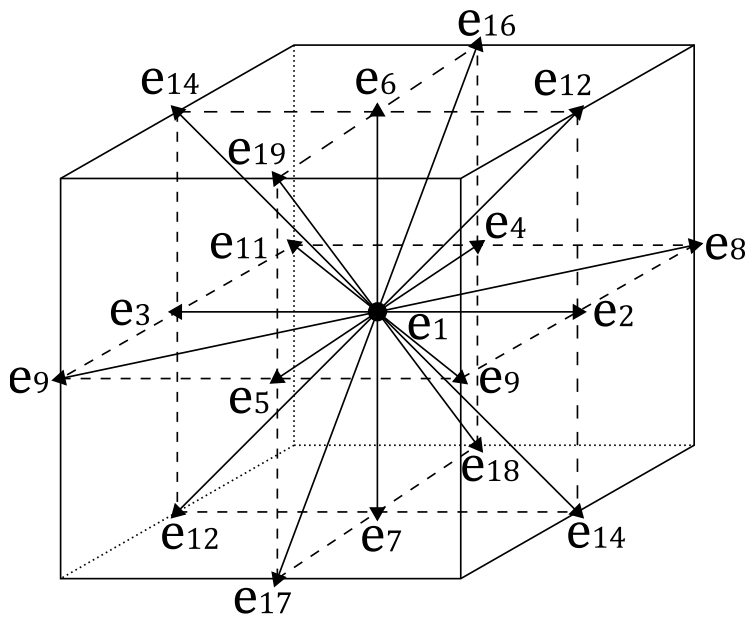


Figure 1.14: The D3Q19 stencil is depicted. Each arrow represents a lattice velocity and depicts the connectivity to the neighbouring grid node. Note that $\mathbf{e}_1$ is located at the centre with length zero, representing particles at rest.

D3Q19 was chosen as it was more accurate than D3Q15, but D3Q27 was not considered due to requiring 40% more memory for negligible benefits [86].

For each of the velocities e_{1-19} there is a floating point number f_{1-19} respectively. These numbers represent the fraction of particles moving with that velocity. For the sake of example, grid-cell size Δx and time-step Δt are normalised to 1. Then, for the D3Q19 stencil, f_1 is the fraction of particles at rest, f_{2-7} the fraction of particles moving at speed 1, and f_{8-19} the fraction of particles moving at speed $\sqrt{2}$, all in their respective directions.

The LBM algorithm has two main steps, streaming and collision. During the streaming step, all the distribution functions (excepting f_1) are advected with their respective velocities. For an example case with unitary time-step and cell size, this results in all the floating point values moving to their neighbouring cells. This streaming step can be expressed as follows for the i^{th} direction where the $'$ superscript denotes the post-streaming values.

$$f_i^{'}(\mathbf{x}, t + \Delta t) = f_i(\mathbf{x}, \mathbf{e}_i, t) \tag{1.25}$$

The collision step then takes place by weighting the distribution functions of a cell with the equilibrium distribution function, f^{eq}. The equilibrium

distribution function is dependent on the macroscopic density and velocity of the fluid, which are obtained by summing the distributions and the velocity-weighted distributions respectively, as shown in Equations 1.26 and 1.27.

$$\rho = \sum_i f_i \tag{1.26}$$

$$\mathbf{u} = \sum_i \mathbf{e}_i f_i \tag{1.27}$$

For a given direction i, the equilibrium distribution function is given by Equation 1.28 [82].

$$f_i^{eq} = w_i \left(\rho + 3\mathbf{e}_i \cdot \mathbf{u} - \frac{3}{2}\mathbf{u}^2 + \frac{9}{2}(\mathbf{e}_i \cdot \mathbf{u})^2 \right)$$

where:

$$\begin{aligned} w_i &= \frac{1}{3} \quad \text{for} \quad i = 1, \\ w_i &= \frac{1}{18} \quad \text{for} \quad 2 \leq i \leq 7, \\ w_i &= \frac{1}{36} \quad \text{for} \quad 8 \leq i \leq 19. \end{aligned} \tag{1.28}$$

Thus the collision stage is defined by Equation 1.29, where each f_i is linearly relaxed towards their equilibrium state using the f_i^{eq} values. This step constitutes the simplified collision term mentioned earlier for the particular method used in this work. Other such simplified collision operators exist, perhaps most commonly the Bhatnagar-Gross-Krook (BGK) operator [87].

$$f_i(\mathbf{x}, t + \Delta t) = (1 - \omega) f_i^{'}(\mathbf{x}, t + \Delta t) + \omega f_i^{eq} \tag{1.29}$$

where ω is a parameter relating to the viscosity of the fluid according to Equation 1.30, where ν is the kinematic viscosity of the fluid in LBM lattice units [88].

$$\omega = \frac{1}{\nu} + 0.5 \tag{1.30}$$

The parameter ω must fall in the range of $0 \leq \omega \leq 2$, where 0 is highly

viscous and 2 highly inviscid.

The further details of the theory underpinning the specific solver model used for this work is further expanded on by Thürey et al. [82], including the no-slip boundary condition, handling of the free surface interface, free surface boundary conditions, and adaptive time-stepping. Further reading on the broader theory of the LBM solver used for this work is available from [89, 90, 88, 91].

1.5.3 The case for lattice Boltzmann for the modelling of planing hulls

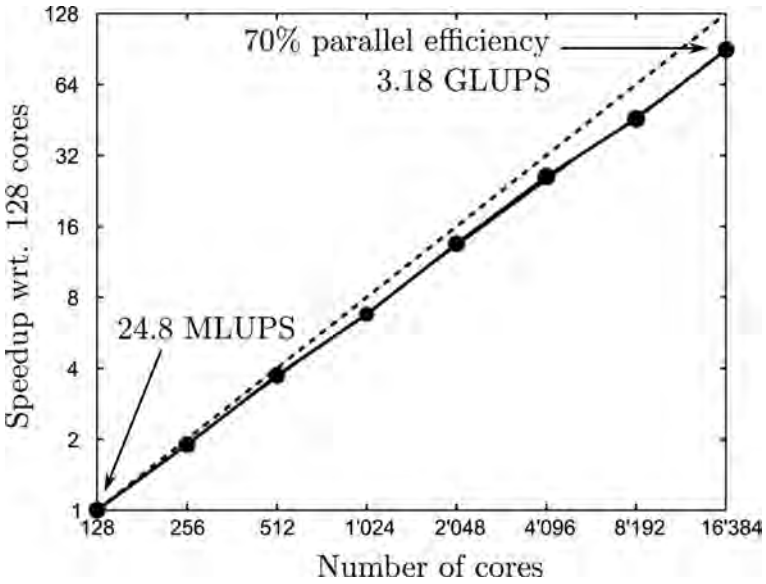
As noted in the literature review of existing solvers, there exists a number of solvers available to industry for the solving of planing hull problems, none without some drawbacks. The LBM offers a number of distinct advantages that made it the choice of method for this work.

Comparisons between LBM and more traditional finite volume/element solvers have been made in the literature [92, 93], with a general consensus that LBM can offer a simpler algorithm that can be more computationally efficient for given scenarios at the cost of higher memory use. This arises from, recalling the previous section and using a D3Q19 model for example, using an array of single precision floating point numbers of size $(2 \times size_x \times size_y \times size_z \times 19)$ [82]. In contrast, a N-S solver might only require $[size_x \times size_y \times size_z \times 7]$ floating point values. However, a N-S solver may require a higher resolution to capture complex geometries to the same fidelity.

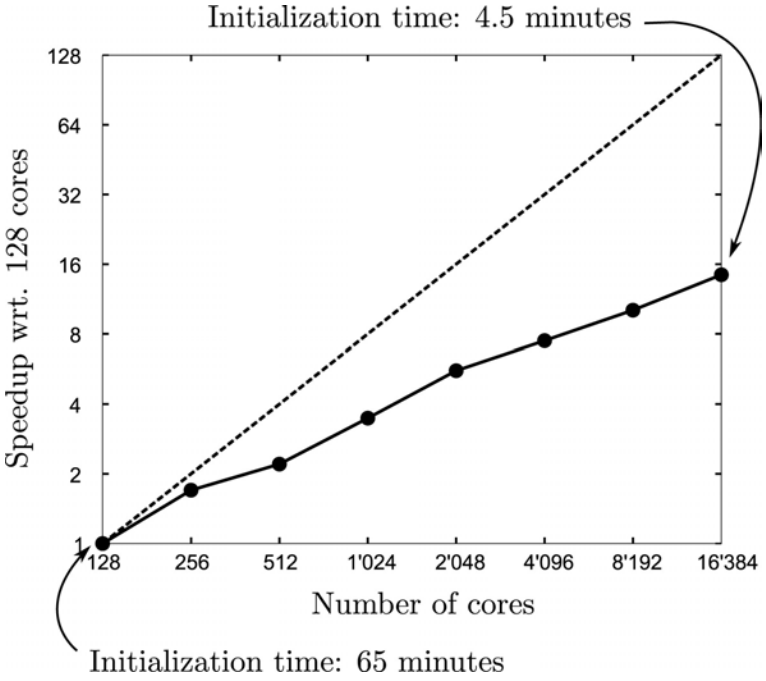
The simplicity of the underlying algorithm of LBM makes the implementation of boundary conditions simple [94]. Free surface flows, especially around a planing hullform, are defined by complex and time-dependent interface geometry, for which LBM is inherently well-equipped to handle.

Computational efficiency on massively parallel computer architectures was also a considerable draw. It was known that High Performance Computing (HPC) resources would be available for this work (see the following section, Section 1.6, for details), and LBM has been shown to work very efficiently on massively parallel HPC computer architectures. This is in part due to most

LBM solvers being specifically designed with such hardware in mind, but also arises naturally from the very efficient domain decomposition. Because each lattice grid-cell is influenced only by its neighbouring grid-cells, decomposition of the domain can be performed with less communication overhead than is seen in finite volume methods. This parallel efficiency is demonstrated in Figure 1.15, where a benchmarked case of flow in an artery using a BGK collision operator is run on an increasing number of parallel HPC cores [89]. The solver in question (Palabos) exhibits excellent parallel efficiency of 70% up to 16,384 cores on a Blue-Gene/P supercomputer. The example in Figure 1.15 is broken down into the main execution and the pre-processing to highlight another benefit of LBM, which is avoidance of the pre-processing bottleneck. Although other solvers may be parallelised, it is not uncommon for them to require a non-parallel or weakly parallel pre-processing stage, for example in mesh generation. For LBM solvers such as Palabos, it is easy to implement fully parallelised pre-processing. This is due to no body-fitted meshes of complex geometries being required in LBM; instead, LBM uses a homogeneous grid and handles the data transfer between the grid and the boundary of the geometry during runtime with appropriate interpolations [95]. Body-fitted grids are, however, among the best ways for enforce the no-slip conditions at the geometry surface. In this work an Immersed Boundary (IB) method is used, which has been found to require, for example, 50 lattice nodes to resolve a flat-plate boundary layer in a $Re = 1,000$ flow to within 5% accuracy of the analytical solution [96].



(a) The speed-up curve for the execution of flow in an artery using a BGK collision operator, excluding pre-processing. Note the acronyms, MLUPS (Mega/Million site Updates Per Second) and GLUPS (Giga/billion site Updates Per Second).



(b) The speed-up curve for the pre-processing of flow in an artery using a BGK collision operator. Although not as strongly scaling as the execution of the main solution, this pre-processing stage constitutes a small fraction of overall run time.

Figure 1.15: Palabos speed-up curves on a Blue-Gene/P supercomputer up to 16,384 cores [89].

1.6 High-performance computing resources

To take full advantage of the parallelisation efficiencies of LBM, the computer simulations presented in this paper were conducted on a High-Performance Computing (HPC) cluster. The label of HPC is given to computer systems that have been aggregated to allow for faster computation of distributable tasks.

For this work computing capacity was provided by Supercomputing Wales (SCW), a project providing HPC resources to scientific and innovation projects within the SCW consortium of universities [97].

The Swansea University branch of this project is the Sunbird HPC system, a large cluster of computing nodes based in Swansea's Dylan Thomas Data Centre available remotely to Swansea and Aberystwyth SCW users. The system specifications are as follows [97]:

- 126 nodes, totalling 5,040 cores, 48.384TB total memory:
  - CPU: 2 x Intel®Xeon®Gold 6148 CPU @ 2.40GHz with 20 cores each.
  - RAM: 384GB per node.
  - GPU: 8 x Nvidia V100 GPUs.
- Storage:
  - 808TB (usable) scratch space on a Lustre filesystem.
  - 231TB of home directory space on a Lustre filesystem.

It should be noted that the simulations in this paper are exclusively run on CPUs rather than GPUs. The maximum limits of use per user on this cluster define the effective limit of the resources available. User limits for this project are as follows:

- Maximum number of cores requestable: 1,024.
- Maximum RAM per CPU: 9575MB.
- Maximum storage: 100GB user directory, plus temporary scratch space.

Chapter 2

Boat hullform solver

2.1 Palabos

The solver selected to conduct this work was Palabos (**Pa**rallel **La**ttice **Bo**ltzmann **S**olver), a CFD solver with a kernel based on the Lattice Boltzmann method [98]. Palabos was developed at Université de Genève and is available under the open source terms of an AGPLv3 license. The Palabos library's native programming interface is written in C++, and is only dependent on Posix and MPI, both common standards for Application Programming Interfacing (API) and Message Passing Interfacing (MPI) respectively.

At the beginning of this project, the commercial aspects of Palabos were managed by FlowKit Ltd, who have since been acquired by NUMECA International [99]. The basis of the hullform solver environment was developed in partnership with FlowKit, and is expounded on in the following section.

2.2 Boat hullform solver

The aim of this part of the project was to develop a general and reusable free surface flow hullform modelling environment using the Palabos library. Regardless of the specifics of the analysis of a given hullform, many parts of the simulation will remain the same: for example, a hull geometry will be imported, flow will need to move relative to the hull at a given inlet velocity, domain bounds will be set, and boundary conditions will be prescribed. To do

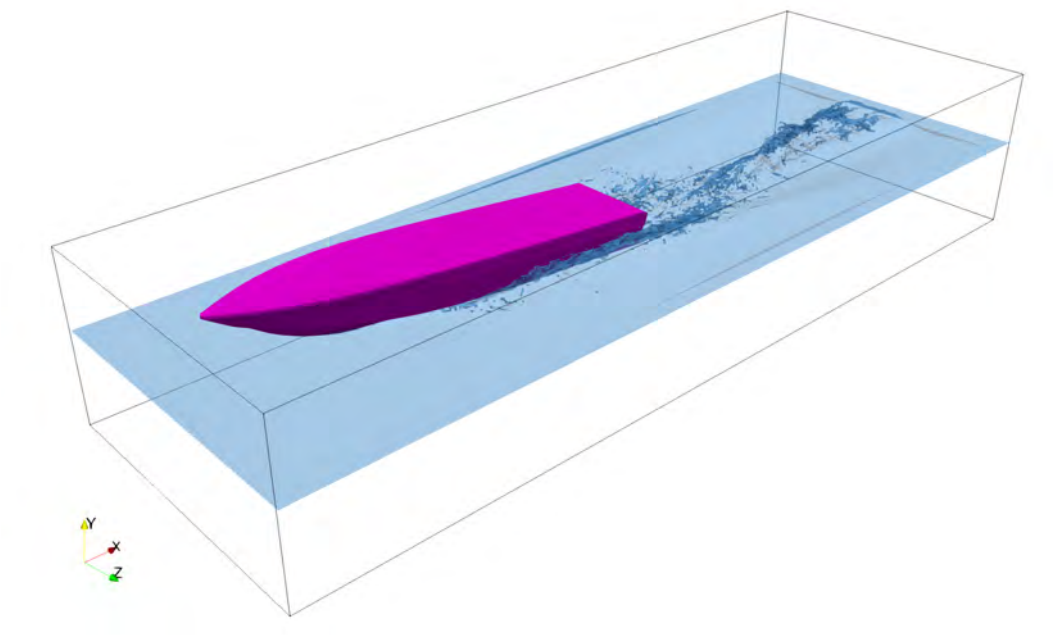


Figure 2.1: The virtual tow tank domain (black outline), with free surface interface (blue) and an example hullform (pink).

this, a reusable, general executable written in C++ that calls on the Palabos library was created to set up and run this virtual flume simulation. The source code for `boatHullFormSolver.cpp` can be found in Appendix A.3.1.

The boatHullFormSolver code virtually replicates a tow tank testing environment: water flows in the x-direction past a semi-submerged hullform defined by the user, with inlet and outlet conditions upstream and downstream respectively. Gravity acts in the negative y-direction. Pressure is projected from lattice nodes adjacent to the hull surface on to the hull geometry. This general configuration is shown in Figure 2.1.

The simulation parameters are configured in the input `.xml` file. The parameters configurable in this file are as follows:

- **Domain dimensions** – this is defined by a minimum and maximum value in x, y and z dimensions.
- **Absorbing zone dimensions** – the sizing of the numerical absorbing zones, described in detail later.
- **Fluid height** – the height of the fluid measured from the minimum y-axis domain limit.

- **Fluid density** (ρ), **kinematic viscosity** (ν), and **surface tension coefficient** (σ) – physical fluid properties.
- **Hullform geometry** – the path and filename of the hullform geometry provided in `.stl` format.
- **Geometry inflation parameter** – a parameter for a tool used to improve geometry import robustness by inflating the geometry to eliminate gaps or holes.
- **Characteristic length** (L_{char}) – the reference length, used to indirectly determine grid size (see Equation 2.1) and Reynolds number.
- **Resolution** – a dimensionless parameter that indirectly determines grid size (see Equation 2.1).
- **Inlet velocity of fluid** (u_{inlet}) – the velocity of the fluid at the inlet boundary condition is prescribed.
- **Reference velocity** (u_{ref}) – used to define the time-step (see Equation 2.2), always set to match the inlet velocity at the recommendation of FlowKit.
- **Lattice velocity** (u_{LB}) – this value determines the time-step (see Equation 2.2). It was recommended by FlowKit to kept this value small (0.01) to avoid numerical instability and compressibility error.
- **Maximum number of iterations** – this determines the simulation length.
- **Checkpointing options** – the user can request that the simulation "checkpoints" after a given number of iterations, saving the state of the simulation to allow it to be restarted at a later time.

Note that gridsize and time-step of each simulation is determined by combinations of some of the above parameters as described in Equation 2.1 and 2.2 respectively [82].

$$dx = \frac{L_{char}}{(resolution - 1)} \tag{2.1}$$

$$dt = \frac{u_{LB} dx}{u_{ref}} \tag{2.2}$$

where dx is the spatial discretisation of the lattice grid, $L_{characteristic}$ is the characteristic length (conventionally taken to be boat beam), dt is the simulation time-step, u_{LB} is the lattice velocity, and u_{ref} is the reference velocity.

The simplicity of editing this `.xml` file to make adjustments to the general simulation setup eliminates the need to edit and compile source code for each simulation.

2.2.1 Solver physics

The boatHullFormSolver uses Palabos' free-surface model [82] on a D3Q19 (3 dimensions, 19 velocity vectors) lattice. The D3Q19 velocity set was chosen as a compromise between the less accurate D3Q15 and more memory intensive D3Q27 set [83]. In this model, the fluid phase (in this application the water phase) is fully simulated, while the gas phase (air) is represented with a constant pressure term set to simulate atmospheric pressure.

Surface tension is implemented by use of finite-difference stencils which evaluate the local curvature of the free surface from a filtered volume-fraction of interface nodes [82]. This algorithm is based on the Young-Laplace equation, shown in Equation 2.3.

$$\Delta p = -\sigma \nabla \cdot \mathbf{n} \tag{2.3}$$

where Δp is the difference in pressure across the interface, σ is the surface tension coefficient, and n is the normal vector perpendicular to the interface in the direction towards the gas phase and away from the water phase. Divergence of the unit normal vector relates to the local mean curvature of the free-surface, which is calculated from finite difference stencils that operate on a filtered volume-fraction field.

A Volume of Fluid (VoF) approach [82] is used to track the free surface boundary of the single-phase flow. Turbulence is modelled using a Large Eddy Simulation (LES) model [78]. The LES occupies a middle-ground solution between Direct Numerical Simulation (DNS), where length and time scales are fully resolved to the minute level at which turbulence can occur, and pure

phenomenological/empirical models of subgrid turbulence such as k-epsilon or Spalart-Allmaras which tend to have specific use-cases. The LES achieves this by low-pass filtering of the fluid solution, which essentially spatially and temporally averages the effects of the small-scale turbulent effects.

A classic static Smagorinsky LES model is used in which subgrid scales influence the viscosity proportionally to the norm of the strain-rate tensor for the filtered scales, as expressed in Equation 2.4 [100].

$$\nu = \nu_0 + \nu_T \tag{2.4}$$

Where ν is the total kinematic viscosity, ν_0 is the molecular-scale viscosity (i.e., the fluid property viscosity), and ν_T is the turbulent viscosity correction given by Equation 2.5.

$$\nu_T = C^2|S| \tag{2.5}$$

Where C is the Smagorinsky constant and $|S|$ the strain rate tensor-norm such that $|S| = \sqrt{S : S}$. The value of the Smagorinsky constant typically varies from 0.1 to 0.2 and is problem-dependent; in all cases in this work the constant is kept at 0.11 on advice from FlowKit based on experience with similar hydrodynamic simulations. This constant is independent with respect to time, thus the model is "static". No explicit wall modelling is made in the Smagorinsky LES model used in this work, which is a notable weakness in the method. One method of implementing wall treatment lies in varying the Smagorinsky constant, C, depending on spatial proximity to a wall, which is permitted by local nature of the dynamics class `SmagorinskyDynamics` [100]. Although desirable, this could not be achieved in the scope of the work and the implications are discussed in Chapter 5.

2.2.2 Boundary conditions

Inlet and outlet conditions are applied upstream and downstream respectively in the x-direction; flow is intended to always travel in the positive x-direction. Periodic boundary conditions are enforced in the z-direction. A

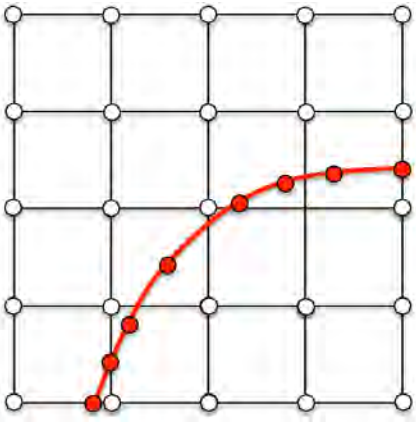


Figure 2.2: A 2-dimensional example of a immersed boundary method implementation, as depicted by De Rosis [95]. The unstructured Lagrangian mesh of the immersed body (red) is shown in relation to the Eulerian fluid grid (black).

free-slip boundary condition is applied to the bottom of the domain. A no-slip wall condition is applied to the "ceiling" of the domain.

An Immersed Boundary (IB) condition as described by De Rosis et al. [95] is applied at the hullform geometry. The IB condition is used as opposed to a simple no-slip condition to allow for future expansion to Fluid Structure Interaction (FSI) capabilities, as the IB can enforce a no-slip condition, satisfy Newton's law, and conserve mass. The IB method takes the Lagrangian unstructured mesh of the body, in this case the geometry `.stl` triangulation, and "immerses" it in the Eulerian LBM grid. This is visualised in Figure 2.2. Interaction between the fluid and the body is then determined by interpolation to enforce the no-slip rule (zero velocity) and momentum conservation [95]. The fluid phase is indirectly affected by the boundary in a manner analogous to a source term added to the LBM distribution functions.

The IB method allows fluid to exist on the "inside" of the boat hull geometry surface, typically only one lattice node from the surface. To avoid non-physical behaviour arising from excessive fluid mass inside the hull geometry, fluid nodes inside the geometry is excluded from contribution to the pressure projected by the fluid onto the hull geometry.

The IB method has been found to be accurate and robust for moving, immersed bodies in Fluid Structure Interaction (FSI) scenarios [95]. It is for this reason that the IB method was chosen over alternatives such as the

Bounce-Back method [101] and Ghost methods [102], which are less suitable for moving geometries. The IB method does, however, have drawbacks noted in the literature, namely first-order velocity accuracy [103]. This is considered to be one of the necessary compromises in selection of LBM boundary methods, and the proven suitability to the modelling of immersed, moving bodies in FSI problems cemented the choice of the IB method for this work.

2.2.3 Absorbing zones

Absorbing zones are numerical constructs which help implement the boundary conditions by "absorbing" physical quantities at the edge of the domain. It should be noted that these zones are purely artificial and do not represent physical flow phenomena. There are two types of absorbing zone: free surface sponge zones and pressure absorption zones.

Free surface sponge zones assist the implementation of the inlet and outlet boundary conditions. The sponge zones smoothly converge the volume fraction and velocity of the water phase back to the initial inlet conditions. This ensures that the disturbed flow aft of the hullform does not affect the inlet flow. Sponge zones are also applied at the "side" domain bounds to cancel out periodicity in the z-direction. The free surface sponge zone applied to the domain ceiling prevents the "wall" effect given by the no-slip wall boundary condition.

The second form of absorbing zones, the pressure absorption zones, purely dissipate pressure disturbances. These are applied within the same user-defined space as the sponge zones, and can be activated by compiling the executable with the macro WAVE_ABSORPTION defined. These zones are not always necessary, but assist in dissipating acoustic waves. Implementation is complex, and therefore using these zones increase compute time and should be avoided when not necessary.

The typical location and dimensions of these numerical absorbing zones are progressively illustrated in Figures 2.3 – 2.5.

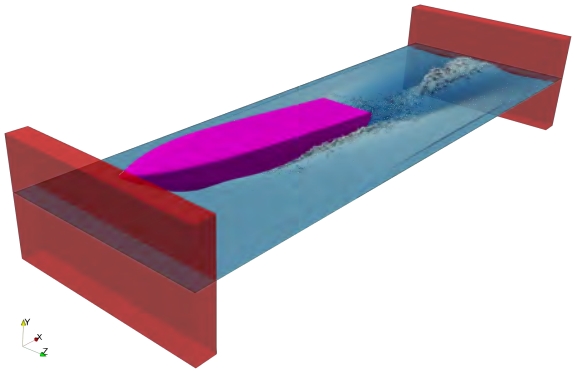


Figure 2.3: The typical position and dimensions of the inlet/outlet absorbing zone volumes are shown in red.

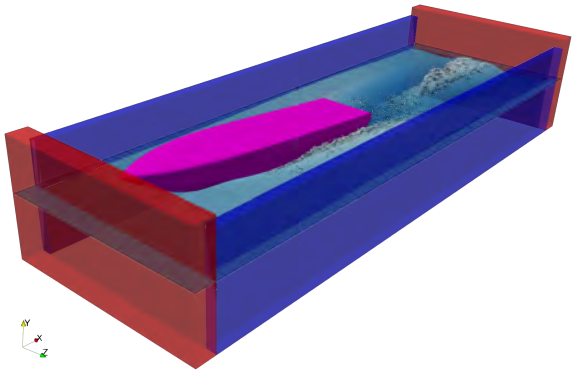


Figure 2.4: The typical position and dimensions of the lateral absorbing zone volumes are shown in blue.

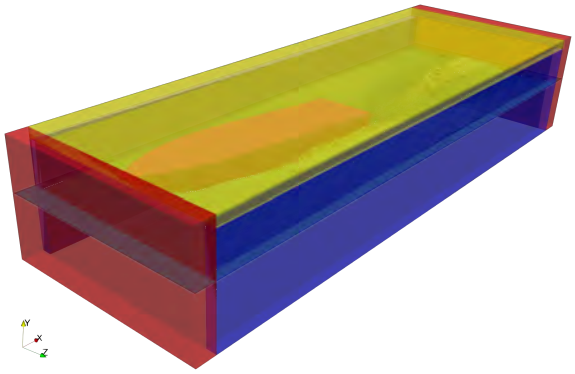


Figure 2.5: The typical position and dimensions of the ceiling absorbing zone volume is shown in yellow.

2.2.4 Wave forcing

A body forcing term can be used to induce waves in a select volume of the domain. This volume should not intersect with the artificial absorbing zones, and will typically be placed upstream of the hullform geometry to simulate a boat passing through waves. The oscillating force F acts in the y-direction (parallel to the gravity force) according to Equation 2.6.

$$F = A \cdot g \cdot sin\left(\frac{2\pi t}{P}\right) \tag{2.6}$$

where A is the wave amplitude, g is the acceleration due to gravity, t is the physical time, and P is the wave periodicity. This capability is not explored in this work, but remains for future development.

2.2.5 Checkpoint restarting

Palabos features a checkpointing system that is implemented in the boatHullFormSolver code. Checkpointing the simulation saves the current state of the simulation in a manner it can be restarted at a later point. To invoke a checkpoint, either a file named `abort` is created by the user, which begins the checkpointing process, or the user can set a fixed number of iterations at which the simulation will checkpoint automatically. Once the checkpoint files are written to the working directory, the original execution can be called with the `checkpoint.xml` file appended after the original `parameter.xml` file. This system is later used in the dynamics implementation of Section 3.3.

2.2.6 Output files

Each simulation exports an `.stl` file containing the geometry of the free surface throughout the simulation, a `.dat` file containing average kinetic energy, force components, and total force experienced by the hullform, and `.vtk` and `.vti` files containing fluid data and hull pressure distribution for post-processing visualisation.

2.3 Grid convergence study

Numerical approaches that rely on spatial discretisation of the domain using a grid will have an inherent discretisation error that is dependent on the grid properties. As the grid is refined – that is, as the size of each grid cell is made smaller – this discretisation error is reduced, theoretically approaching zero as the grid cell size approaches zero.

A simulation is said to be grid independent when the outputs are negligibly affected by variations in the grid resolution. It is important to achieve this as the grid is a non-physical artefact required by the numerical approach and should therefore not influence the physical results. Grid convergence studies are thus very common in CFD and should be conducted not only to validate a method or scheme, but to ensure that the particular case with its unique parameters is grid independent.

The approach used in this section follows advice laid out by Roache for standardised grid convergence assessment [104, 105, 106]. This method has been taken up by the American Society of Mechanical Engineers (ASME) [107] and by the American Institute of Aeronautics and Astronautics (AIAA) [108], and has been applied, with further analysis, to fluid simulations of marine craft [109]. It should be noted that criticism of the Roache approach used here points out that the measured points must be relatively close to the converged criteria in order to accurately assess it; this makes the approach less useful in cases where there is little prior knowledge the the rough level of grid resolution needed, and rather acts to confirm initial estimates. It should also be considered that due to the larger scales of turbulent flow being highly anisotropic, exploration of the anisotropy of the mesh is usually considered when assessing grid quality for LES modelling [110]. The restriction of a uniform grid in this work limits such an exploration.

The grid convergence study process is outlined as follows:

1. A set of three simulations are run with identical parameters excepting grid resolution.

2. An output from these simulations is selected to be studied for grid con-

vergence.

3. A Richardson extrapolation is performed on the results to project an estimated value at zero grid discretisation error.

4. A Grid Convergence Index (GCI) is calculated for each change in grid size.

5. A check is performed to determine whether any of the simulations lie in the asymtotic region of grid convergence.

The number of simulations conducted and the degree of grid refinement in each case was considered. Practical limitations of compute time and cluster availability had to be taken into account, resulting in three simulations – coarse, medium, and fine – being conducted. The resolution of the finest grid was determined by practicalities of the maximum amount of compute cores available for a single simulation on the Swansea Sunbird system (1024 cores, see Section 1.6). The limitation of having a globally uniform grid cell size simplifies this study as no accounting for variable grid cells needs to be performed. Exact grid data for each is given in Table 2.1; note the computational cost is quantified by the mean average time per iteration, which is discussed later.

The simulation output that is examined for convergence must be selected appropriately, as some parameters may have more or less stringent grid convergence requirements. As the goal for this work is to accurately predict dynamic equilibrium in heave for planing hulls, integrated vertical force exerted by the fluid on the hull (lift) is chosen.

Table 2.1: Grid values for each grid convergence simulation and the mean average time taken for a timestep iteration.

simulation	*resolution*	*dx* (m)	*dt* (s)	Avg. time per iteration (s)
coarse	251	0.04	6.479e-05	0.5162
medium	501	0.02	3.240e-05	1.7153
fine	1,001	0.01	1.620e-05	4.8373

Note that "resolution" is a user input for each simulation and is used to set the grid spacing (dx) as follows:

$$dx = \frac{L_{char}}{resolution - 1} \tag{2.7}$$

where L_{char} is characteristic length. Note that resolutions were chosen such that the ratio of dx between subsequent simulations was constant.

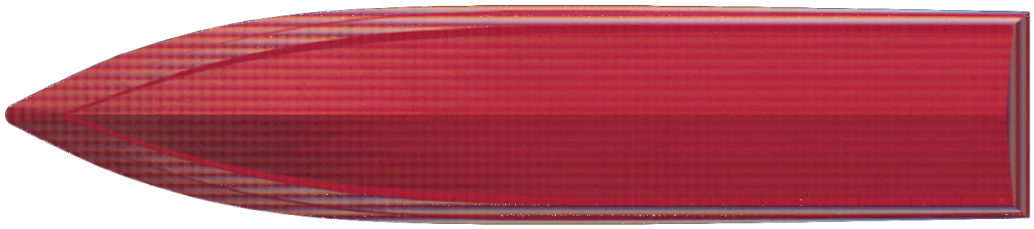
All other simulation parameters were kept constant as control parameters. These parameters are given in Table 2.2. The boat geometry chosen, a DV15 hullform, was placed in a flow speed and estimated equilibrium position provided by Norson Design (trim angle of 6.2° and draft of 900mm). Visual representations of the voxelisation of the boat geometry on the underlying grid are depicted in Figure 2.6 and 2.7.

Table 2.2: Grid convergence study control parameters.

parameter	*units*	*value*
domain size:		
x range	(m)	$[-15.0, 3.0]$
y range	(m)	$[-2.0, 2.6]$
z range	(m)	$[-8.0, 8.0]$
absorbing zone widths:		
inlet	(m)	0.5
outlet	(m)	0.5
lateral	(m)	1.0
top	(m)	0.1
fluid properties:		
density, ρ	(kg/m^3)	$1,030.0$
kinematic viscosity, ν	(m/s^2)	$1.0e-6$
fluid height	(m)	3.0
surface tension	(N/m)	0.0728
Palabos parameters:		
characteristic length, L_{char}	(m)	10.0
inlet velocity, u_{inlet}	(m/s)	6.173
reference velocity, u_{ref}	(m/s)	6.173
lattice velocity, u_{LB}	(m/s)	0.01

(a) coarse grid voxelisation, dx = 0.04m.

(b) medium grid voxelisation, dx = 0.02m.

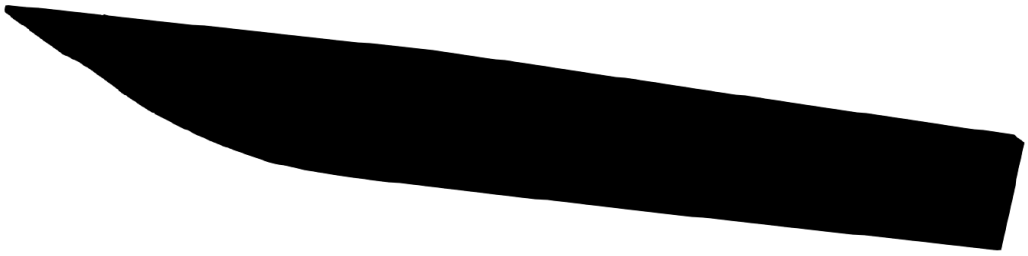


(c) fine grid voxelisation, dx = 0.01m.

Figure 2.6: Voxelisations of the boat geometry on the underlying grid seen from below for each level of grid refinement.

(a) coarse grid voxelisation, dx = 0.04m.

(b) medium grid voxelisation, dx = 0.02m.



(c) fine grid voxelisation, dx = 0.01m.

Figure 2.7: A slice down the centreline of the hull for each level of grid refinement shows the voxelisation of the hull geometry on the underlying grid.

The lift results of these three simulations are shown in Table 2.3. In this table the grid spacing is normalised by the finest grid.

Table 2.3: Grid convergence simulation results.

simulation	*dx* (m)	*normalised grid size*	*integrated vertical force (N)*
coarse	0.04	$h_3 = 4$	$f_3 = 107{,}440$
medium	0.02	$h_2 = 2$	$f_2 = 75{,}291$
fine	0.01	$h_1 = 1$	$f_1 = 61{,}790$

A Richardson extrapolation was carried out on the values obtained from the three simulations to evaluate an estimate of the continuum value (i.e.: the theoretical value at zero grid spacing). To do this the order of convergence, p, was first determined by:

$$p = \ln\left(\frac{f_3 - f_2}{f_2 - f_1}\right) / \ln(r) \tag{2.8}$$

where f_{1-3} are the lift results of each case and r is the ratio of grid refinement, given by:

$$r = \frac{h_2}{h_1} = \frac{h_3}{h_2} = 2 \tag{2.9}$$

Thus substituting this r value and lift results into Equation 2.8, the order of convergence is evaluated as:

$$p = \ln\left(\frac{107,440 - 71,415}{75,291 - 61,790}\right) / \ln(2) = 1.90 \tag{2.10}$$

The Richardson extrapolation for a three-simulation case was then performed as follows:

$$f_{h=0} = f_3 + \frac{f_1 - f_2}{r^p - 1} = 61,790 + \frac{61,790 - 75,291}{2^{1.90} - 1} = 58,267 \tag{2.11}$$

The trend of the progressively refined simulations approaching this projected value is illustrated in Figure 2.8.

Although Figure 2.8 gives a visual indication of convergence, numerically

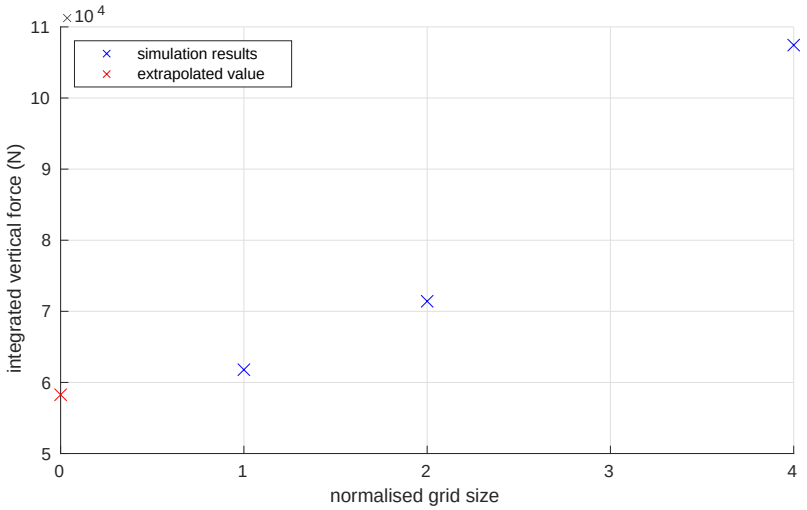


Figure 2.8: The trend towards the extrapolated value is shown in context of the grid convergence simulations.

defining the degree of convergence allows for a more rigorous assessment of whether acceptable convergence has been achieved. Continuing with the standardised structure of a grid convergence study [104], Grid Convergence Indices (GCIs) were calculated for each step in grid refinement as follows:

$$GCI = \frac{F_s|e|}{r^p - 1} \tag{2.12}$$

where e is the error (difference) between adjacent grid resolutions and F_s is a safety factor. The factor of safety for a three grid study is recommended to be 1.25 by Roache [106]. The GCIs for grid spacing reduction from h_3 to h_2 ($GCI_{2,3}$) and from h_2 to h_1 ($GCI_{1,2}$) are acquired by:

$$GCI_{2,3} = 1.25\left|\frac{75,291 - 107,440}{75,291}\right|/(2^{1.90} - 1) = 0.231 = 23.1\% \tag{2.13}$$

$$GCI_{1,2} = 1.25\left|\frac{61,790 - 75,291}{61,790}\right|/(2^{1.78} - 1) = 0.071 = 7.1\% \tag{2.14}$$

The GCI is the standardised numerical value to quantify grid convergence quality. For a three grid study the following relationship determines whether solutions lie in the asymptotic region of grid convergence, where solutions very

close to 1 are considered converged:

$$\frac{GCI_{2,3}}{r^p GCI_{1,2}} \tag{2.15}$$

In this case,

$$\frac{GCI_{2,3}}{r^p GCI_{1,2}} = \frac{0.231}{2^{1.90}0.071} = 0.872 = 87.2\% \tag{2.16}$$

Although this value does not put solutions within the desirable 95% confidence region, it does show reasonable convergence to grid independence on which to develop capabilities with the assumption that better grid resolution will be achievable in the future. This could be done with either an increase in the computational resources available, or – more desirably – with local grid refinement capabilities being added to the solver in future work.

As the uniform grid restriction stands, the impact on grid refinement on computational cost must be observed. As is seen in Table 2.1, the timestep (driven by Equation 2.2) is reduced as grid resolution increases, but so does the average time taken per timestep iteration. The four times increase in resolution between the coarse and fine simulations results in a 19,346% increase in compute time taken to simulate the same physical time. This is clearly a significant cost increase that affects the viability of simply scaling the available compute power to the problem or grid resolution.

2.4 Experimental validation – SB90E flume experiments

A set of experiments were performed using a flume in the Swansea University Civil Engineering Laboratory for the purpose of validating the ability of Palabos to model free surface fluids in the context of a surface vessel. At the time of this experiment, it was thought that a full scale Storebro Stridsbåt Enkel (SB90E) would be available for gathering sea trial data later in the project, and so validation of this specific geometry was determined to be useful. This was not the case, but the SB90E (see Section 2.4.2 for details) is typical of a planing hullform and thus appropriate for validation purposes.

2.4.1 Experimental apparatus

The Swansea University Civil Engineering laboratory granted access to an Armfield S6 MkII flume (see Figure 2.9). This flume has a $300mm$ wide channel capable of a 32 litres per second volume flow rate through a closed-loop system, with an adjustable ramp at the outlet to alter the flow height throughout the channel. The $300mm$ width restriction of the flume impinged on the wake of the model, which would otherwise be free to propagate in open-water conditions. This was accounted for by also modelling this channel width restriction in the equivalent computational simulations. Computational simulations were also made at a 1:1 scale to avoid scaling.

A Stratasys Objet 1000 Plus 3D printer was used for fabrication of the scale model (Figure 2.12). Use of a 3D printer conferred the advantage of quickly fabricating a model that is accurate to the same CAD geometry used for the numerical simulations. The size restriction of the Objet 1000 Plus build tray ($1000 \times 800 \times 500mm$) easily exceeded the limitations imposed by the width of the Armfield S6 MkII flume.

A previous experiment on validation of surfboard fin simulations conducted in the Armfield S6 MkII flume by Dr David Carswell left a custom-built rig for the flume [111]. This rigging allowed for isolating movement in principle directions, or could be completely clamped to allow for static model tests.

Figure 2.9: The Armfield S6 MkII tilting flume in the Swansea University Civil Engineering Laboratory.

The rigging is shown in Figure 2.10.

A Digitron 2022P manometer was provided for pressure measurements. This 2 port differential manometer has a maximum pressure capability of 2 bar and accuracy of 0.15%.

2.4.2 The 3D-printed SB90E scale model

The scale model used was of a Storebro Stridsbåt Enkel (SB90E), which was designed as a fast littoral combat craft for the Swedish Navy. A 3D CAD model of the SB90E was created based on technical specifications and reference photographs of the vessel [112]. The CAD model is shown in Figure 2.11 alongside photos of the full-scale boat for reference.

The printed model was made to a 1:25 scaling ratio ($432mm$ in length versus the $10.8m$ vessel) with the width restriction of the flume in mind. This granted a $152mm$ beam, making the choke ratio for the $300mm$ wide channel 0.51. The model was printed as a single piece of Stratasys Vero White RGD825 photopolymer material. Due to the porous nature of this material and the need to be partially submerged for extended periods of time, a water

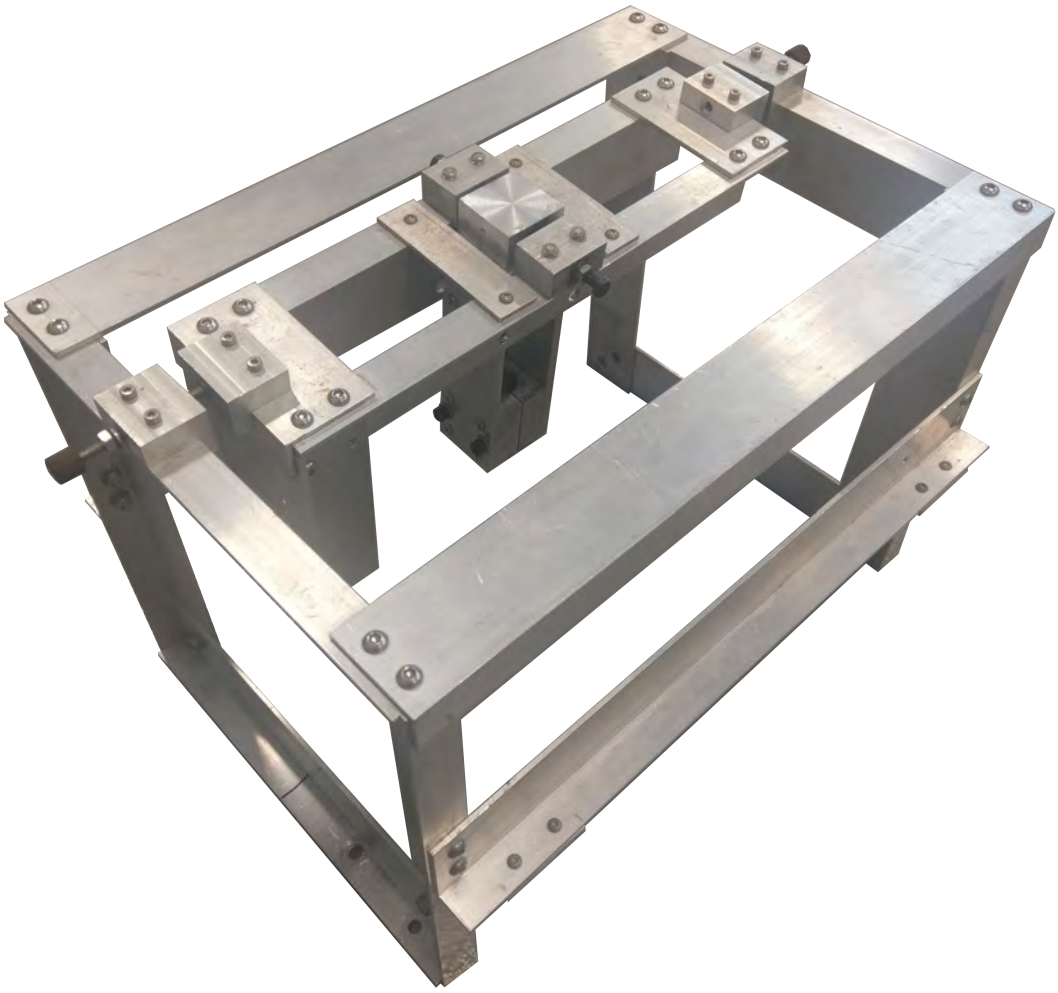


Figure 2.10: The custom-built flume rigging from Dr David Carswell's experiments [111], fabricated by Mr. Graham Foster.

resistant coating was required. A CopperCoat anti-fouling coating was applied under the double chine by local marine services company Wray Marine, with marine-grade paint applied on the remaining surfaces. The uncoated and coated model is shown in Figure 2.12.

2.4.3 Acquiring pressure readings

It was determined that the most appropriate variable to measure in a flume validation experiment would be pressure experienced at specific points on the submerged portion of a scale boat model, rather than surge, heave, or sway forces. This was due to the pressure profile being the more fundamental variable in the sense that total heave or surge force is derived from the pressure distribution and may match computational results only due to incorrect local

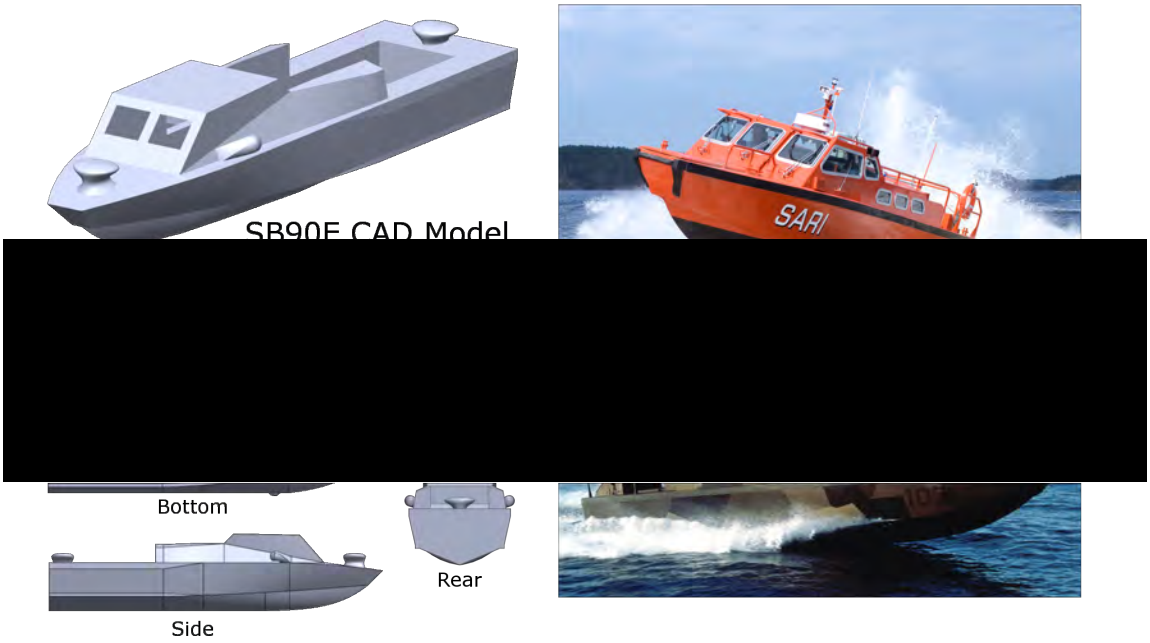


Figure 2.11: Multiple views of the SB90E CAD geometry and reference photos of the SB90E, courtesy of Storebro [112].

pressure differences balancing one another out. By measuring pressure as opposed to forces, this validation provided more supplementary support to the prismatic validation of Section 2.5, which only considered overall forces and wetted lengths. Further, previous experimentation conducted by Dr Carswell in the same Armfield S6 MkII flume experienced difficulties in measuring forces due to vibrations induced by the tank that distorted the load cell readings [111].

To measure the pressure on the hull of the boat model, a number of pressure tappings were drilled into the model perpendicular to the hull surface. Into these holes were placed brass tubes with an internal bore diameter of $0.5mm$, which were filed flush with the hull on the end that protruded from the principal wetted area. These tappings are shown in Figure 2.13.

The other end of these tubes protruded into the "cabin" of the model, where $1.85mm$ bore PVC tubing could be firmly attached. This tubing was then run along the rod connecting the model to the flume rigging, and connected to the Digitron 2022P differential manometer (Appendix A.1). To

Figure 2.12: The SB90E 1:25 scale model as retrieved from the printer work tray (left), and the coated model mounted on a stand (right).

ensure the pressure at the hull surface was measured correctly it was required that the tubing be filled entirely with degassed water with no air bubbles permitted to enter the tubing. To do this, water was pushed through the tubing using a syringe. To ensure no air entered the tubing when tubes were changed over on the manometer connection, a water bath was required so that the changing of tubes on the manometer could occur underwater. Further, the locations of the pressure tappings on the hull had to be chosen such that at no point are they above the water surface, allowing air bubbles to enter the tubing. Not only did this require careful placement of the tapping points on the hull, but also meant care had to be taken such that flow conditions did not allow bow waves to dip under these points on the hull.

2.4.4 Experimental procedure

The model was suspended in the flume from the custom rigging, and the flume was turned on and adjusted to the desired (maximum) flow rate. Flow speed readings were taken, indicating an inlet speed of $0.76m/s$. The end-ramp of the flume was then adjusted such that the water level reached a representative level on the boat model, which was observed to remain steady relative to the boat throughout the experiment. Flow conditions in the tank were such that small waves were generated along the length of the tank, though these waves were small relative to the size of the model, and the profile of the flow was

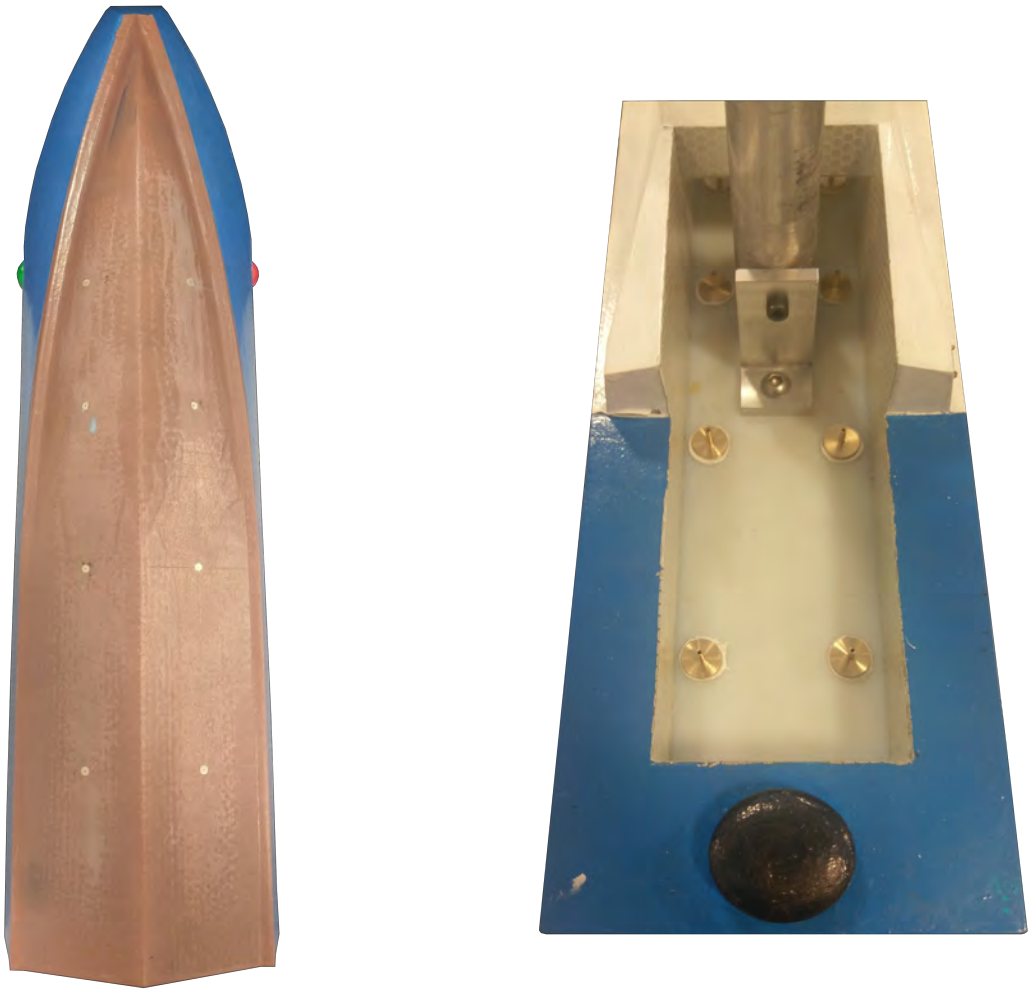


Figure 2.13: The location of the pressure tappings on the underside of the hull are shown (left), along with the tube attachment points on the other end, inside the boat "cabin" (right).

steady over the duration. To reduce these adverse factors, all measurements were taken over a 60 second time period to average out fluctuations.

The manometer was zeroed to atmospheric pressure. The Digitron 2022P differential manometer featured two pressure ports, marked positive and negative: the tubes from the SB90E model were connected to the positive port, while the negative port was connected to a tube that was vented to atmospheric pressure (doing so prevented water from coming in contact with the negative port pressure membrane while the positive port tube changeover occurred underwater). Thus the pressure read on the 2022P7 digital display will be the total pressure on the hull surface plus the gravitational head pressure induced by the height difference between the pressure tapping on the hull and

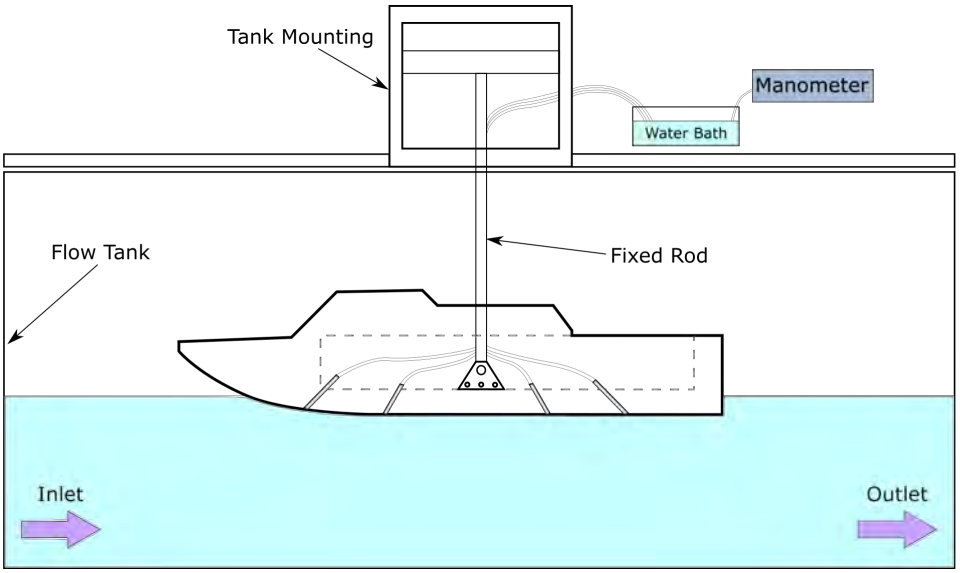


Figure 2.14: A diagram of the experimental flume set-up is shown (not to scale).

the position the manometer was held at, calculated as shown in Equation 2.17.

$$\text{gravitational head} = \rho g h \tag{2.17}$$

where ρ is the fluid density ($1,000 kg/m^3$), g is acceleration due to gravity ($9.81 m/s$), and h is the vertical displacement between the pressure tapping on the hull and the manometer membrane. The manometer was placed in the same fixed position relative to the tank for the duration of the 60 second recording period for every measurement.

Once the manometer was zeroed, the first tube from the boat model was submerged in the water bath and water was forced through the tubing with a syringe until the entire tube was filled. When the tube line was checked and found to be clear of any air bubbles, the tube was connected to the positive port on the manometer while submerged in the water bath. The manometer was then placed in its fixed position relative to the tank, and the 60 second recording period began. The Digitron 2022P is capable of recording the minimum, maximum, and average pressure recorded during this time, all of which were logged. Once the 60 second period was over, the manometer was disconnected, restarted, and re-zeroed to atmospheric pressure. The procedure was then repeated for the other pressure tappings, the numbering convention

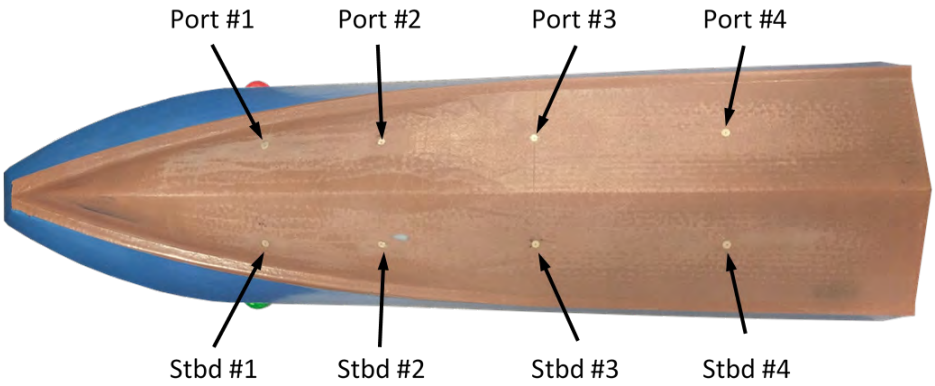


Figure 2.15: The numbering convention for the hull pressure tappings on both port and starboard (stbd) sides.

of which can be seen in Figure 2.15. The results of the experiment described can be seen in Figure 2.16, where they are compared with CFD values.

2.4.5 Palabos numerical simulations matching the SB90E flume conditions

A Palabos simulation was run at conditions matching the experiment. The geometry used to print the experimental model was the same model used for the CFD simulation. The inlet speed used was calculated using the volume flow rate and cross sectional area of the flow in the flume. This approximation gave an inlet speed of $0.76m/s$ based on the known velocity profile in the flume as recorded previously by Dr David Carswell [111].

The simulation ran on the Swansea University Astute HPC Cluster, simulating 11.9 seconds of physical time. The run parameters are detailed in Table 2.4, and the results are detailed in the following section.

Table 2.4: SB90E validation simulation parameters.

parameter	*units*	*value*
domain size:		
x range	(m)	$[-0.30, 1.00]$
y range	(m)	$[-0.05, 0.15]$
z range	(m)	$[-0.17, 0.17]$
absorbing zone widths:		
inlet	(m)	0.10
outlet	(m)	0.10
lateral	(m)	0.02
top	(m)	0.00
fluid properties:		
fluid inlet velocity	(m/s)	0.76
density, ρ	(kg/m^3)	$1,000$
kinematic viscosity, ν	(m/s^2)	$1.0e-6$
fluid height	(m)	0.07
surface tension	(N/m)	0.0728
Palabos parameters:		
resolution	$(-)$	401
dx	(m)	$1.25e-3$
dt	(s)	$1.64474e-5$
characteristic length, L_{char}	(m)	0.5
lattice velocity, u_{LB}	(m/s)	0.01
computational resources:		
HPC cluster used	$(-)$	Swansea Astute cluster
number of CPU cores	$(-)$	500
memory per CPU core	(Mb)	1024
total wall clock time	$(-)$	$6d:22h:10m:01s$
solver iterations	$(-)$	$857,696$
solver time	$(-)$	$105h:16m:48s$

2.4.6 Comparison of experimental and CFD results

The experimental results are plotted in Figure 2.16 alongside the CFD experiments. The tapping position numbers refer to the positions shown in Figure 2.15. The CFD values are consistently lower than the experimental values, by approximately $100Pa$. An explanation for this is that the crude estimation of flow velocity in the tank is under-predicting the flow speed experienced in the portion of the tank occupied by the model. The volume flow rate method of calculating flow velocity assumes a uniform velocity profile across the cross-section of the flume tank, which is not the case for a viscous fluid: the flow will be faster further from the walls, and slower closer to the walls. Another potential cause for the consistent $100Pa$ difference between CFD and experimental pressures is inaccuracies in the measurement of the head height between manometer port and boat pressure tapping ports: given by h in Equation 2.17, a $100Pa$ discrepancy would only require an error in measurement of head height by $1cm$. This is demonstrated in Equation 2.18. Due to parallax issues in measurement of this distance, this is the most likely systematic source for such a discrepancy.

$$\frac{p[Pa]}{\rho[kg/m^3]g[m/s^2]} = \frac{100[Pa]}{1,000[kg/m^3] \times 9.81[m/s^2]} = 0.010[m] \tag{2.18}$$

Also visible in Figure 2.16 is the pressure at each point along the hull varying differently on each side of the boat, as well as for the mode of simulation (CFD and experimental). These changes in the experimental case all fall well within the error bands plotted. A likely cause of this is that the pressure tapping locations have been chosen at positions on the boat hull which – according to CFD results – should experience similar pressures. This can be seen in Figure 2.17, where the pressure distribution on the hull is shown time-averaged over the $11.9s$ period.

The pressure profile in Figure 2.17 shows trends that are dominated by the static pressure, i.e. the pressure correlates strongly with depth. This is to be expected, as the limitation in flume flow speed and zero trim angle

position mean the boat is in a displacement position rather than planing. This is further reinforced by the higher pressure stagnation seen at the front of the hull, dropping off along the length of the hull: this is consistent with displacement hull behaviour. A planing pressure profile would be characterised more by a vee-shaped high pressure stagnation line. This is made impossible in large part due to the fixed, zero trim angle position, but also due to the flow speed. A $0.76m/s$ flow speed, using a reference length of the beam of the boat ($152mm$), gives a Froude number of 0.62. Typically planing begins at a Froude number of approximately 1.00 and higher.

In summary, this experiment provided an order of magnitude validation of the basic free surface flow physics of Palabos. It also established the ability to model a "virtual towing tank" in Palabos. In practice, the civil flume used for this experiment was not appropriate for the modelling of high-speed planing boats due to a low maximum flow rate. Validation of the solver for the more specific design problem of high speed boats will require validation against data from the literature that was collected with access to high-speed marine testing laboratories, as is presented in Section 2.5.

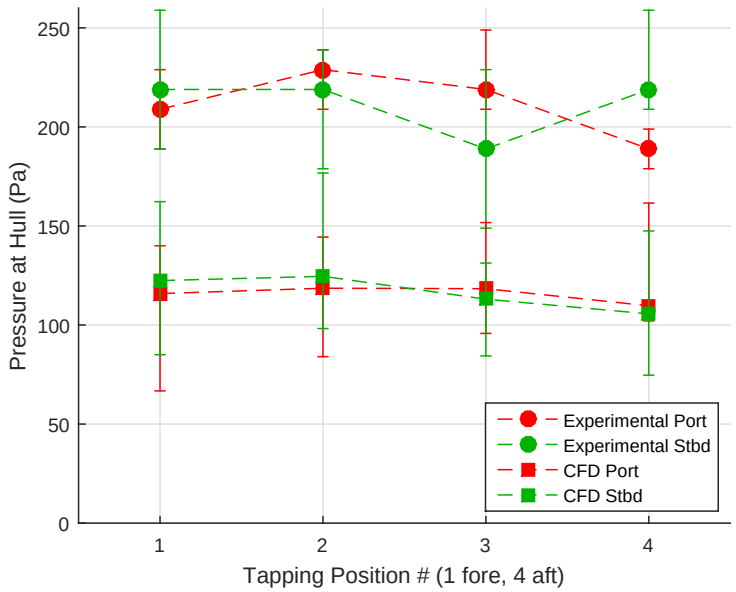


Figure 2.16: Experimental flume pressure results. The mean average values for the 60 second duration are given by the squares/circles, and whiskers show the minimum and maximum pressures measured in that period. The tapping position numbers refer to positions shown in Fig. 2.15.

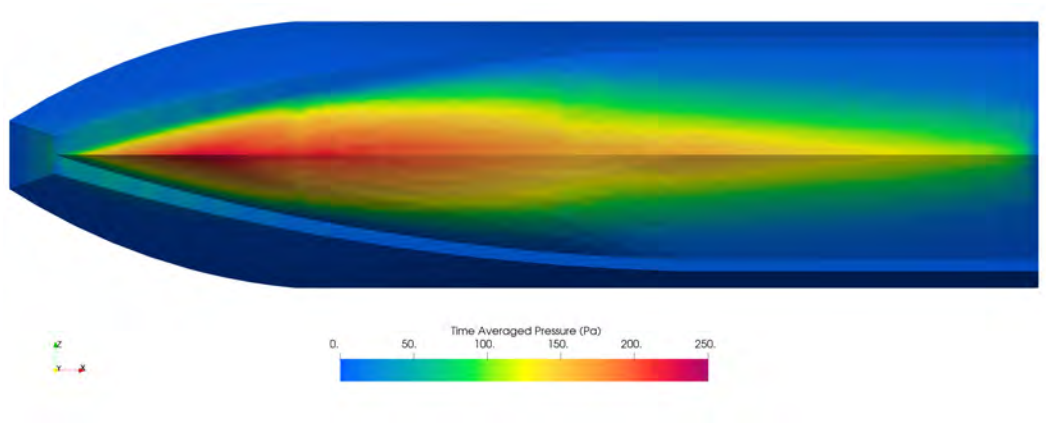


Figure 2.17: The pressure distribution across the hull is shown time-averaged over the whole simulation.

2.5 Validation of the Palabos LBM model against analytic and experimental data on a prismatic hullform

The complexity of modern high-speed boat hullform geometry today can vary greatly, including features such as single or multiple steps, variable deadrise, and air entrainment vents. For the purpose of initial validation of the Palabos solver the simplest form of planing hull (beyond a flat plate) was determined to be the logical starting point: a prismatic vee-hull with constant deadrise angle, single chine, and no discontinuities (steps).

In this section Palabos results are compared to results of the semi-empirical Savitsky method (detailed in Section 1.3.4), and against the empirical results of the Chambliss and Boyd flume experiments [29]. The Chambliss and Boyd experiments were part of a series of experiments carried out in the 1940s and 1950s by the National Advisory Committee for Aeronautics (NACA) at the Davidson laboratory to assess the performance and characteristics of vee-shaped planing hulls. In these experiments, a prismatic vee-shaped hull was positioned in flows of various speeds under various loadings, and data was collected on the wetted hull lengths, resistance, and draft. The simple prismatic hull geometry lends the data to cross-referencing with the Savitsky method. The Chambliss and Boyd data has been used as benchmark data to assess the performance of other computational methods, for example by Brizzolara and

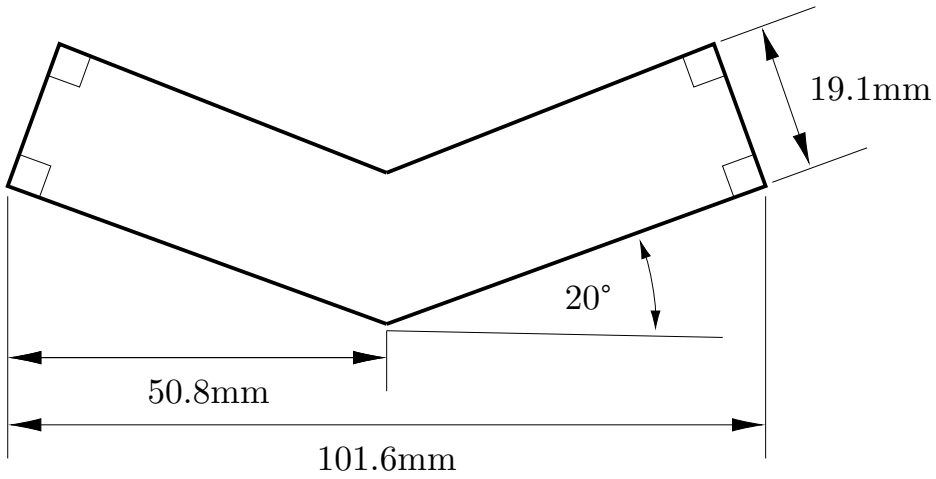


Figure 2.18: Chambliss and Boyd's 20° deadrise hull cross section with dimensions converted to metric.

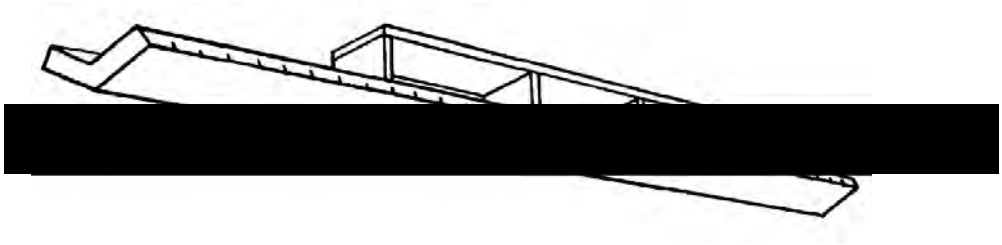


Figure 2.19: The 20° deadrise hull model, as depicted by Chambliss and Boyd [29].

Serra [34].

2.5.1 The Chambliss and Boyd experiments

Figure 2.18 shows the cross sectional dimensions of the 20° deadrise prismatic hull used in the Chambliss and Boyd experiment, and Figure 2.19 shows the roughly 1 metre long (36 inch) hull with its mounting. This mounting was used to pull the attached hull through stationary water. Aerodynamic shielding was placed around the mounting and hull to reduce the impact of aerodynamic resistance, though apart from this no further accounting for aerodynamic resistance was made and it was simply assumed to be negligible relative to the hydrodynamic resistance – this should be borne in mind when assessing resistance values.

Figure 2.20 shows the experimental setup as depicted by Chambliss and Boyd. This rigging allowed for the desired trim and loading to be fixed for a

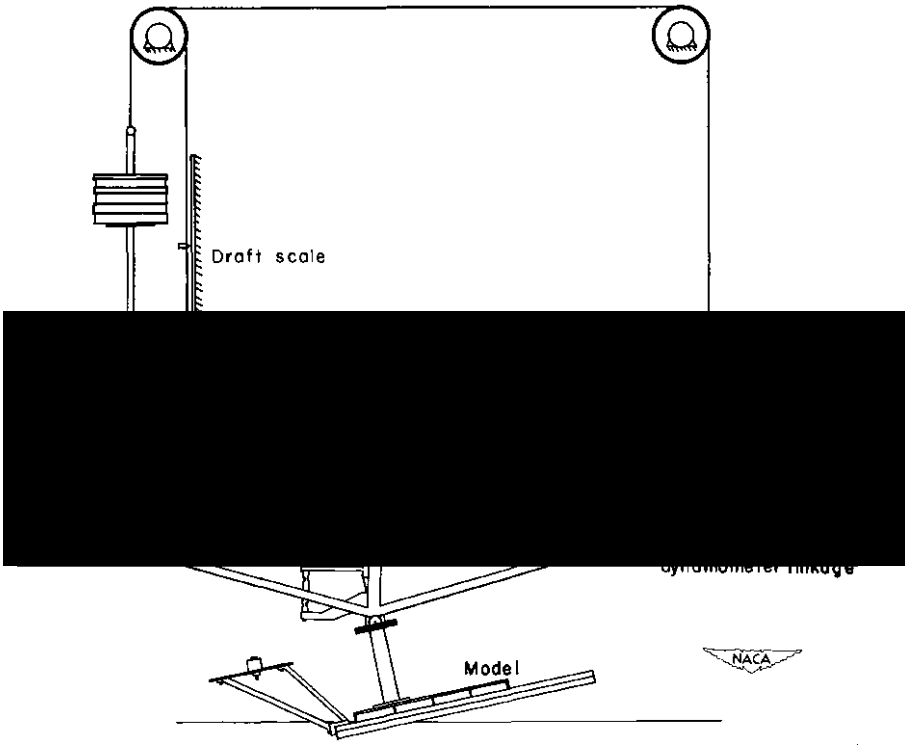


Figure 2.20: The experimental setup, as depicted by Chambliss and Boyd.

given test, but left the hull able to surge against the flow, varying its draft to settle at an equilibrium position. In this sense the independent variables for each experiment are the flow speed, trim angle, and load, while the dependent variables measured after the hull reaches dynamic equilibrium are the draft, wetted lengths, and resistance.

Wetted lengths of the keel and chines were measured using underwater photography (see Figure 2.21 for an example). Dynamic drafts were considered too difficult to measure without unacceptable error, and thus derived from the fixed trim and more accurately measurable wetted lengths as defined in Equation 2.19.

$$d = L_k \sin \tau \tag{2.19}$$

where d is the draft, L_k the wetted keel length, and τ the trim angle.

A notable phenomenon brought up by Chambliss and Boyd is "pile up" at the stagnation point at the keel. The pile up phenomenon is defined as a local increase in the free surface height above that of the global free surface level. Due to wetted keel lengths largely being collected from underwater

Figure 2.21: An example of the underwater photography that informed the wetted length values of the keel and chines in the Chambliss and Boyd experiment [29].

photographs that gave a perspective from directly below the hull (see Figure 2.21), pile up was difficult to discern directly. It was, however, quantified by comparing the difference between the measured wetted keel length and the computed wetted keel length, which revealed greater pile up at higher trim angles, as shown in Figure 2.22 [29].

The calculated draft refers to the draft derived from the wetted keel length as per Equation 2.19, whereas the experimental values are the directly measured dynamic draft (values of which are not comprehensively provided due to difficulty in measurement). In Figure 2.22 a distinct gap between calculated and measured draft appears in the range of $6^\circ \leq \tau \leq 12^\circ$, and is seen to grow even larger for higher trim angles [29].

The Chambliss and Boyd experiments were carried out wholly in imperial units, though for consistency in this work have been converted to metric where necessary. Some of the parameters used by Chambliss and Boyd are non-dimensionalised, and their definitions are given by:

$$\text{Load coefficient: } C_\Delta = \frac{\Delta}{\rho b^3} \tag{2.20}$$

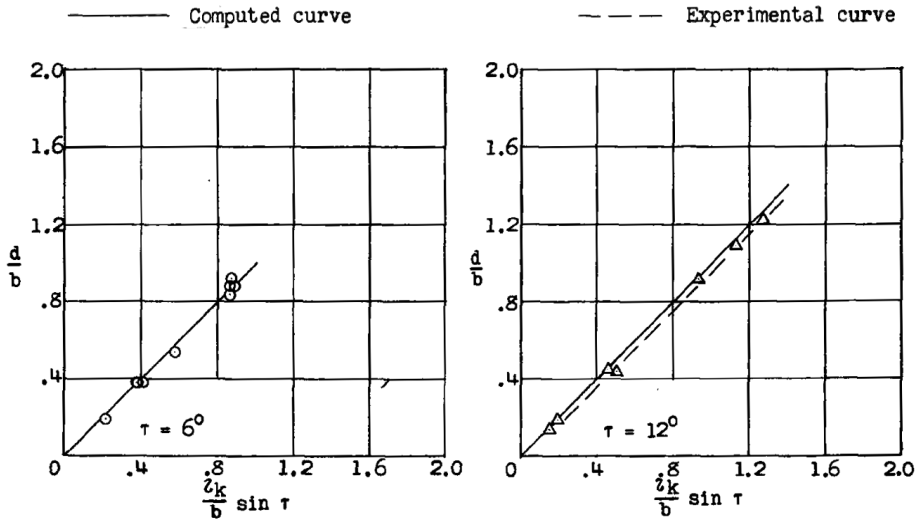


Figure 2.22: A comparison of the experimentally measured draft versus the computed draft in the Chambliss and Boyd experiments [29] for trim angles of 6° and 12°.

$$\text{Speed coefficient: } C_v = \frac{v}{\sqrt{gb}} \tag{2.21}$$

where Δ is the vertical load applied to the hull, b is the beam length, g is acceleration due to gravity, v is the hull velocity, and ρ is the specific weight of water, reported to be $63.4lb/ft^3$ or $1,015.6kg/m^3$ in the Chambliss and Boyd experiment. Note that the speed coefficient, C_v, is equivalent to the Froude number, Fr. Also note that the lift force is not directly measured, but rather assumed to be equal to the vertical loading (Δ) applied to the hull.

The coefficients for lift (C_L) and resistance/drag (C_D) are defined as follows, with coefficients normalised by beam length (b), and also by wetted area (S):

$$C_{L_b} = \frac{\Delta}{\frac{1}{2}\rho v^2 b^2} \tag{2.22}$$

$$C_{D_b} = \frac{R}{\frac{1}{2}\rho v^2 b^2} \tag{2.23}$$

$$C_{L_S} = \frac{\Delta}{\frac{1}{2}\rho v^2 S} \tag{2.24}$$

Table 2.5: Selected scenarios from the Chambliss and Boyd data set for the 20° deadrise hull.

property	units	scenario 1	scenario 2	scenario 3	scenario 4	scenario 5	scenario 6
τ	°	2	2	4	4	6	12
C_Δ	-	4.26	4.26	6.39	19.17	36.21	87.33
C_v	-	13.48	19.89	17.08	21.96	17.48	18.60
v	m/s	13.46	19.86	17.05	21.92	17.45	18.57
C_R	-	5.56	2.06	1.48	4.94	7.90	22.79
l_c/b	-	5.50	0.68	0.38	3.00	7.12	5.50
l_k/b	-	8.42	3.70	1.80	4.42	8.12	5.98
l_m/b	-	6.97	2.19	1.09	3.72	7.62	5.74
l_c	mm	559	69	39	305	723	559
l_k	mm	855	375	183	449	825	608
l_m	mm	708	223	111	378	774	583
C_{L_b}	-	0.0468	0.0216	0.0439	0.0795	0.2370	0.5049
C_{D_b}	-	0.0262	0.0104	0.0103	0.0204	0.0516	0.1317
C_{L_S}	-	0.007	0.010	0.040	0.021	0.031	0.088
C_{D_S}	-	0.0038	0.0047	0.0094	0.0055	0.0068	0.0229

Table 2.6: Chambliss and Boyd experimental measurement absolute error.

Measured variable	Absolute instrument error	Mean relative error	Maximum relative error
Load, lb	±0.15	0.76%	1.50%
Trim, deg	±0.10	2.92%	5.00%
Speed, ft/sec	±0.20	0.35%	0.45%
Resistance, lb	±0.15	1.83%	4.33%
Wetted length, in	±0.25	2.18%	5.73%

$$C_{D_S} = \frac{R}{\frac{1}{2}\rho v^2 S} \tag{2.25}$$

Chambliss and Boyd tested two prismatic hulls of 20° and 40° deadrise at dozens of combinations of speed, trim angle, and loading. Due to the limitations of computational resources, only the 20° deadrise hull is considered at the six scenarios detailed in Table 2.5. Some of the non-dimensional representations of the wetted lengths have been converted to dimensional metric lengths for conceptualisation and direct comparision to the lengths that are later obtained from the Palabos numerical simulations.

Absolute error attributed to the measurement equipment used by Chambliss and Boyd as shown in Table 2.6. The relative errors given by these absolute errors for the sampled scenarios is also presented.

The error accounted for here is related solely to error in the measuring instrumentation. This does not account for any further error originating from, for example, small variations in the equilibrium position of the hull or human error in measuring wetted lengths where underwater photography was not taken. Error in these experiments is expanded on in the discussion in Section 2.5.4.

2.5.2 Application of the Savitsky method

The Savitsky method – covered in Section 1.3.4 – is an analytical method developed around empirical data for predictive analysis of planing hullforms: equations were fit to empirical data gathered and published by Savitsky and Neidinger [28]. An analytical approach like this is not best suited to a physical problem as complex as the flow around a high speed planing boat, though the method remains popular to this day among smaller boat designers. This is in no small part due to the ease of application and minimal resource requirements to apply the method in comparision to a more complex and compute-intensive physics solver, or indeed an array of towing tank experiments.

The Chambliss and Boyd experiments were selected in part due to their use of a very simple prismatic hullform that minimises the assumptions made in the Savitsky method, as opposed to a complex hull with variable deadrise angles, steps, or spray rails. In doing this, validation of the Palabos solver could be made against two approaches that are well referenced within the field of planing boats: that is, the Chambliss and Boyd data set and the Savitsky method.

The following equations define the method applied to each of the six hull scenarios sampled from the Chambliss and Boyd experiments.

Firstly, the flat-plate lifting coefficent is found for a reference hull at the same trim angle, mean wetted length, and Froude number:

$$C_{L_0} = \tau^{1.1}\left(0.0120\lambda^{\frac{1}{2}} + \frac{0.0055\lambda^{\frac{1}{2}}}{F_r^2}\right) \tag{2.26}$$

The lift coefficient for a hull of deadrise angle β is empirically derived by Savitsky from the lift coefficient of a flat plate (C_{L_0}) operating at the same τ, λ, and F_r values by:

$$C_{L_\beta} = C_{L_0} - 0.0065\beta C_{L_0}^{0.60} \tag{2.27}$$

Regarding drag, the following equation is used by Savitsky where whisker spray drag is ignored:

$$D = \Delta \tan\tau + \frac{\rho v_1^2 C_f \lambda b^2}{2\cos\beta\cos\tau} \tag{2.28}$$

where v_1 is the mean bottom velocity and C_f is the Schoenherr turbulent friction coefficent [113], for which the value was taken as used by Savitsky [30].

The mean bottom velocity is expressed by Savitsky and Ross [114] as:

$$\frac{v_1}{v} = \sqrt{1 - \frac{0.0120\tau^{1.1}}{\lambda^{\frac{1}{2}}\cos\tau} f(\beta)} \tag{2.29}$$

where v_1 is dependant on trim angle τ, mean wetted length λ, and a function of the deadrise angle β. This relationship was determined empirically by Savitsky and Ross [114] and is represented graphically in Figure 2.23 [30] for a deadrise of 20°. Note that Equation 2.29 takes into account bouyant forces and is thus only applicable for ($1.0 \leq F_r \leq 13.0$), as beyond that range the bouyant forces being accounted for compose a negligible contribution to drag. As all scenarios lie in a speed regime greater than $F_r = 13.0$, care should be taken in using the Savitsky derived whisker spray resistance forces. This is discussed in Section 2.5.4.

Savitsky later accounted for whisker spray drag in 2007 [33]. Whisker spray drag is the drag attributed to the spray projected fore of the stagnation line. Savitsky defines the following expression for whisker spray drag:

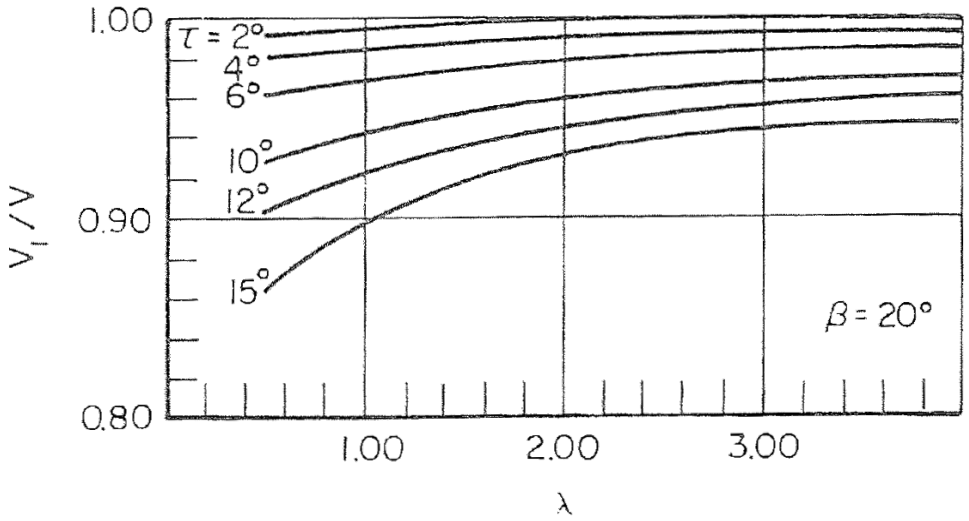


Figure 2.23: Graphical representation of the relation given in Equation 2.29 derived by Savitsky and Ross [114, 30] for the range of $1.0 \leq F_r \leq 13.0$. Here v_1 is the mean bottom velocity.

$$R_S = \frac{1}{2}\rho v^2 \Delta\lambda b^2 C_f \tag{2.30}$$

where $\Delta\lambda \times b^2$ is the effective increase in wetted area attributed to whisker spray contribution to total resistance, for which values were determined graphically from Figure 2.24. In this case, C_f is the friction coefficient as defined by Savitsky in the whisker drag formulation [33]. This friction coefficient is qualitatively determined by the turbulence of the flow, which is in turn influenced by the geometry of the whisker spray such that:

$$Re_{WS} = \frac{vL_{WS}}{\nu} \tag{2.31}$$

where Re_{WS} is the Reynolds number as determined by the characteristic length of L_{WS}, which is the length of the whisker spray drag determined geometrically by:

$$L_{WS} = \frac{1}{2}\frac{b/2}{\sin 2\alpha \cos\beta} \tag{2.32}$$

where α is the angle between the keel and stagnation line in the horizontal ground plane, and β is the deadrise angle. Thus the size of L_{WS} will be

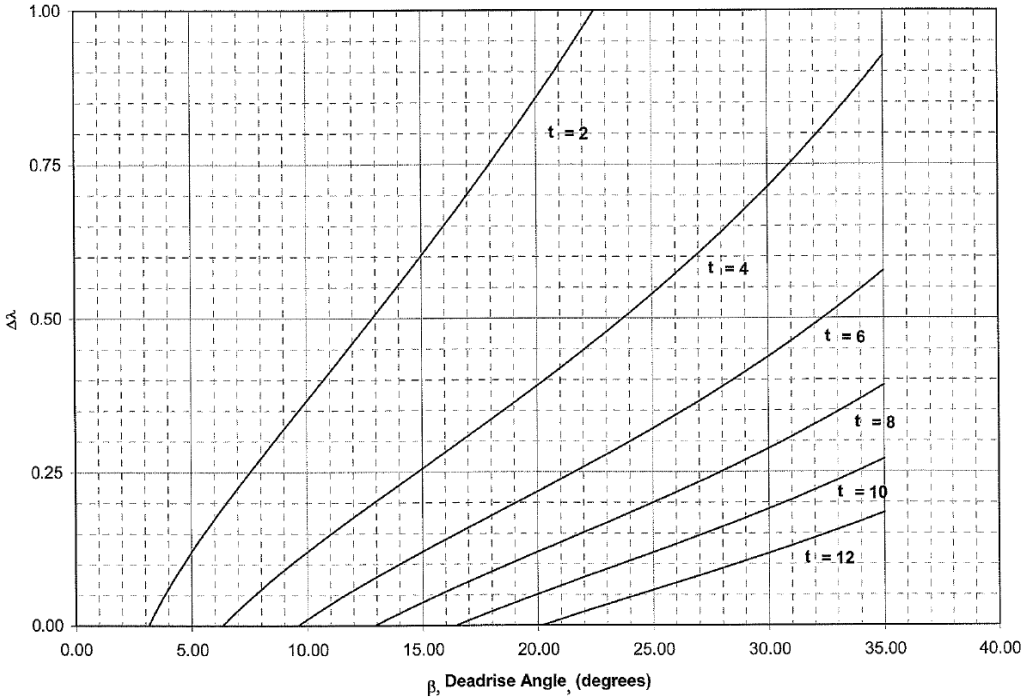


Figure 2.24: The increase in non-dimensional wetted length-beam ratio due to whisker spray ($\Delta\lambda$) is plotted as a function of deadrise angle (*beta*) and trim (here denoted by t) [33].

substantially smaller than the mean wetted length – used to determine the Reynolds number for the pressure area – in most cases, driving the need to consider the turbulence of the whisker spray region seperately. In all cases tested, the turbulence of the whisker spray region lay in the laminar region ($Re_{WS} < 1.5 \times 10^6$), and so the friction coefficient was determined by [33]:

$$C_f = \frac{1.328}{\sqrt{Re_{WS}}} \tag{2.33}$$

$$R_s = \frac{1}{2}\rho v^2 \Delta\lambda b^2 C_f \tag{2.34}$$

Equation 2.34 for total drag associated with whisker spray can then be summed with Equation 2.28 for drag in the pressure area to obtain overall drag inclusive of whisker spray. Resistance values both with and without whisker spray drag are presented and discussed in Secton 2.5.4.

The results of the Savitsky analysis for the selected scenarios are presented in Section 2.5.4 where they are compared to Palabos numerical results.

2.5.3 Palabos numerical simulations

The Palabos numerical simulation setup necessarily differed from the Chambliss and Boyd physical experiments. The physical experiments involved towing a hull through static water at a fixed trim, speed, and loading, then measuring the wetted lengths and forces. The numerical experiments were conducted, in contrast, on a static hull at a fixed trim angle placed in a flow of constant velocity. This approach of a globally fixed geometry with moving flow is common in both physical testing (tow tank or wind tunnel) and numerical simulations, and is physically equivalent to a real boat moving across globally stationary water. The differences arise not from the perspective of the relative motion, but in the practicalities of the experimental setup: some parameters will be dependent in one experiment, but independant in the other.

For example, in the Chambliss and Boyd experiment, dynamic draft and wetted lengths are dependent, measured variables, while load is a dictated independent variable. In the Palabos numerical simulations, the wetted lengths from the Chambliss and Boyd data are used to calculate the fixed draft of the hull and position it in the flow, while the resulting vertical load is a dependent, measured variable.

In the Chambliss and Boyd experiments the dynamic draft is calculated from the wetted keel length, as mentioned in Section 2.5.1, due to the more accurate measurements of wetted lengths versus direct observation of the dynamic draft. In determing the fixed draft of the hull in the Palabos simulations, wetted keel length is used, though it should be kept in mind that "pile up" may be included in these wetted lengths. Pile up is discussed in Section 2.5.1, and was found to occur measurably in the case of trim angles of 12° or more. For this reason it is expected that scenario 6 will involve a hull that has been given slightly greater draft than in the Chambliss and Boyd experiements due to a keel length measurement inflated by pile up.

The control parameters of the Palabos numerical simulations are given in Table 2.7. These values are constant through all scenarios. The parameters that vary by scenario are then given along with the results in Table 2.8, and an example visualisation of the setup of scenario 4 is given in Figure 2.25. Result

Table 2.7: Prismatic validation Palabos simulation control parameters.

parameter	*units*	*value*
domain size:		
x range	(m)	$[-1.10, 0.50]$
y range	(m)	$[-0.40, 0.11]$
z range	(m)	$[-0.40, 0.40]$
absorbing zone widths:		
inlet	(m)	0.05
outlet	(m)	0.10
lateral	(m)	0.10
top	(m)	0.01
fluid properties:		
density, ρ	(kg/m^3)	$1,015.57$
kinematic viscosity, ν	(m/s^2)	$9.801e-7$
fluid height	(m)	0.4
surface tension	(N/m)	0.0728
Palabos parameters:		
resolution	$(-)$	501
dx	(m)	0.002
characteristic length, L_{char}	(m)	1.0
lattice velocity, u_{LB}	(m/s)	0.01

values of force have been converted to dimensionless lift and drag coefficients normalised by beam.

These simulations were run on the Supercomputing Wales' Sunbird HPC system as detailed in Section 1.6. Each simulation was allocated 300 cores with 4GB of memory per process for up to 72 hours wall clock time, during which all simulations achieved convergence residuals of the order 10^{-3}.

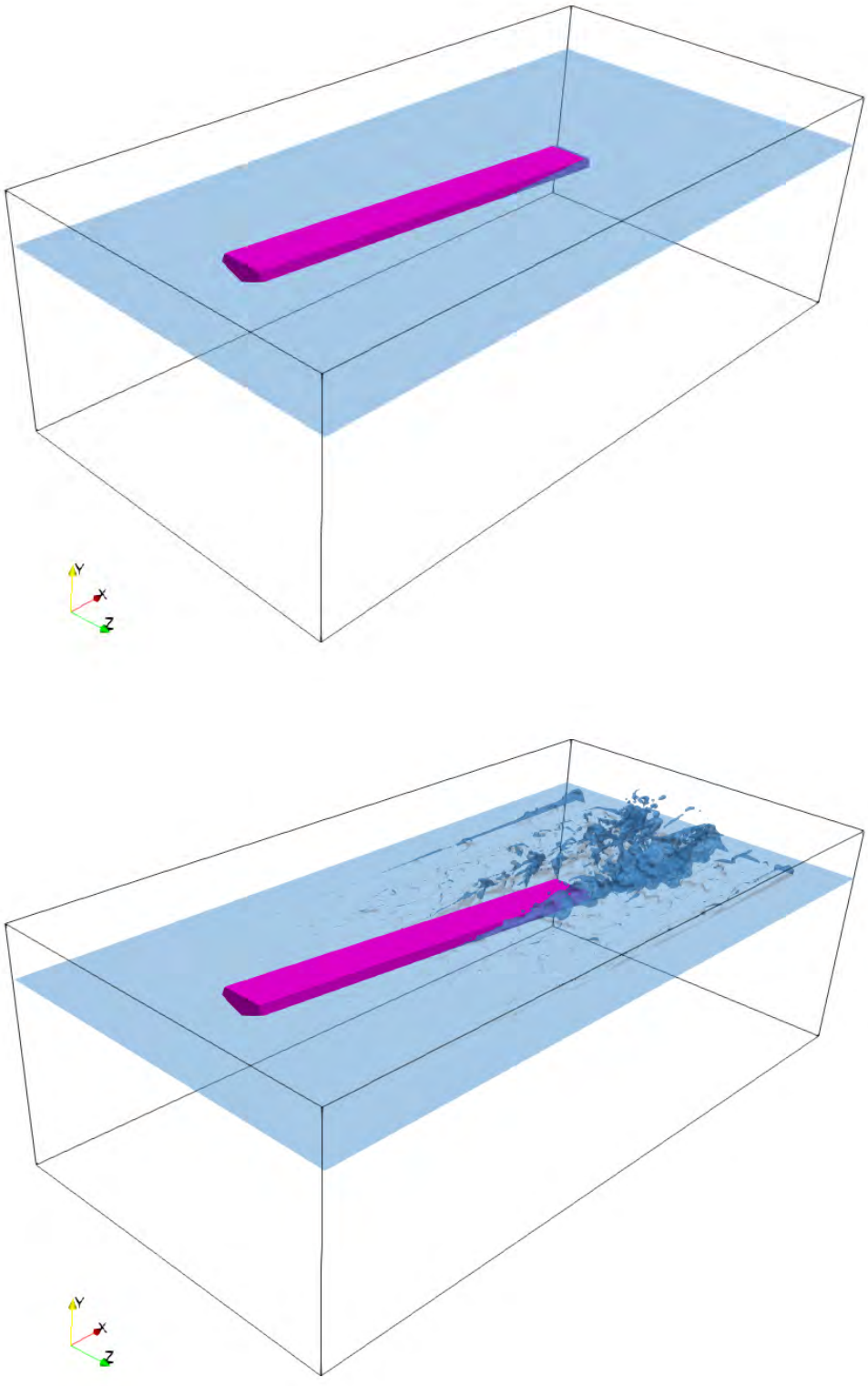


Figure 2.25: an example of the virtual flume setup in Palabos for the case of scenario 4, both at initiation (above) and 1 second into the simulation (below). The hull geometry (pink), free surface interface (blue), and domain bounds (black) are shown.

Table 2.8: Prismatic validation Palabos simulation parameters for each scenario.

property	units	scenario 1	scenario 2	scenario 3	scenario 4	scenario 5	scenario 6
		Variables					
τ	(°)	2	2	4	4	6	12
d	(mm)	29.9	13.1	12.8	31.3	86.2	126.3
v_{inlet}	(m/s)	13.46	19.86	17.05	21.92	17.45	18.57
		Results					
Lift	(N)	23.474	18.774	43.353	146.11	308.39	815.84
C_{L_b}	(-)	0.02473	0.00908	0.02845	0.05799	0.19319	0.45139
Resistance	(N)	2.1176	2.4152	4.2067	13.466	34.484	176.43
C_{D_b}	(-)	0.00223	0.00117	0.00276	0.00534	0.02160	0.09761

2.5.4 Results and discussion

In comparing the results of the three data sets, consideration should be given to the necessary difference in the experimental setup of the Chambliss and Boyd experiments versus the Palabos numerical simulations regarding pile-up, discussed in Section 2.5.3. Evidence of a variation in the wetted chine and keel lengths between the Chambliss and Boyd data and the Palabos simulations is presented in Figure 2.26. From Figure 2.26 it can be seen that there is a consistent trend towards the Palabos simulations exhibiting longer wetted lengths at the keel and chine by roughly proportional amounts for each scenario. This consistent trend is in line with the prior assumption that in placing the hulls at the positions determined by the Chambliss and Boyd wetted lengths, the pile up effect, although small, would be compounded.

Further evidence of this is seen specifically for scenario 4 in Figure 2.27, where an artificial plane (grey) is placed at $1mm$ above and parallel to the free surface initial head, revealing any fluid (blue) rising above the far field fluid height by more than $1mm$. This reveals the expected local rise ahead of the keel of the hull (pink) in excess of $1mm$, consistent with the pile up phenomena described by Chambliss and Boyd.

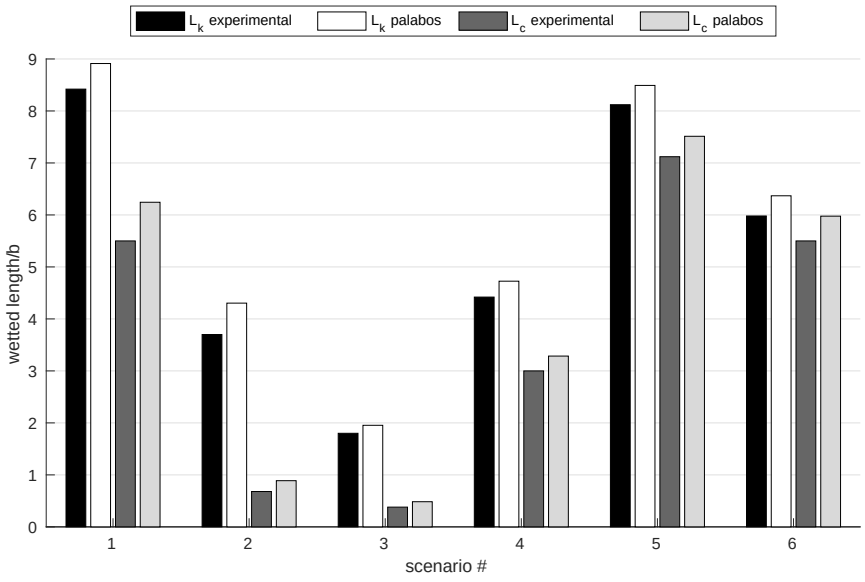


Figure 2.26: The wetted keel length (L_k) and wetted chine length (L_c) for both the Chambliss and Boyd experiments and Palabos numerical results are shown.

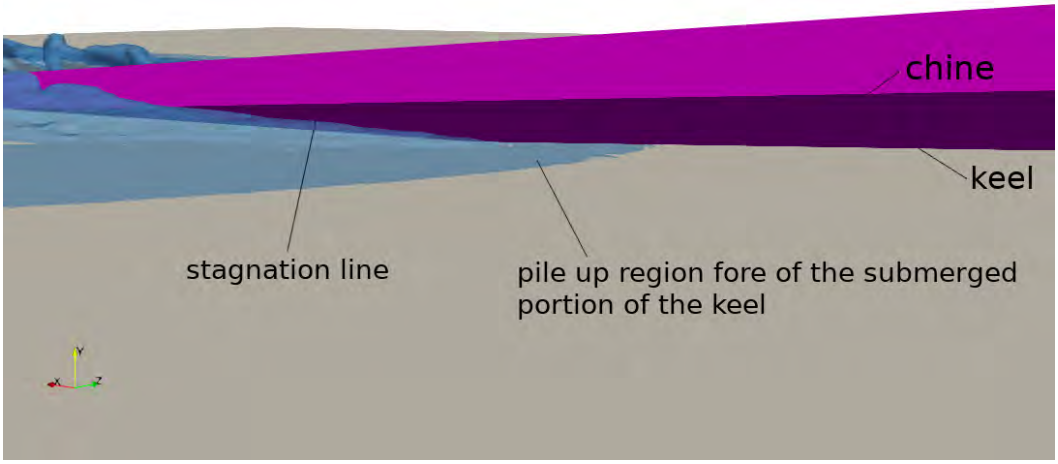


Figure 2.27: Scenario 4 is depicted with the hull (pink) seen from the side and a horizontal plane (grey) at $y = 0.0001m$ to reveal the pile-up of water (blue) ahead of bow. Flow is moving in the x-direction.

The expected result of this – borne out by Equations 2.26 and 2.28 and observed trends in the experimental data of Chambliss and Boyd – is that an increase in the mean wetted length (i.e.: greater hull sink) as seen in all of the Palabos simulations should result in an increase in both lift and resistance above the expected results. This should be kept in mind when comparing the Chambliss and Boyd data and Savitsky method results (which are based on the Chambliss and Boyd wetted lengths) against the Palabos numerical results.

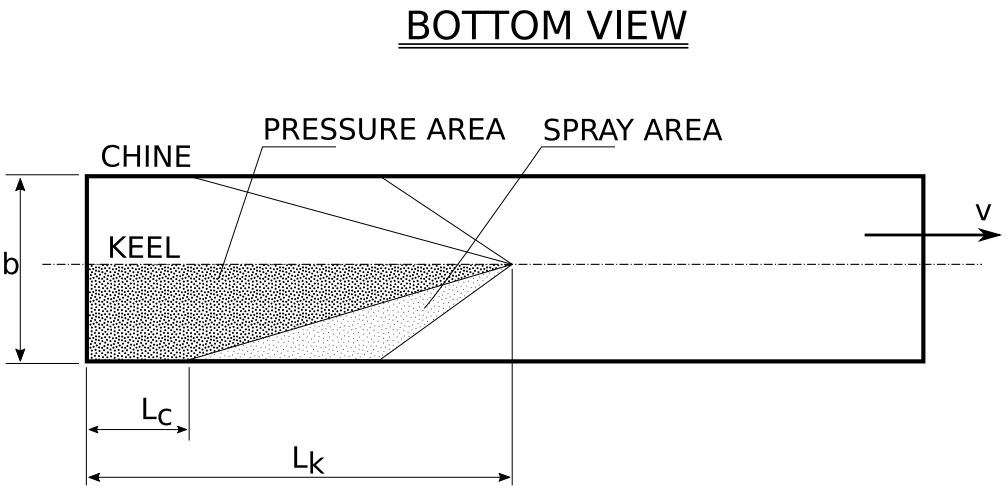
It is also worth noting that, taking into account the overall increase assumed to be arising from compounded pile up in the experimental setup, the Palabos wetted lengths show a consistent trend in line with the Chambliss and Boyd observations. This is indicative of some degree of accuracy in the resolution of the free surface. A global grid resolution of dx = $2mm$ was necessary due to limits in computational resources. Such a spatial resolution means that physical phenomena that occur on a smaller lengthscale than $2mm$ will not be captured. This is most noticably apparent with the whisker spray region, which is absent ahead of the stagnation line, as seen in Figure 2.28. The whisker spray region is defined by a very thin layer of flow in contact with the hull surface projected forward from the stagnation line, and so it

is not surprising that a $2mm$ grid resolution did not capture this. To assess the capability to capture whisker spray drag with equivalent computational resources, local grid refinement would need to be added to the current suite of capabilities. This was deemed beyond the scope of this project, and would be recommended for future work.

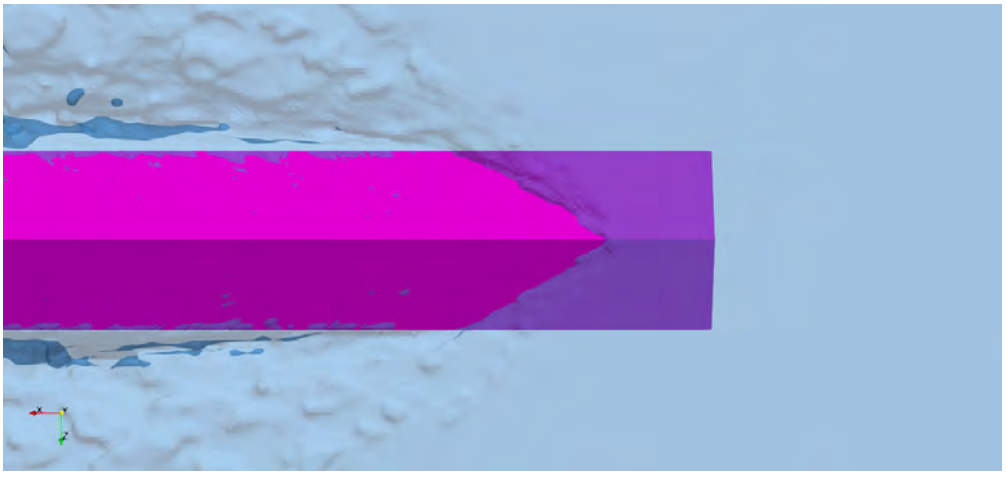
Full numeric data is presented in Table 2.9. The lift and resistance coefficients of the Palabos simulations are presented graphically in comparison to the Chambliss and Boyd and Savitsky analytical method in Figure 2.29 and 2.30 respectively. In each case, coefficients of lift and resistance are given normalised by hull beam (constant across all methods and scenarios), and normalised by wetted area (varies from scenario to scenario and also between numerical versus other methods). This is done to account for the inconsistency in wetted area caused by pile up, though as can be seen by comparing subfigures of Figures 2.29 and 2.30, the general trends are preserved with only small variations in relative values between methods.

Regarding lift, the Palabos numerical results lie between the Chambliss and Boyd data and the Savitsky predictions, though there is a large discrepency between the Chambliss and Boyd data and the Savitsky predictions themselves, with Savitsky underpredicting by a wide margin in all cases. This is not entirely unexpected, as the underlying flat plate lift equation (Equation 2.26) is only valid for the regime defined by $0.60 \leq F_r \leq 13.00$, $2^\circ \leq \tau \leq 15^\circ$, and $L_m/b \leq 4$. Although all scenarios chosen fall within the trim angle constraint, only scenarios 2 and 3 meet the mean wetted length constraint, and all scenarios are in excess of the Froude number constraint. For this reason, lift performance should only be meaningfully compared to the Chambliss and Boyd empirical data set.

Regarding resistance, the Chambliss and Boyd and Savitsky data match one another more closely, with the Palabos numerical results under-predicting resistance. Recall that Chambliss and Boyd considered the aerodynamic resistance of their rigging to be negligible, providing a possible source of additional resistance that may in some part narrow the gap between the numeric and experimental results. Little information is available about the rigging besides



(a) A general example of the flow on the bottom of a simple planing hull of constant deadrise and with no discontinuities according to Savitsky [30]. There is a pressure region aft of the stagnation line, and a whisker spray region fore of the stagnation line.



(b) Scenario 5 viewed from below. The hull is pink, and the free surface blue with 50% opacity to reveal any whisker spray fore of the stagnation line. The stagnation line and resulting pressure area are clearly defined, however no whisker spray region is captured.

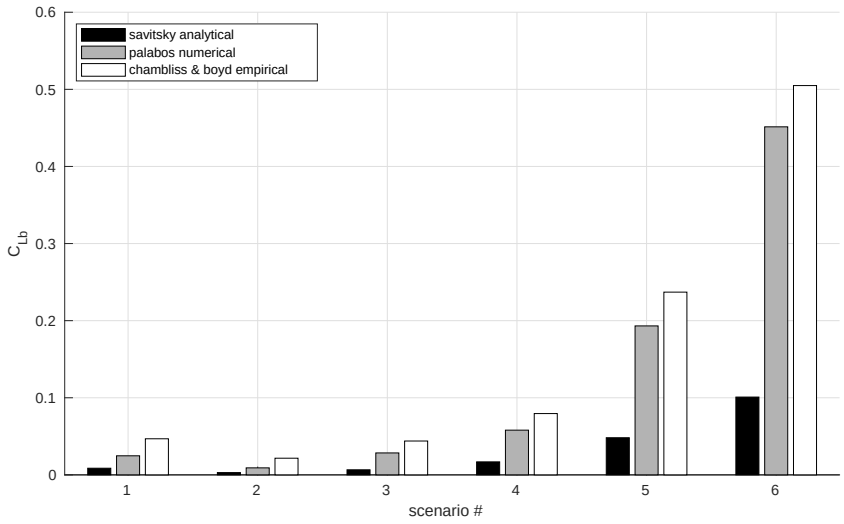
Figure 2.28: Savitsky's description of the regions on a planing hull compared to observations from the Palabos numerical simulations.

the provided sketches (see Figures 2.19 and 2.20) and most importantly the mentioned aerodynamic fairing is not detailed, leaving no way to accurately account for possible misattributed resistance.

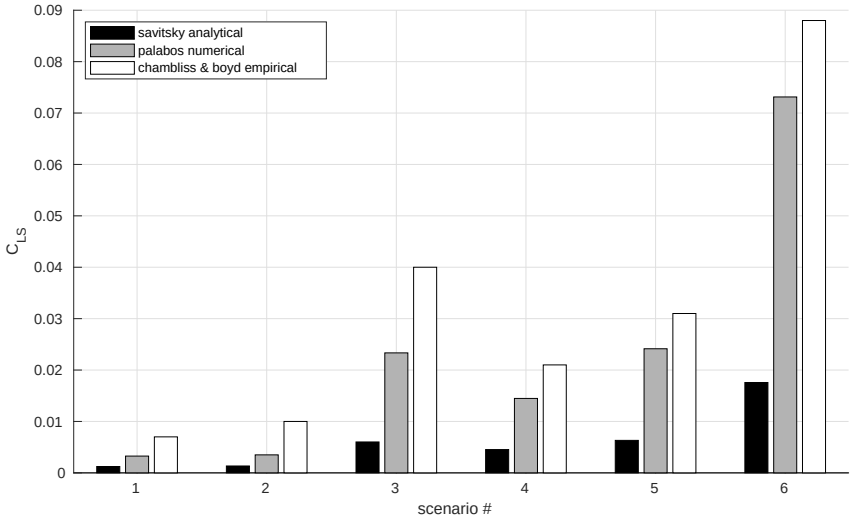
Total resistance inclusive of whisker spray drag is shown directly next to resistance without whisker spray resistance in Figure 2.30. Whisker spray resistance is also based on a mean bottom velocity derived from equations valid only at $F_r \leq 13.00$, while all scenarios are at speeds in excess of this. The formulation of Equation 2.29 indicates that whisker drag will be overestimated due to over accounting for negligible bouyanancy-caused resistance, so in this case the already fairly negligible addition from whisker spray drag is expected to be even smaller. The physical pheneomena of whisker spray drag was not observed in the Palabos numerical simulations, and so the accompanying addition to resistance would not be expected either. The lack of available grid refinement to the physical lengthscale on which whisker spray drag occurs inhibits any further investigation in this respect.

Both these trends for lift and drag are reported in similar comparisons between Chambliss and Boyd data and analytical methods (Savitsky and Shuford methods) against numerical approaches; Brizzolara et al. found that numerical lift results for a RANS method most closely matched experimental values (numerical underprediting versus experimental) rather than analytical results (where numerical results over-predicted) [34]. Numerical resistance results similarly saw a general under-prediction versus both experimental and analytical approaches, which agreed with one another more closely than in the case of lift.

In general, the trends of lift and resistance in response to the tested parameters are captured, though lift and resistance are both under-predicted (discounting the Savitsky lift predictions for the stated reasons). Free surface behaviour shows close agreement with literature data once the compounding pile-up effect is accounted for. This preliminary validation study indicated promise in applying the Palabos Lattice Boltzmann method solver to the task of modelling planing boats, and put in place confidence to expand on the capabilities presented so far.

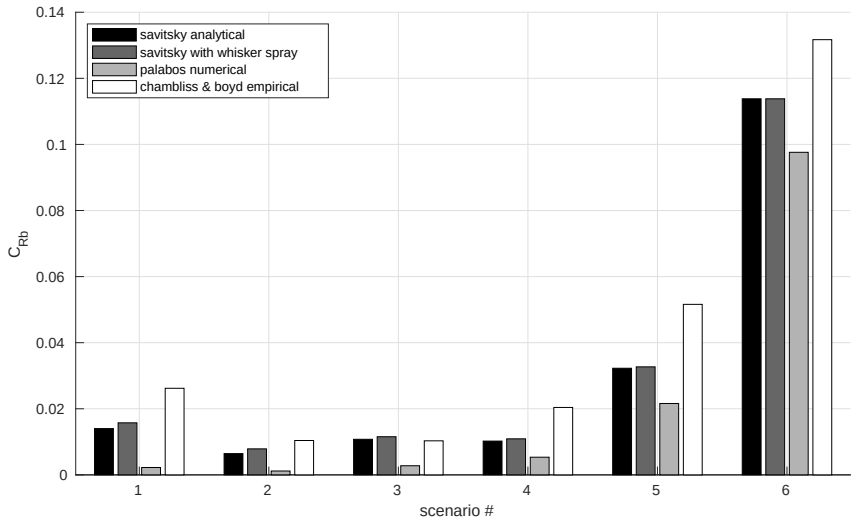


(a) Coefficient of lift normalised by hull beam (C_{L_b}), which is constant between all three data sets.

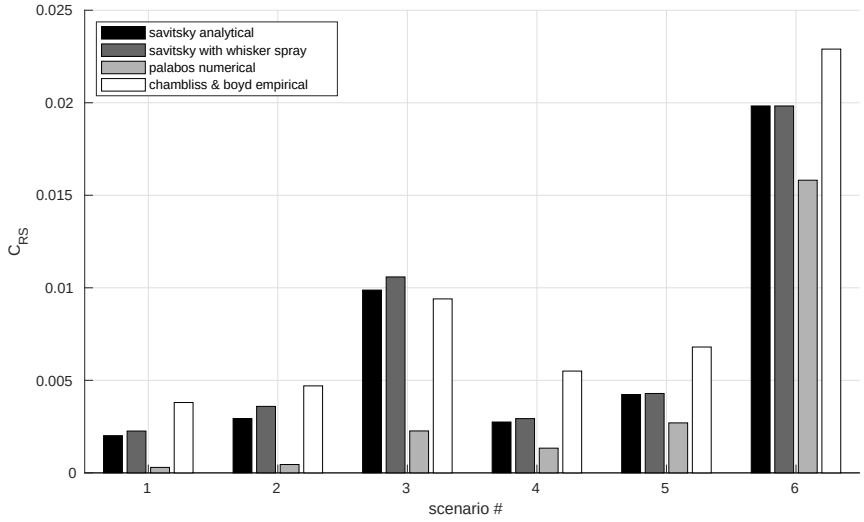


(b) Coefficient of lift normalised by wetted area (C_{L_S}), which differs between the Palabos results versus the Chambliss and Boyd and Savitsky results.

Figure 2.29: Coefficient of lift results from the Savitsky analytical predictions, Palabos numerical simulations, and Chambliss and Boyd experiments are compared for each of the six scenarios given in Table 2.5. The coefficient is normalised by both beam (b) and wetted area (S).



(a) Coefficient of resistance normalised by hull beam (C_{R_b}), which is constant between all three data sets.



(b) Coefficient of resistance normalised by wetted area (C_{R_S}), which differs between the Palabos results versus the Chambliss and Boyd and Savitsky results.

Figure 2.30: Coefficient of lift results from the Savitsky analytical predictions, Palabos numerical simulations, and Chambliss and Boyd experiments are compared for each of the six scenarios given in Table 2.5. The coefficient is normalised by both beam (b) and wetted area (S).

Table 2.9: Prismatic validation results (C&B refers to Chambliss and Boyd data).

property	scenario 1	scenario 2	scenario 3	scenario 4	scenario 5	scenario 6
trim angle, τ (degress)	2	2	4	4	6	12
Froude Number, Fr	13.48	19.89	17.08	21.96	17.48	18.6
L_k/b						
Chambliss & Boyd	8.42	3.70	1.80	4.42	8.12	5.98
Palabos	8.91	4.30	1.95	4.73	8.49	6.37
% error vs C&B	105.8%	116.3%	108.5%	106.9%	104.6%	106.5%
L_c/b						
Chambliss & Boyd	5.50	0.68	0.38	3.00	7.12	5.50
Palabos	6.24	0.89	0.48	3.29	7.51	5.98
% error vs C&B	113.5%	130.7%	127.3%	109.5%	105.5%	108.7%
L_m/b						
Chambliss & Boyd	6.97	2.19	1.09	3.72	7.62	5.74
Palabos	7.58	2.60	1.22	4.01	8.00	6.17
% error vs C&B	108.7%	118.5%	111.8%	107.7%	105.0%	107.5%
$S, (m^2)$						
Chambliss & Boyd	0.07185	0.02261	0.01125	0.03830	0.07866	0.05925
Palabos	0.07823	0.02680	0.01258	0.04135	0.08260	0.06372
% error vs C&B	108.9%	118.5%	111.8%	108.0%	105.0%	107.5%
C_{L_S}						
Chambliss & Boyd	0.007	0.01	0.04	0.021	0.031	0.088
Palabos	0.00326	0.00350	0.02334	0.01448	0.02414	0.07313
% error vs C&B	46.6%	35.0%	58.3%	68.9%	77.9%	83.1%
Savitsky	0.00124	0.00134	0.00601	0.00453	0.00632	0.01758
% error vs Savitsky	264.0%	261.9%	388.3%	319.8%	137.3%	416.0%
C_{R_S}						
Chambliss & Boyd	0.0038	0.0047	0.0094	0.0055	0.0068	0.0229
Palabos	0.00029	0.00045	0.00226	0.00133	0.00270	0.01581
% error vs C&B	7.7%	9.6%	24.1%	24.3%	39.7%	69.1%
Savitsky	0.00201	0.00294	0.00988	0.00275	0.00423	0.01983
% error vs Savitsky	14.7%	15.3%	22.9%	48.6%	63.8%	79.8%

Chapter 3

Addition of heave dynamics capabilities to the boat hullform solver

3.1 Dynamic hullform solver introduction

Planing boat dynamics are sensitive to wetted area and trim. For this reason the solver as it has been presented in previous chapters (with a static hull geometry) has limited use in the practical application of boat design. Based on input from industry sponsors it was decided that the most useful first extension of the current capabilities would be to add the rigid body dynamic heave response between the hull and the flow. Heave dynamics would allow hullforms to settle to an equilibrium heave position for a given flow speed, trim and weight, rather than being set at a fixed position and arriving at non-equilibrium steady-state force values. This reduces the number of static simulations that would be required to evaluate the heave equilibrium position, as otherwise a range of heave positions would need to be tested with interpolation between them to find the equilibrium position.

Figure 3.1: The V15 at sea, courtesy of Privinvest [115].

3.2 V15 prescribed dynamics study

At the request of industry partner Norson Design, an experiment was carried out to assess the lift and resistance prediction of the solver on a hullform that they had design experience with rather than the simple prismatic hullforms tested so far. The hullform selected was the V15, depicted in Figure 3.1. The V15 is a high speed interceptor with a deep-vee hull and hard chines measuring $15.5m$ in length and $3m$ in beam, driven by twin propellors powered each by two diesel engines [115]. Capable of over $50kts$ ($57.5mph$, $25.7m/s$), the V15 is an archetypical non-stepped high speed planing hullform and was thus a good candidate for assessing the solver capabilities for high speed planing boats in general.

This experiment was carried out as a prelude to dynamics capabilities, and so a prescribed dynamics approach was taken instead: the hullform position and flow speed was cycled through four prescribed positions and flow speeds based on available analytical and sea trial data provided by Norson Design, which is detailed in Table 3.1. Using an automated HPC submission script, the position and flow speed was cycled through the configurations in Table 3.1 for half second periods. The hypothesis was that, as the positions in Table

3.1 are for heave equilibrium conditions, the reactive heave force experienced by the hull would remain constant at approximately the same magnitude as the weight of the boat.

Table 3.1: Industry-provided V15 planing equilibrium position values for various speeds.

Speed (kts)	Speed (m/s)	Trim (°)	Sink (mm)
25	12.86	5.0	800
35	18.01	4.0	700
45	23.15	3.0	600
55	28.29	2.2	520

The data provided by Norson Design features estimated equilibrium position based on analytical calculations that were then confirmed by observations of the vessel in near-still waters at high speeds. This project is seeking to replace/supplement the analytical method with a more accurate computational prediction. Further, observational confirmation had an unquantified degree of accuracy as factors such as inconsistent speed, small waves/disturbances, and inherent oscillatory heave and trim motion meant the boat was never in a perfect equilibrium condition. The boat weight was estimated at 10 tonnes based on a known empty weight plus an approximated equipment weight.

The goal of this experiment was to assess whether Palabos could return a reasonable lift force for each position given the estimates of real performance, and also as a trial of moving the hull geometry within a simulation as a proof-of-concept for the full dynamics implementation.

The results of this experiment are shown in Figure 3.2. The coloured bands define the time windows spent in each of the four flow speeds and positions, with the hull geometry and flow speed instantaneously changing to the next configuration at each $0.5s$ interval. The red dotted line denotes the estimated weight of the boat, which the lift response is expected to match. The raw lift response data is given along with a smoothed $0.05s$ moving average to better reveal trends.

From Figure 3.2 it can be seen that the lift response falls approximately within a ±20% margin from the estimated equilibrium position, with lift re-

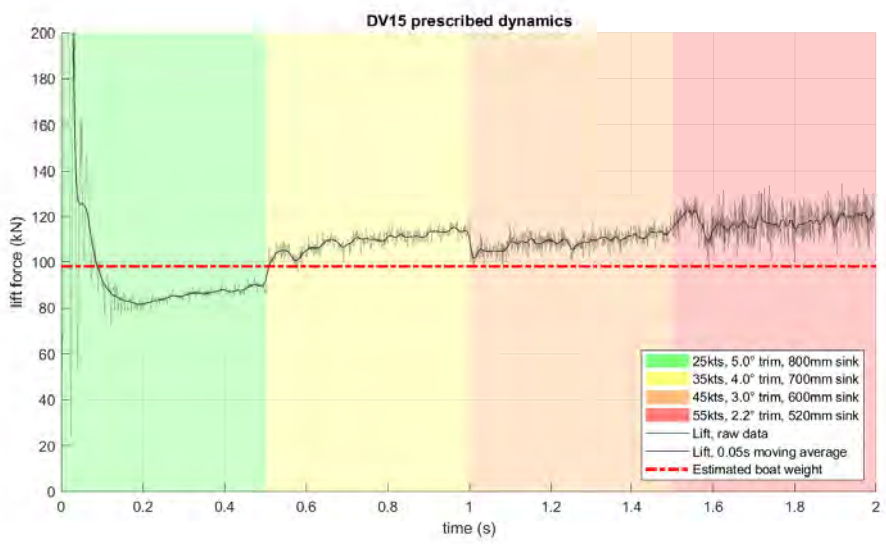


Figure 3.2: The lift force response of the V15 for the prescribed dynamics experiment.

sponse not drastically changing between each configuration despite significant changes in flow speed. This granted confidence that this implementation was matching the predicted behaviour in heave.

The prescribed dynamics method used to achieve this performed as intended, allowing the position changes to happen as an automated checkpointing event within a single submitted job to the cluster. The job batchfile ran a Matlab script on a single HPC core, which in turn submitted the Palabos job to be run on the remaining allocated cores. This Matlab script submitted the first Palabos simulation and at each $0.5s$ interval used Palabos' native checkpointing method to checkpoint the simulation, change the simulation parameters to the next hull position and flowspeed, and resubmit the simulation continuing from the checkpointed state. By automating this process the full 72 hour cluster wall clock time limit could be used, reducing both time spent queueing on the HPC system and any downtime caused by the user not instantly submitting the follow-up job.

The change in hull position and flow speed is depicted in Figure 3.3, where the hull is seen from the side (pink), with the fluid free surface depicted in light blue with opacity, and a slice through the fluid domain along the centreline of the hull plots the velocity of the flow. The four images show the simulation at the end of each flowspeed region, from top to bottom: $0.5s(25kts)$, $1.0s(35kts)$,

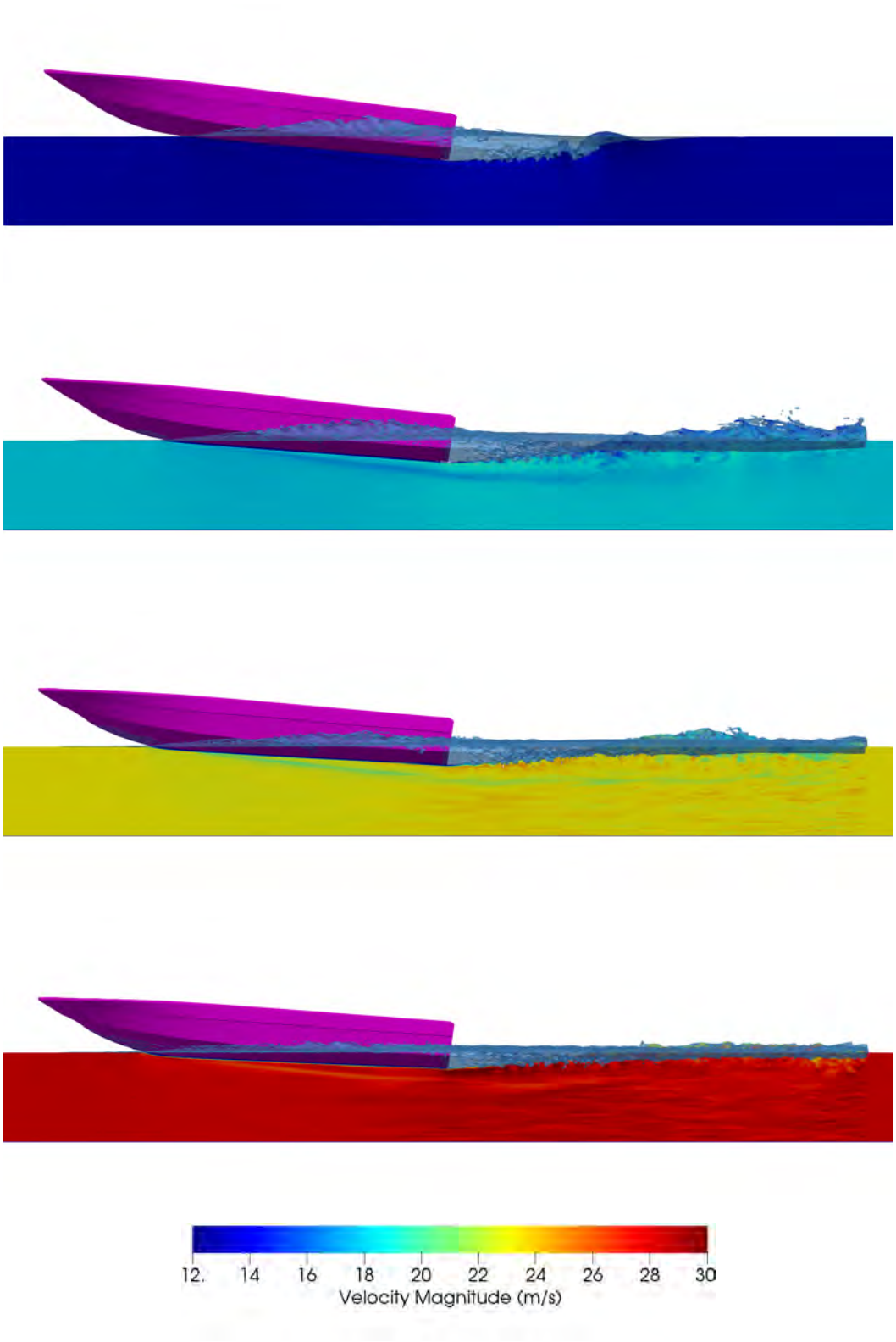


Figure 3.3: The V15 hull is shown from the side at (from top to bottom) $t = 0.5s$, $t = 1.0s$, $t = 1.5s$, and $t = 2.0s$ through the prescribed motion experiment. A slice made through the symmetry plane of the hull is coloured by fluid velocity. The free surface is depicted in blue with half opacity.

$1.5s(45kts)$, $2.0s(55kts)$. This allows the flow to be seen at the closest it gets to steady state for each flow speed. Note that the legend is consistent across all four images, and is intended more to show the progression through different inlet speeds rather than to show variation in the velocity field for each snapshot.

With each position change a minimal amount of fluid volume became "trapped" inside the hullform geometry during the instantaneous position changes. This is of small consequence, as when inside the hullform volume this fluid is excluded from contribution to the pressure profile projected from lattice nodes onto the geometry, thus having no effect on the pressure profile from which the integrated heave force is obtained. Although the solver continues to model this fluid inside the hullform, this trapped fluid constitutes a practical failure in mass conservation from the perspective of measuring fluid force on the hull. The trapped mass of fluid is a consequence of the non-physical, instantaneous position changes of the hull, and is thus reduced in effect the smaller these position changes become.

Table 3.2 details the simulation parameters used for this experiment that are not covered in Table 3.1 along with the computational resource details. Note that the HPC cluster time limit per job was 72 hours, and so the run times given neglect time spent in the cluster queue and the downtime between jobs being terminated by the time limit and being resubmitted by the user to continue.

In summary, this preliminary experiment showed that the Palabos solver was capable of simulating the heave force on an industry hullform and that prescribed hullform movement could be readily implemented. This gave confidence to and laid the groundwork for the development of heave dynamics detailed in Section 3.3.

Table 3.2: Prescribed dynamics experiment parameters and computational resource requirements.

parameter	*units*	*value*
domain size:		
x range	(m)	$[-15.0, 15.0]$
y range	(m)	$[-3.0, 2.5]$
z range	(m)	$[-5.0, 5.0]$
absorbing zone widths:		
inlet	(m)	1.0
outlet	(m)	1.0
lateral	(m)	0.5
top	(m)	0.1
fluid properties:		
density, ρ	(kg/m^3)	$1,030$
kinematic viscosity, ν	(m/s^2)	$1.0e-6$
fluid height	(m)	3.0
surface tension	(N/m)	0.0728
Palabos parameters:		
resolution	$(-)$	801
dx	(m)	0.01875
characteristic length, L_{char}	(m)	10.0
lattice velocity, u_{LB}	(m/s)	0.01
computational resources:		
HPC cluster used	$(-)$	HPC Wales, Sunbird
number of CPU cores	$(-)$	450
memory per CPU core	(Mb)	1024
total wall clock time	$(-)$	$6d:22h:10m:01s$
solver iterations	$(-)$	$421,011$
solver time	$(-)$	$6d:10h:01m:59s$
% solver time	$(-)$	92.70%
input/output time	$(-)$	$10h:35m:30s$
% input/output time	$(-)$	6.37%

3.2.1 V15 prescribed dynamics sink sensitivity study

The heave equilibrium sink positions provided by Norson Design for the previous section were estimates that are difficult to accurately assess in sea trials at full scale due to heave response, speed variability, water surface disturbances, and other practicalities that are hard to control outside of laboratory conditions. To explore how sensitive the lift force is to changes in draft, and therefore to what degree small errors in measuring draft may lead to large changes in the expected force, a repeat simulation was made at $40mm$ less sink. Excepting the $40mm$ raising of the hull position, the parameters for this simulation were identical to those detailed in Table 3.2.

The results of this simulation are plotted in Figure 3.4 along with the lift response at the original industry-provided positions. The $-40mm$ sink response matches the trends of the original position quite closely, though there is some variation between different speeds. This is most likely due to the equilibrium position also being a function of trim angle, which remains constant here. In some cases the lift response from the $-40mm$ sink case better matches the expected position. This could be a result of inaccuracies in measurement of the weight, trim angle, the sink of the boat, or combinations of all three.

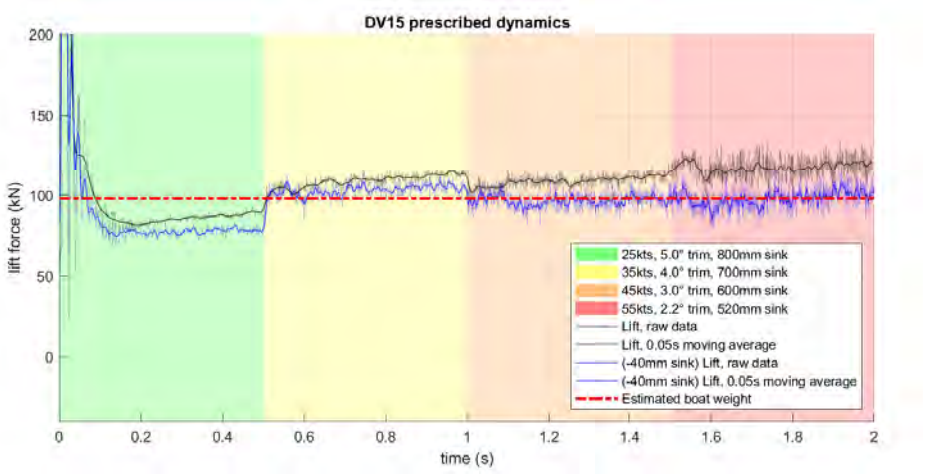


Figure 3.4: The lift force response of the V15 for the prescribed dynamics experiment at $40mm$ less sink (i.e.: 40mm higher in the water) is compared to the results at the original positions.

For the highest speed case, $55kts$, the sensitivity of lift to sink is calculated by Equation 3.1 from the average lift values between $1.75s \leq t \leq 2s$.

$$\frac{dL}{dd} = \frac{(117.6 - 98.2)}{40} \frac{kN}{mm} = 485N/mm \tag{3.1}$$

This indicates significant sensitivity in lift response for a small change in vertical position. For example a $10mm$ vertical change in position – small considering the $15m$ length of the hull – would result in a change in lift response equal to 4.9% of the boat weight.

This additional prescribed motion study indicates that the lift response is very sensitive to the vertical sink of the boat. For this reason, a model in which the hull is free to move to an equilibrium position in response to fluid forces is considerably more valuable than the static hull simulations presented so far. Addition of such a heave dynamics capability is detailed in the following Section 3.3.

3.3 Heave dynamics

Informed by feedback from industrial sponsors the goal of developing a 1 degree of freedom (DoF) heave dynamics capability for the current solver was arrived at. Though it would be a desirable feature to allow pitching motion as a second degree of freedom to allow interaction between trim angle and sink for a given speed, it was determined that the added complexity may introduce the need for dynamics and solver tuning that would otherwise not be necessary if trim angle remained fixed and heave displacement represented the sole degree of freedom. Trim angle is also affected by boat features such as the propulsion devices and trim tabs, the modelling of which is beyond the scope of this project. Even with trim angle fixed, the ability to assess the equilibrium sink of a hull was determined to be valuable to industry.

3.3.1 Explicit and implicit numerical methods

Many physical phenomena can be described using Ordinary Differential Equations (ODEs), including the heave dynamics of a planing hullform, which can be derived from first principles of Newton's second law, expressed in Equation 3.2.

$$F_z = m\ddot{z} = m\frac{d\dot{z}}{dt} \tag{3.2}$$

where F_z is the heave component of the fluid force acting on the hull, m is the mass of the hull, g is acceleration due to gravity, and z is displacement in the heave direction.

If the dependent variables of an ODE can be computed from known values, the method is said to be explicit. In a time-stepping scheme, this means that the state of the system in the following time-step is calculated using the values of the system in the current time-step. Conversely, an implicit method uses a coupled system of equations dependent on parameters from more than one time-step. Take for example a mathematical system describing physical phenomena dependent on time where the current state of the system is given by $Y(t)$ and the next state is given by $Y(t+dt)$, where dt is a small increment

in time. An explicit method would take the general form of Equation 3.3.

$$Y(t + dt) = F(Y(t)) \tag{3.3}$$

Here the state of the system at the next time-step can be defined by the state in the current time-step. This is opposed to a implicit method, which to find $Y(t + dt)$ would take the general form of Equation 3.4.

$$G(Y(t), Y(t + dt)) = 0 \tag{3.4}$$

Here the state of the system for the next time-step is dependent both on the current time-step state and the state in the next time-step. For this reason implicit methods are typically more difficult to implement. Explicit methods are comparatively easier to implement, though in some cases will require very small time-steps to remain numerically stable in the face of stiff systems [116].

Based on this, the Forward Euler method was chosen to solve the dynamics in anticipation of a non-stiff response, with the alternative of exploring implicit methods if numerical instability was encountered. This conferred the advantage of simplicity and rapid implementation for turn-around to industry. The disadvantage of being forced to use smaller time-steps versus an implicit method is reduced by the insight given by earlier testing in Section 3.2, where it was discovered that small time-steps in the dynamics would be desirable to minimise water trapped in the hull geometry when position changes are made.

3.3.2 Forward Euler method for heave dynamics

For a time-stepping scheme where the n^{th} time-step is given by t_n, for which the system state is given by y_n such that $y_n = y(t = t_n)$, the forward Euler method can be written as in Equation 3.5.

$$y_{n+1} = y_n + hf(y_n, t_n) \tag{3.5}$$

where h is the step size such that $h = t_n - t_{n-1}$. The forward Euler method is a based on a truncated Taylor series expansion such that expanding y in the neighbourhood of $t = t_n$ gives Equation 3.6.

$$y(t_n + h) \equiv y_{n+1} = y(t_n) + h\frac{dy}{dt}|_{t_n} + O(h^2) = y_n + hf(y_n, t_n) + O(h^2) \quad (3.6)$$

Thus the step error is proportional to the square of the step size, $O(h^2)$, and the method is considered first order.

From Equation 3.2 Newton's second law can be expressed as an ODE by rearranging for velocity ($\dot{z}$):

$$\Delta\dot{z} = \frac{F_z}{m}\Delta t \quad (3.7)$$

Expressing this in Euler time-stepping terms:

$$\Delta\dot{z}_i = \frac{F_i}{m}\Delta t \quad (3.8)$$

Then the velocity for the next time steps is calculated by summing the previous velocity with the incremental change in this step:

$$\dot{z}_{i+1} = \dot{z}_i + \Delta\dot{z}_i \quad (3.9)$$

For the position at the next timestep, given the following:

$$\dot{z} = \frac{dz}{dt} \rightarrow \Delta z_i = \dot{z}_i\Delta t \quad (3.10)$$

Then similarly to Equation 3.9, the position in the next time-step can be given by:

$$z_{i+1} = z_i + \Delta z_i \quad (3.11)$$

In the case of heave force acting on the hull, F_z is the resultant force of the weight force of the boat and the vertical heave component of the force exerted by the fluid on the hull such that:

$$F_z = \ddot{z} = F_{z_{palabos}} - mg \quad (3.12)$$

where $F_{z_{palabos}}$ is the vertical component of the fluid force acting on the boat

hull according to the Palabos simulation for this dynamics time-step, m is the mass of the boat, and g is acceleration due to gravity. Note that up is considered the positive direction.

Thus by substituting Equation 3.7 into Equation 3.12 the change in velocity for this time-step is given by:

$$\Delta \dot{z}_i = \frac{F_{z_{palabos}} - mg}{m} \Delta t \tag{3.13}$$

This can be substituted into Equation 3.9 to then solve Equation 3.11. Thus with initial values determined by the user, typically as:

$$z(t=0) = 0 \qquad and \qquad \dot{z}(t=0) = 0 \tag{3.14}$$

Thus the forward Euler method for the heave dynamics is fully defined. This process is carried out in the Matlab subfunction `funcMoveBoat.m`, detailed in Appendix A.2.6. This subfunction takes as inputs the time-step size (`dt`), Palabos-calculated heave force acting on the boat (`y_forceAvg`), mass of the boat (`m_boat`), and values of z_i and $\dot{z}_i$ (`z, z_dot`); it returns the z_{i+1} and $\dot{z}_{i+1}$ values, and the position change for the hull geometry (`z_diff`).

3.3.3 Heave dynamics implementation

The forward Euler dynamics requires the hydrodynamic forces acting on the hull ($F_{z_{palabos}}$, Equation 3.13) to evaluate the hull movement. The hullform geometry would then be moved incrementally in the Palabos simulations according to the dynamics solution. To do this, a Palabos simulation is needed for each iteration of the dynamics solution.

The approach taken to achieve this was an expansion on the method used in Section 3.1, where a Matlab script is submitted to the HPC cluster on a single core with the remaining cores reserved for running the Palabos executable.

This approach can be thought of as two concurrent simulations: a dynamics simulation solved using the forward Euler method inline within the Matlab script, and a Palabos simulation carried out by the `boatHullFormSolver` executable. There are, therefore, two separate iteration counts for each simu-

lation.

In the following explanation, the working directory refers to the "hub" directory containing the main executables. The contents of this directory are as follows:

- `boatHullFormSolver` – the executable for the Palabos hullform solver, see Appendix A.3.1. Compiled using the `boatHullFormSolver.cpp` source using the `Makefile`.
- `jobSubmitter.m` – the Matlab job submission and dynamics script, see Appendix A.2.1. Call on several Matlab subfunctions, see Appendices A.1.2-A.1.8.
- `<hullform>.stl` – the hullform geometry to be simulated, in `.stl` format.
- `batchfile.job` – the batchfile for submission of the `jobSubmitter.m` to the HPC cluster for execution. The contents of this will depend on the HPC hardware and queue manager.
- `params_template.xml` – an `.xml` formatted file containing the user defined inputs for Palabos simulations.
- `\tmp` – a temporary directory that the simulation outputs are stored in.
- `checkpoint_xxxxxxxx_x.plb/dat` and `continue.xml` – in the case that the simulation is being started from a previously checkpointed simulation (at iteration number denoted by `xs`), these files left by the previous simulation will be needed in the working directory.

The Palabos solver files and libraries then lie one level above the working directory (path relative to working directory: `../palabos/`).

The Matlab submission script (`jobSubmitter.m`) has a number of user defined input parameters at its head, and also takes user input parameters from `params_template.xml`. Using these, the timestep of the Palabos simulation is calculated by Equation 3.15.

$$dt_{palabos} = \frac{u_{LB} L_{char}}{u_{ref}(resolution - 1)} \tag{3.15}$$

where the right hand terms are Palabos input values discussed in Section 2.1. Note that for this section the dt terms will be subscripted to denote whether they belong to the Palabos or dynamics simulations. As the user also defines the physical time to be simulated and time-step for the dynamics ($maxTime$ and $dt_{dynamics}$), the number of Palabos iterations that need to be run to complete a single dynamics iteration can be calculated.

The Matlab submission script then iterates until the user defined time limit is reached, with each iteration submitting a complete Palabos simulation. When the Palabos simulation for the current dynamics time-step reaches the number of iterations required it checkpoints, saving the outputs and current state of the simulation in the working and `/tmp` directory. The Matlab submission script then reads the force values from the Palabos simulation outputs as an input for the dynamics calculations, using the forward Euler method described in Section 3.3.2.

The dynamics calculations described in Section 3.3.2 are carried out in-line in the Matlab submission script, and the resulting displacement of the hull for the next time-step is calculated. Several subfunctions (Appendices A.2.4, A.2.5, A.2.6, and A.2.7) then handle the translation of the `.stl` file to this new position for the next dynamics time-step. The dynamics loop iterates, and new values for the next Palabos simulation are streamed to the `params.xml` file prior to submission.

This process is visually described in further detail in Figure 3.5. The full Matlab submission script (`jobSubmitter.m`) is available in Appendix A.2.1, with the Matlab subfunctions in Appendices A.1.2-8.

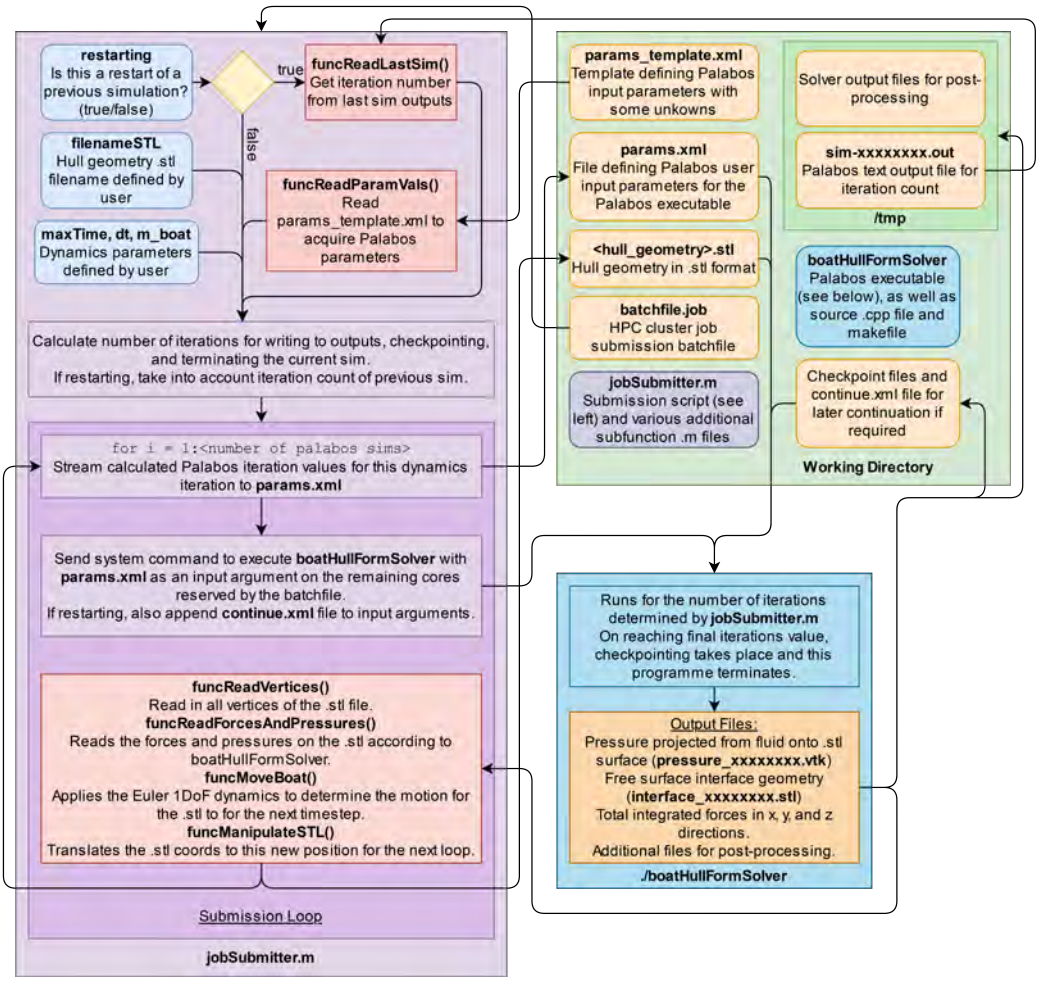


Figure 3.5: A flowchart detailing the workflow of the dynamics implementation. The process begins with the submission of the `batchfile.job` file in the working directory to the HPC cluster job scheduler.

3.4 V15 dynamic hullform drop test

As an initial test of the dynamics an extreme case of heave motion was simulated: the V15 geometry was dropped from a high initial position (given the flow speed), and allowed to settle to an equilibrium position. The hull was positioned at a 5° trim angle and initiated at only 560mm sink below the waterline measured at deepest part of the transom in a $10m/s$ flow. For context, industry data indicated the equilibrium sink measured at the same point in a $25kts$ ($12.86m/s$) flow would be $760mm$, so although the hull is partly wetted on initiation the hull is essentially being dropped relative to its estimated equilibrium – significant motion was expected, akin to severe slamming conditions. This test sought to identify whether the method described in Section 3.3.3 was capable of accurate solving of the hydrodynamic forces on the boat and, subsequently, accurate and robust capture of the dynamic motion in response to these forces.

The input parameters and computational resources used for the simulation are detailed in Table 3.3. In Figure 3.8 the simulation is visualised from a side perspective of the hull (pink) with the free surface (blue, 50% opacity) displayed to show the complex free surface behaviour being modelled. Figure 3.10 shows this again, with a slice through the fluid along the centreline of the hull coloured by velocity to reveal complexity throughout the fluid. The heave motion response is plotted in Figure 3.6.

From Figures 3.10 and 3.6 it can be seen that the dynamics method worked as intended, as the hull settles to an equilibrium steady-state heave position at a reasonable amount of sink. The drop height was quite significant, as the whole hull is temporarily submerged in the initial oscillation as seen in Figure 3.10. In these snapshots, velocity magnitude is plotted on a slice made through the fluid domain along the boat centreline, revealing the complex flow being solved by the Palabos solver that could not be accomplished in full fidelity by an analytical method. The free surface is shown in 50% opacity blue, revealing complex free surface flow with droplets and entrained air.

Although this kind of motion is possible in rough sea states, and may occur after the hull pierces a wave and falls into the wave trough, such motion can be

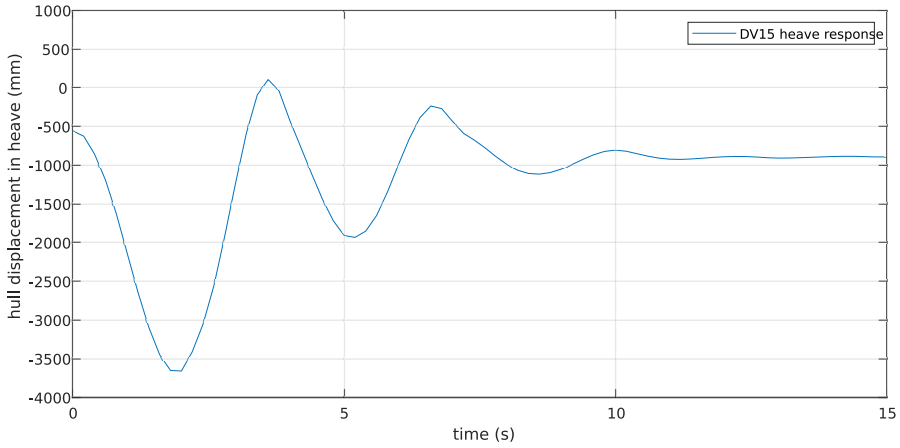


Figure 3.6: The heave motion response of the V15 hull is plotted, showing the trend towards a steady-state equilibrium position. The y-axis is based on sink measured at the lowest point of the transom. The hull is dropped from a position of $560mm$ sink, and reaches heave equilibrium at a position of $898mm$ sink. Note the infrequent output of position data causes a low resolution curve; this is resolved in later plots of heave motion.

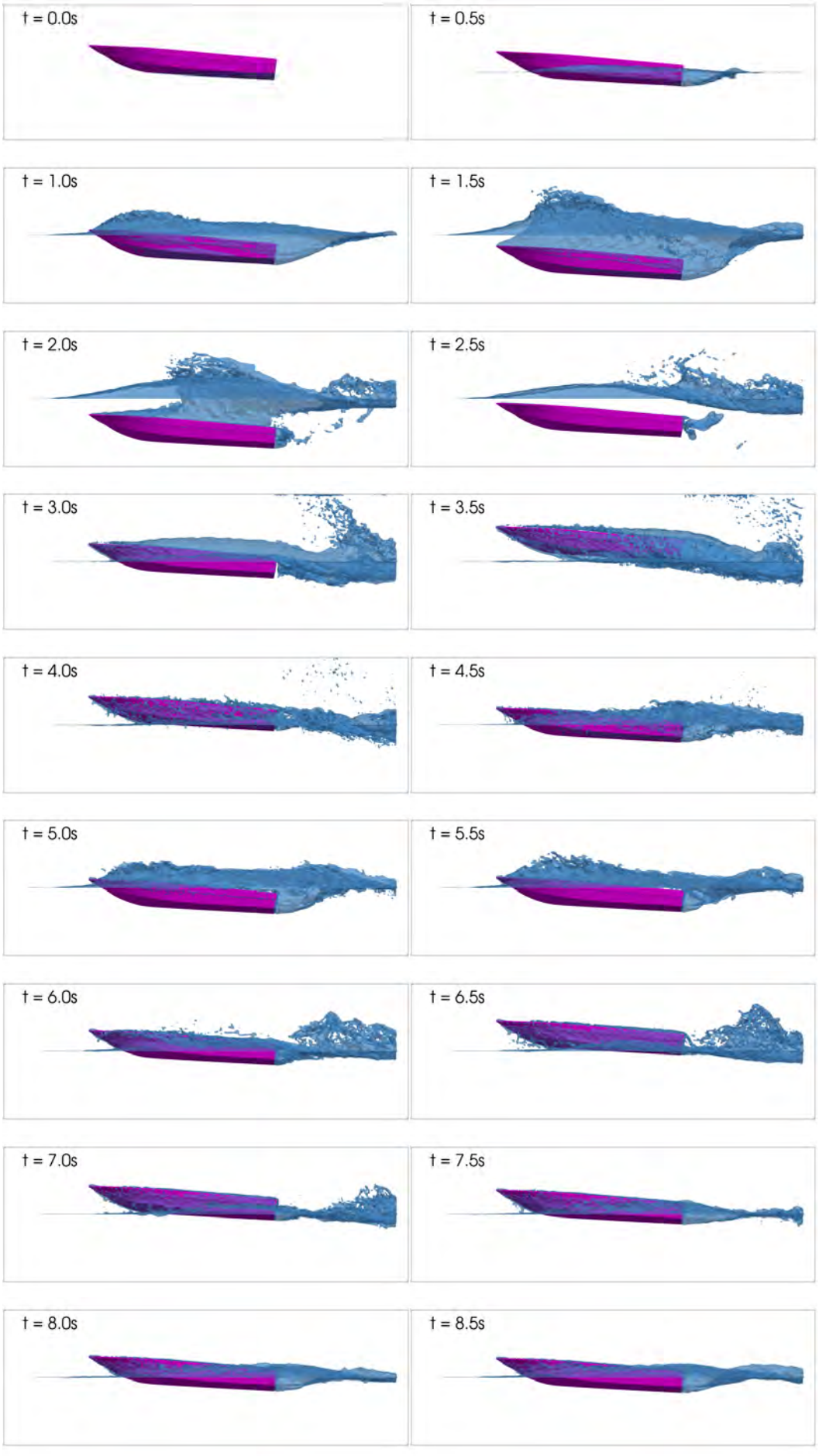
considered on the bounds of the V15 performance envelope. This experiment shows that the dynamics method is robust enough to handle hull motion far beyond the necessary magnitudes to evaluate heave equilibrium in calm water.

Exact validation of the final position arrived at cannot be made from this test case as the flow speed was arbitrarily set at $10m/s$ and did not exactly match the given industry data, for which the closest estimate is only available at $12.86m/s$ $(25kts)$. The steady-state sink in this experiment was $898mm$ compared to $760mm$ estimated by industry data at the same $5°$ trim angle. Given how close these values are, and that the slower speed would be expected to yield a greater sink, as is seen, this experiment gave reasonable confidence in the accuracy of the heave force predictions on top of demonstrating robustness in the face of large heave position changes.

To provide more rigorous validation against the industry data, a similar set of simulations are performed in Section 3.5 for each of the conditions given for the V15 by industry.

Table 3.3: V15 dynamics drop test parameters and computational resource requirements.

parameter	*units*	*value*
domain size:		
x range	(m)	$[-20.0, 10.0]$
y range	(m)	$[-5.0, 5.0]$
z range	(m)	$[-4.0, 4.0]$
absorbing zone widths:		
inlet	(m)	1.0
outlet	(m)	1.0
lateral	(m)	0.5
top	(m)	0.0
fluid properties:		
density, ρ	(kg/m^3)	$1,000$
kinematic viscosity, ν	(m/s^2)	$1.0e-6$
fluid height	(m)	5.0
surface tension	(N/m)	0.0728
Palabos parameters:		
resolution	$(-)$	301
dx	(m)	0.05
characteristic length, L_{char}	(m)	15.0
lattice velocity, u_{LB}	(m/s)	0.01
dynamics parameters:		
hullform geometry	$(-)$	V15
trim angle	$(^\circ)$	5
initial sink at transom	(mm)	560
hull mass	(kg)	$9,000$
dynamics time-step size	(s)	$5e-3$
computational resources:		
HPC cluster used	$(-)$	HPC Wales, Sunbird
number of CPU cores	$(-)$	500
memory per CPU core	(Mb)	$4,000$
total wall clock time	$(-)$	$3d:20h:19m:40s$


t = 0.0s
t = 0.5s
t = 1.0s
t = 1.5s
t = 2.0s
t = 2.5s
t = 3.0s
t = 3.5s
t = 4.0s
t = 4.5s
t = 5.0s
t = 5.5s
t = 6.0s
t = 6.5s
t = 7.0s
t = 7.5s
t = 8.0s
t = 8.5s

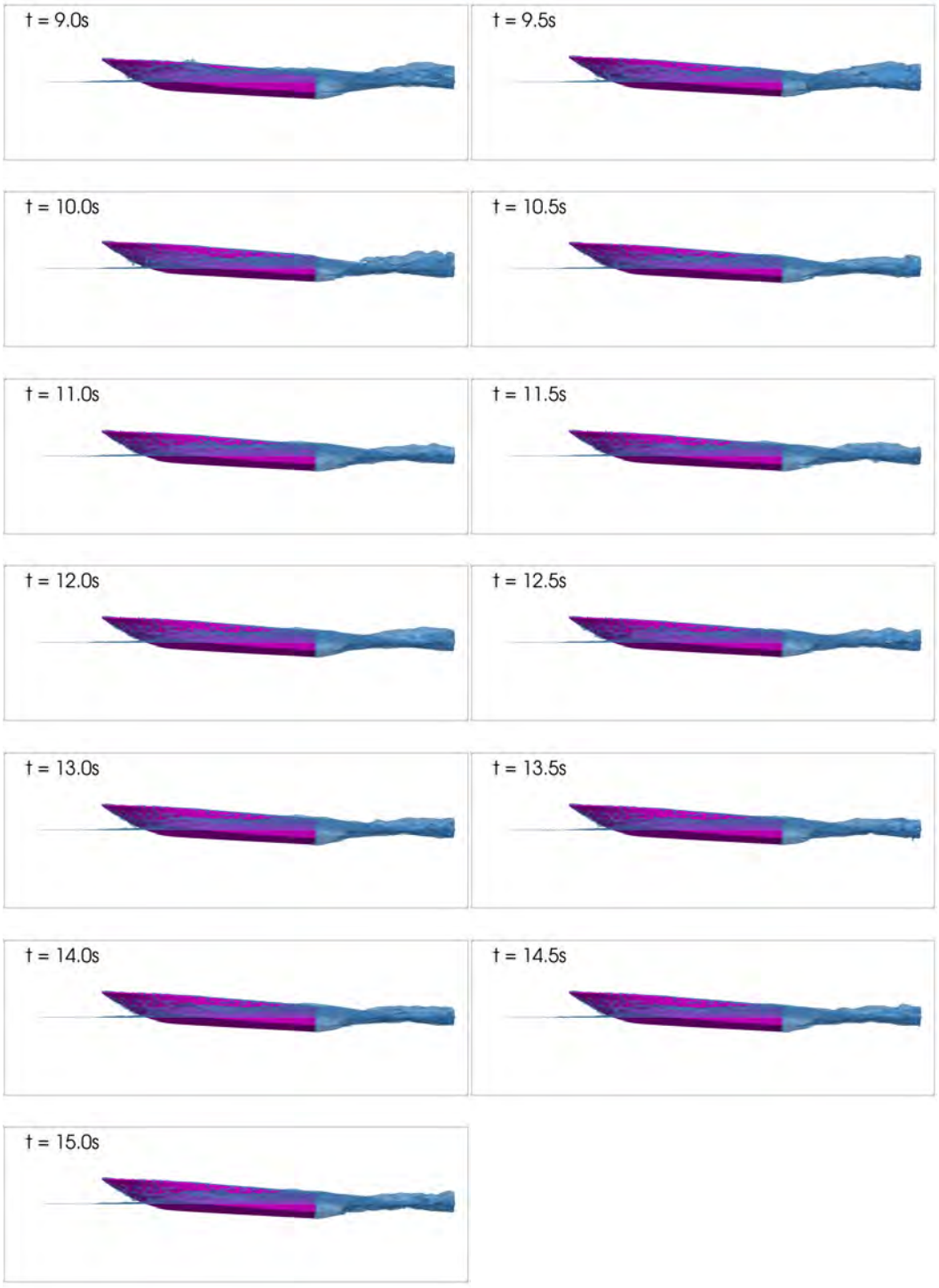
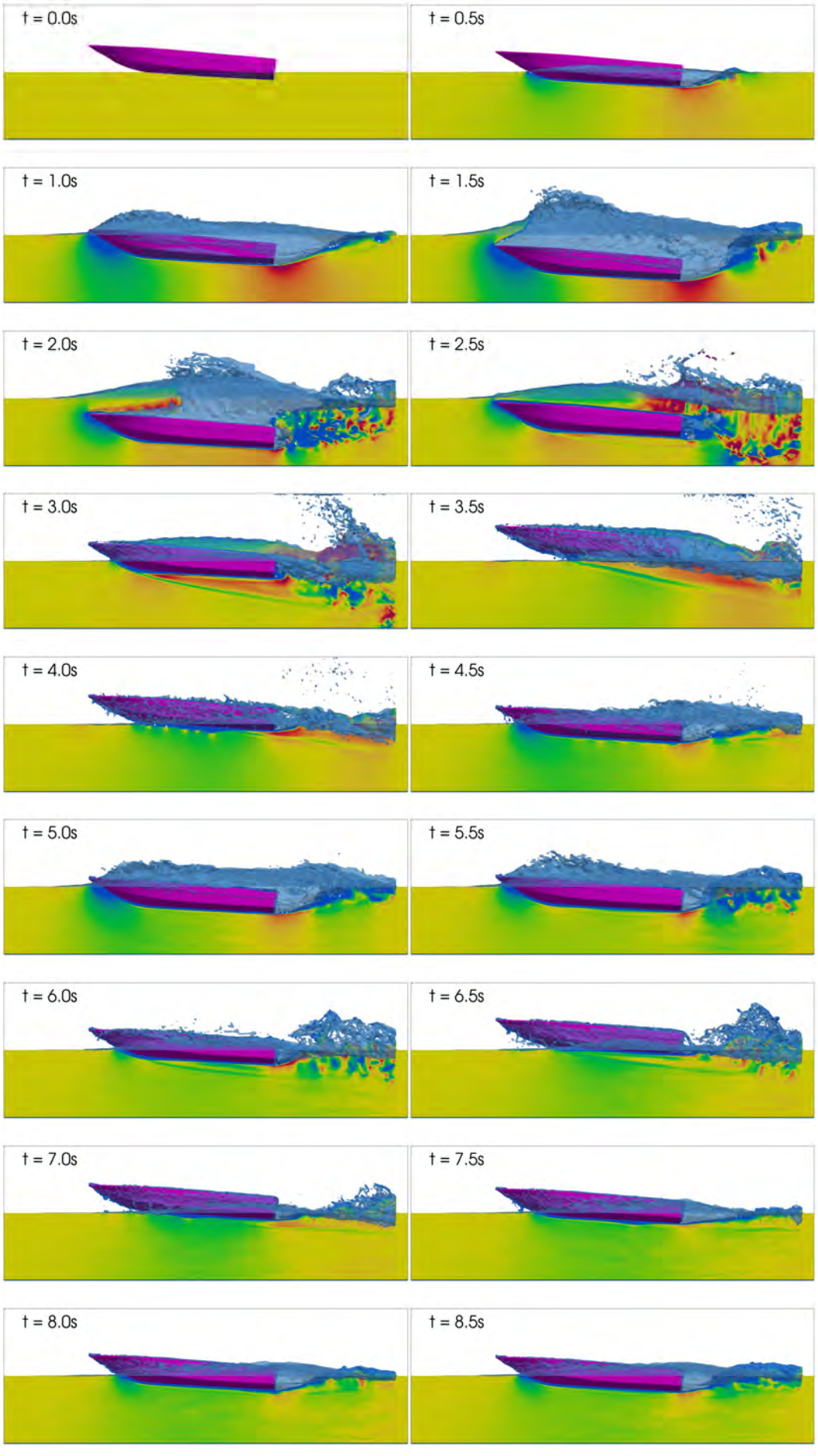


Figure 3.8: The V15 drop test is seen from the side to illustrate heave motion in snapshots of $0.5s$ intervals. The hullform is pink, and the free surface blue with 50% opacity.


t = 0.0s
t = 0.5s
t = 1.0s
t = 1.5s
t = 2.0s
t = 2.5s
t = 3.0s
t = 3.5s
t = 4.0s
t = 4.5s
t = 5.0s
t = 5.5s
t = 6.0s
t = 6.5s
t = 7.0s
t = 7.5s
t = 8.0s
t = 8.5s

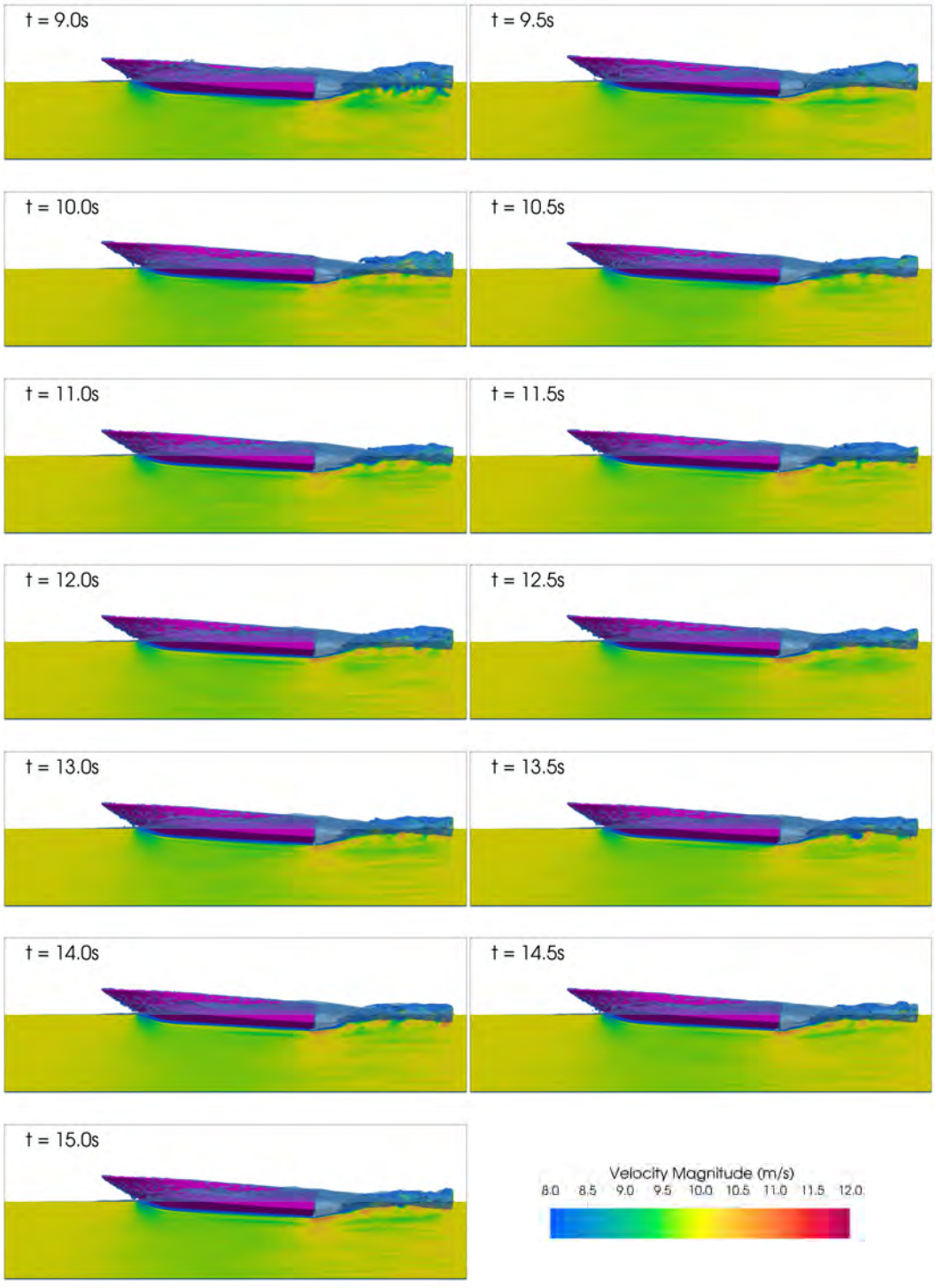


Figure 3.10: The V15 drop test is seen from the side to illustrate heave motion in snapshots of $0.5s$ intervals. The hullform is pink, and the free surface blue with 50% opacity. A slice through fluid domain along the centreline of the hull is coloured by fluid velocity.

3.5 Dynamic validation against industry data for the V15 hullform

To validate the ability of the heave dynamics method to assess the equilibrium heave position of a hull, simulations were run matching the industry data used in Section 3.2, reprinted in Table 3.4. Due to computer resource and time limits, the $25kts$ case was skipped, as the case in Section 3.4 was considered to sufficiently match the conditions with a satisfactory equilibrium position. Validation at the higher speeds was considered more valuable by industry, as designs would typically be optimised for near-maximum design speed.

A dynamics simulation was run for each of the cases in Table 3.4, similarly to the prescribed dynamics experiment, though in this case the hulls are allowed to move freely in heave. The validity of the simulations will then be quantified by the steady-state heave position they reach compared to industry sink estimates in Table 3.4.

Table 3.4: Industry-provided V15 planing equilibrium position values for various speeds.

Speed (kts)	Speed (m/s)	Trim (°)	Draft (mm)
35	18.01	4.0	700
45	23.15	3.0	600
55	28.29	2.2	520

The force and displacement response for each of these cases are plotted in the following respective subsections. As the vertical motion was minimal due to the hull being initiated at near-equilibrium positions, no side-on visualisations are shown as with the drop test in Section 3.4. Instead, a general snapshot visualisation of the free surface wake at equilibrium is shown in Section 3.5.4 for each case. Simulation parameters and resource usage are also detailed in the accompanying tables.

3.5.1 Dynamic validation results – 35kts, 4.0° trim

The heave displacement and velocity responses are shown in Figure 3.11. The heave displacement is zeroed to the predicted equilibrium position of 700mm. The displacement initially surges and over-corrects in the first $0.7s$, before temporarily settling at a slightly surged state of approximately $+80mm$ between $0.7s - 1.5s$. It then settles to a steady value close to the industry-estimated equilibrium sink value, indicating close agreement with industry data. From 3.5s–4.7s the V15 was judged to have reached equilibrium; the heave response averaged a mean value of +8.4mm with a standard deviation of ±2.5mm during this period. This simulation took 144 hours of wall-clock time to run on 700 cores.

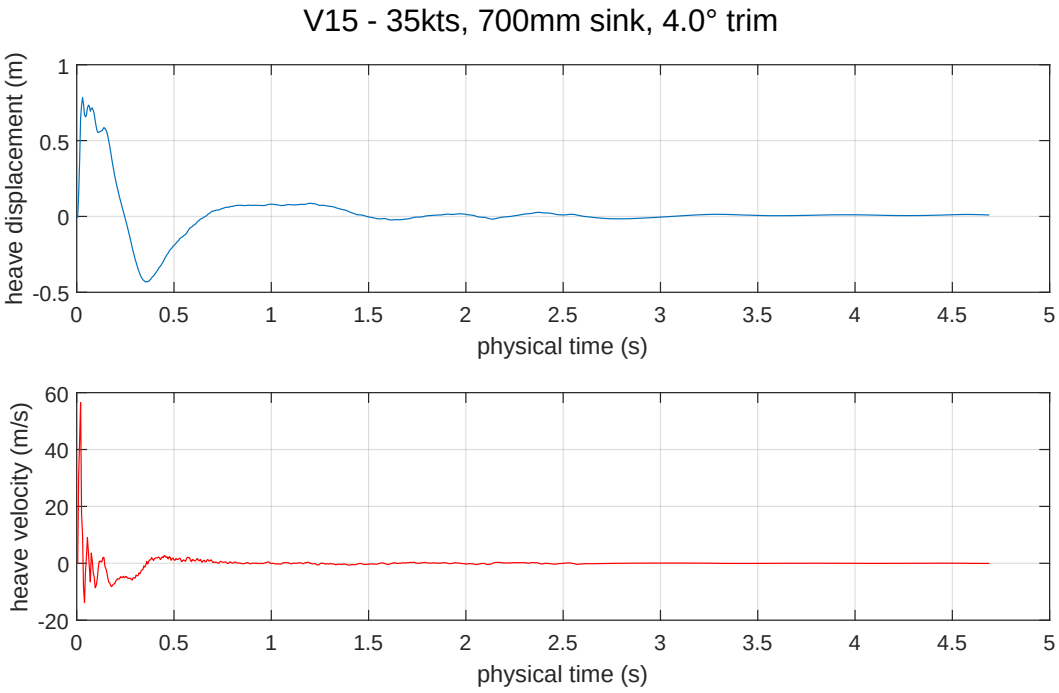


Figure 3.11: The heave displacement (blue) and heave velocity (red) are plotted against time for the $35kts$ scenario. The heave displacement is zeroed to the industry estimated equilibrium value of $700mm$ sink measured at the deepest part of the transom.

Table 3.5: V15 dynamics: 35kts, 4.0° trim – simulation parameters and computational resources.

parameter	*units*	*value*
domain size:		
x range	(m)	$[-20.0, 10.0]$
y range	(m)	$[-5.0, 5.0]$
z range	(m)	$[-4.0, 4.0]$
absorbing zone widths:		
inlet	(m)	1.0
outlet	(m)	1.0
lateral	(m)	0.5
top	(m)	0.0
fluid properties:		
Fr	$(-)$	3.31
Re	$(-)$	36,000,000
density, ρ	(kg/m^3)	$1,030$
kinematic viscosity, ν	(m/s^2)	$1.0e-6$
fluid height	(m)	5.0
surface tension	(N/m)	0.0728
Palabos parameters:		
resolution	$(-)$	301
dx	(m)	0.0333
dt	(s)	$1.8519e-05$
characteristic length, L_{char}	(m)	10.0
lattice velocity, u_{LB}	(m/s)	0.01
dynamics parameters:		
hullform geometry	$(-)$	V15
trim angle	$(^\circ)$	4
initial sink at transom	(mm)	700
hull mass	(kg)	$10,000$
dynamics time-step size	(s)	0.005
computational resources:		
HPC cluster used	$(-)$	HPC Wales, Sunbird
number of CPU cores	$(-)$	700
memory per CPU core	(Mb)	$8,000$
total wall clock time	$(-)$	$144h:00m:00s$

3.5.2 Dynamic validation results – 45kts, 3.0° trim

The heave displacement and velocity responses are shown in Figure 3.12. The heave displacement is zeroed to the predicted equilibrium position of 600mm. The trends closely match the previous 35kts case: the displacement initially surges and over-corrects in the first 0.7s, before temporarily settling at a slightly surged state of approximately +80mm between 0.7s – 1.2s. From 2.5s–3.0s the V15 was judged to have reached equilibrium; the heave response averaged a mean value of +4.0mm with a standard deviation of ±7.0mm during this period. This simulation took 144 hours of wall-clock time to run on 700 cores.

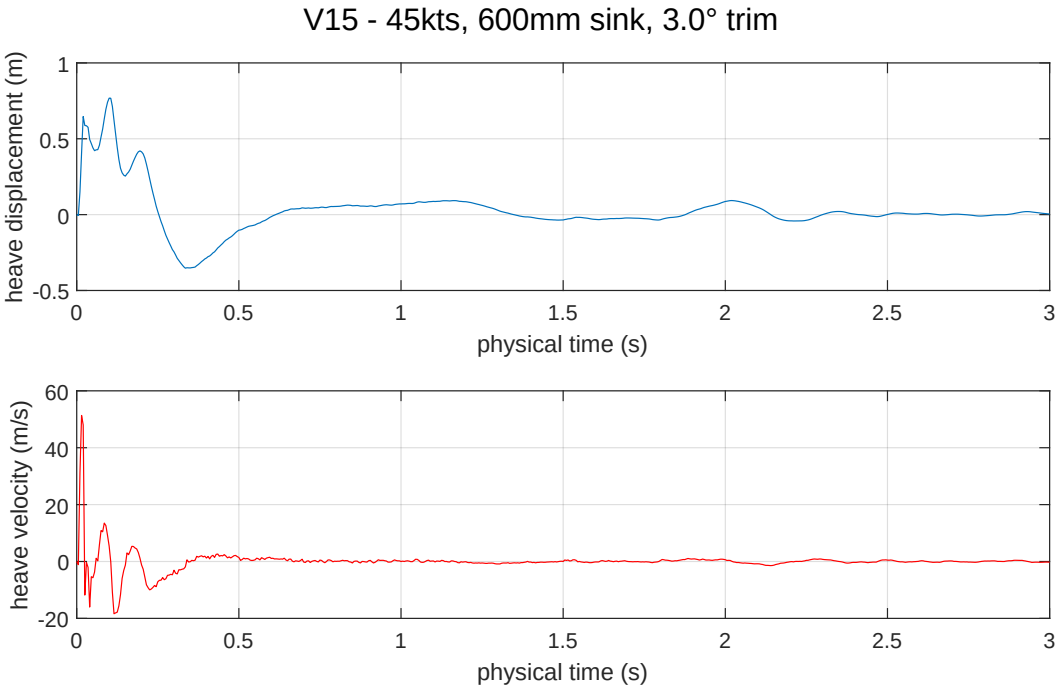


Figure 3.12: The heave displacement (blue) and heave velocity (red) are plotted against time for the 45kts scenario. The heave displacement is zeroed to the industry estimated equilibrium value of 600mm sink measured at the deepest part of the transom.

Table 3.6: V15 dynamics: 45kts, 3.0° trim – simulation parameters and computational resources.

parameter	*units*	*value*
domain size:		
x range	(m)	$[-20.0, 10.0]$
y range	(m)	$[-5.0, 5.0]$
z range	(m)	$[-4.0, 4.0]$
absorbing zone widths:		
inlet	(m)	1.0
outlet	(m)	1.0
lateral	(m)	0.5
top	(m)	0.0
fluid properties:		
Fr	$(-)$	4.26
Re	$(-)$	46,300,000
density, ρ	(kg/m^3)	$1,030$
kinematic viscosity, ν	(m/s^2)	$1.0e-6$
fluid height	(m)	5.0
surface tension	(N/m)	0.0728
Palabos parameters:		
resolution	$(-)$	301
dx	(m)	0.0333
dt	(s)	$1.4399e-05$
characteristic length, L_{char}	(m)	10.0
lattice velocity, u_{LB}	(m/s)	0.01
dynamics parameters:		
hullform geometry	$(-)$	V15
trim angle	$(^\circ)$	3
initial sink at transom	(mm)	600
hull mass	(kg)	$10,000$
dynamics time-step size	(s)	0.005
computational resources:		
HPC cluster used	$(-)$	HPC Wales, Sunbird
number of CPU cores	$(-)$	700
memory per CPU core	(Mb)	$8,000$
total wall clock time	$(-)$	$144h:00m:00s$

3.5.3 Dynamic validation results – 55kts, 2.2° trim

The heave displacement and velocity responses are shown in Figure 3.12. The heave displacement is zeroed to the predicted equilibrium position of 520mm. The displacement trends do not match as closely as the previous two cases: there is no notable over-surge, but rather an over-sink period that slowly corrects towards an equilibrium near the industry-estimated value. Once reached, the equilibrium displacement is noticeably more consistent than the $45kts$ case. From 5.50s–7.75s the V15 was judged to have reached equilibrium; the heave response averaged a mean value of +73.0mm with a standard deviation of ±1.7mm during this period. This simulation took 216 hours of wall-clock time to run on 700 cores.

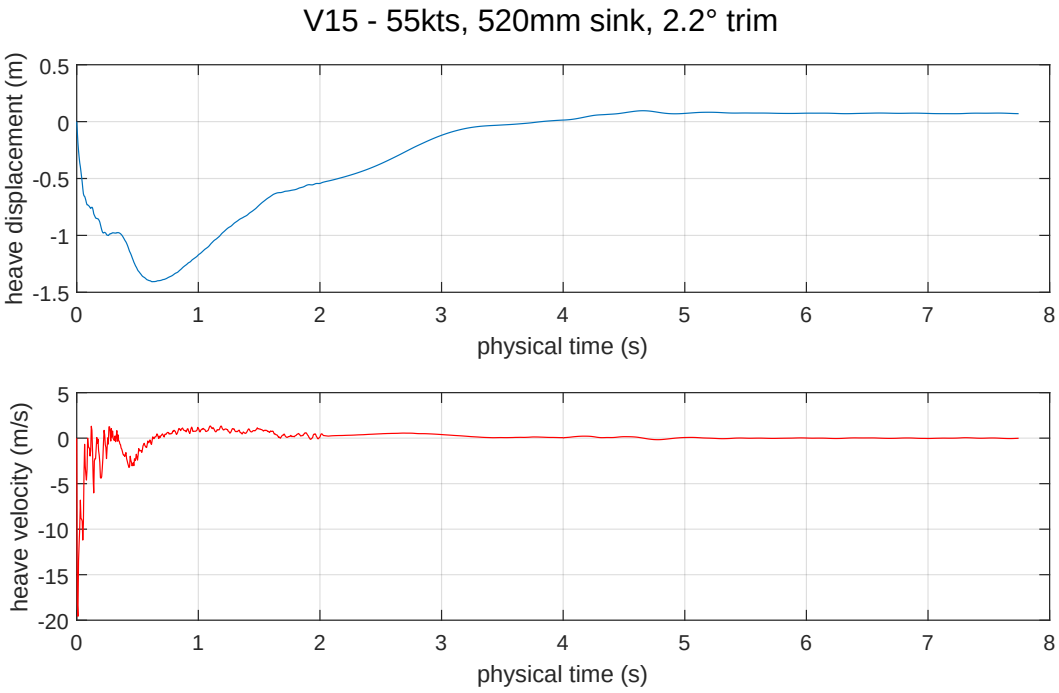


Figure 3.13: The heave displacement (blue) and heave velocity (red) are plotted against time for the $55kts$ scenario. The heave displacement is zeroed to the industry estimated equilibrium value of $520mm$ sink measured at the deepest part of the transom.

Table 3.7: V15 dynamics: 55kts, 2.2° trim – simulation parameters and computational resources.

parameter	*units*	*value*
domain size:		
x range	(m)	$[-20.0, 10.0]$
y range	(m)	$[-5.0, 5.0]$
z range	(m)	$[-4.0, 4.0]$
absorbing zone widths:		
inlet	(m)	1.0
outlet	(m)	1.0
lateral	(m)	0.5
top	(m)	0.0
fluid properties:		
Fr	$(-)$	5.21
Re	$(-)$	56,580,000
density, ρ	(kg/m^3)	$1,030$
kinematic viscosity, ν	(m/s^2)	$1.0e-6$
fluid height	(m)	5.0
surface tension	(N/m)	0.0728
Palabos parameters:		
resolution	$(-)$	301
dx	(m)	0.0333
dt	(s)	$1.1783e-05$
characteristic length, L_{char}	(m)	10.0
lattice velocity, u_{LB}	(m/s)	0.01
dynamics parameters:		
hullform geometry	$(-)$	V15
trim angle	$(^\circ)$	2.2
initial sink at transom	(mm)	520
hull mass	(kg)	$10,000$
dynamics time-step size	(s)	0.005
computational resources:		
HPC cluster used	$(-)$	HPC Wales, Sunbird
number of CPU cores	$(-)$	700
memory per CPU core	(Mb)	$8,000$
total wall clock time	$(-)$	$216h:00m:00s$

3.5.4 Heave dynamics convergence to equilibrium compared across speeds

A comparison of the displacement trends from each of the three dynamics simulations is presented graphically in Figure 3.14. The significant difference between the $55kts$ and the two slower simulations can be seen: the $35kts$ and $45kts$ simulations follow very similar trends and converge to equilibrium in almost the same $2s$ time frame; the $55kts$ case takes much longer to converge, though does not experience a significant overshoot as the other two cases do. The $55kts$ case can be seen to have the greatest variance from the estimated equilibrium position. The equilibrium position results for each case are also compared numerically in Table 3.8. In Figures 3.15 and 3.16 the free surface deformations at the final computed time-step for each speed case are compared.

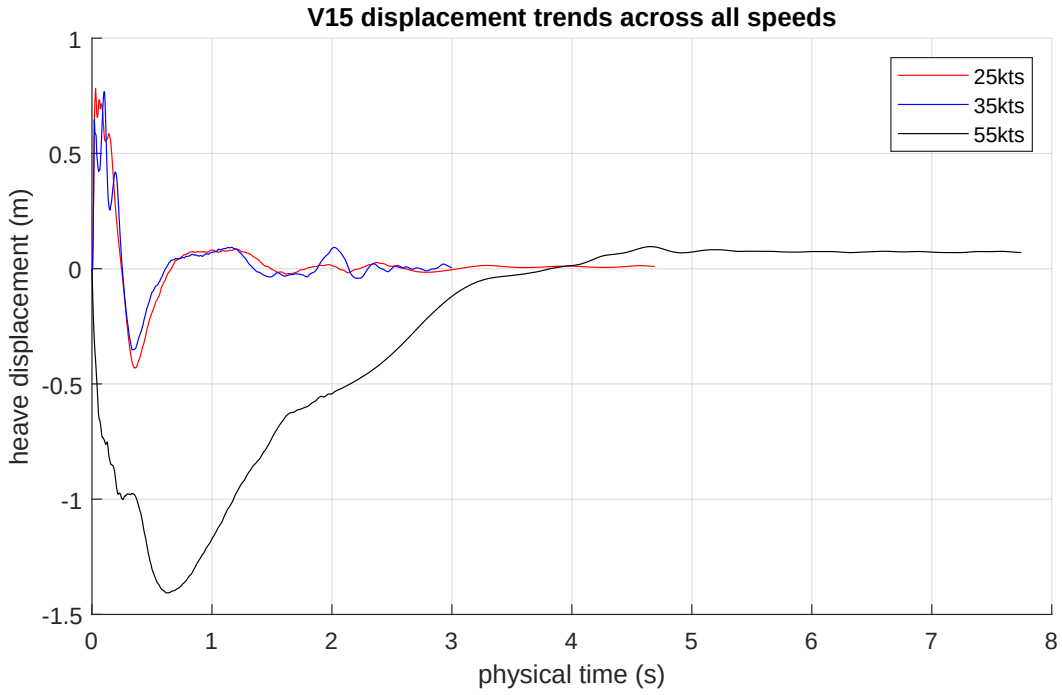
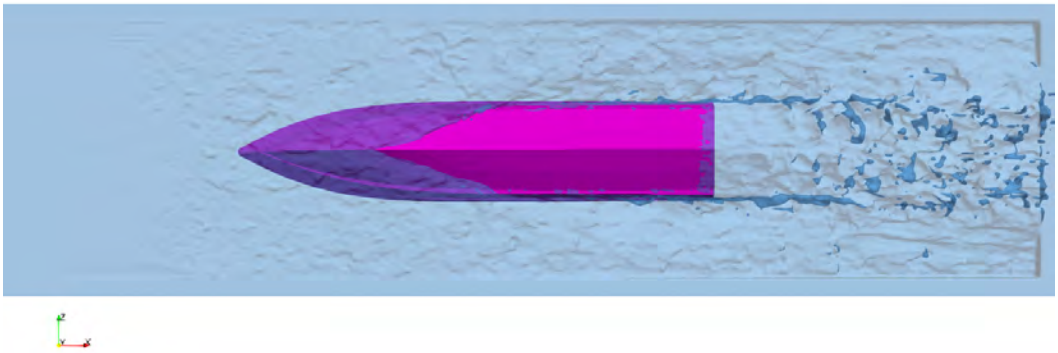


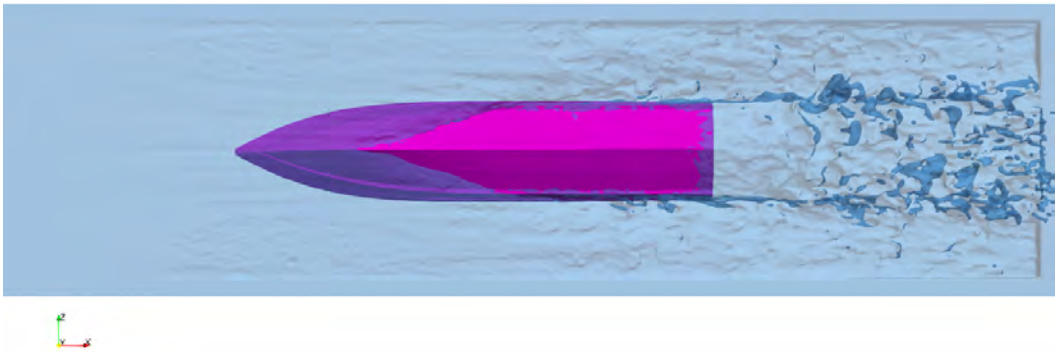
Figure 3.14: The heave displacement for the dynamics simulations are plotted together. The convergence to equilibrium patterns and time taken can be seen.

Table 3.8: The mean average sink during the equilibrium period of each simulation is presented with a standard deviation and percentage variance of the mean value against the industry-predicted value.

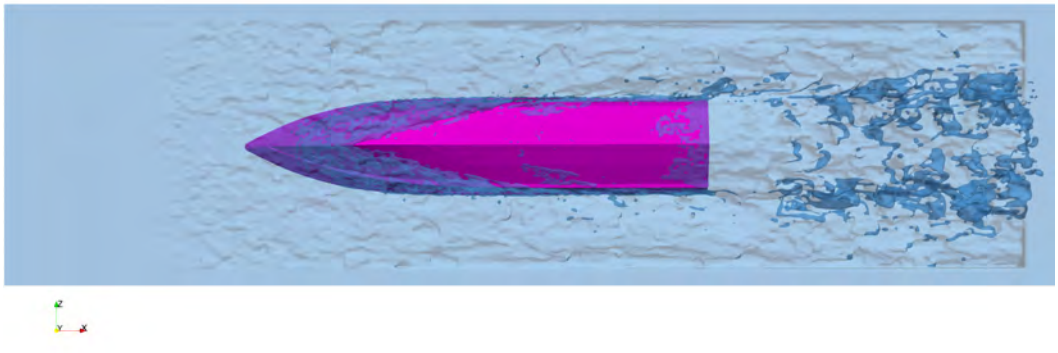
Speed (kts)	Industry-estimated sink (mm)	Mean average numerical sink (mm)	Period of averaging (s)	Percentage difference relative to industry values
25	800	886.0±7.6	12.0–15.0	10.75%
35	700	691.7±2.5	3.50–4.70	-1.19%
45	600	596.0±7.0	2.50–3.00	-0.67%
55	520	447.0±1.7	5.50–7.75	-14.0%



(a) 35kts, 4° trim.

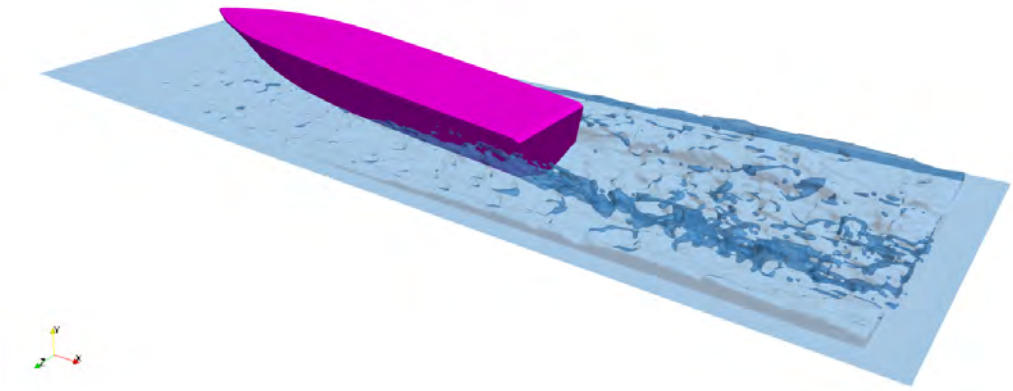


(b) 45kts, 3° trim.

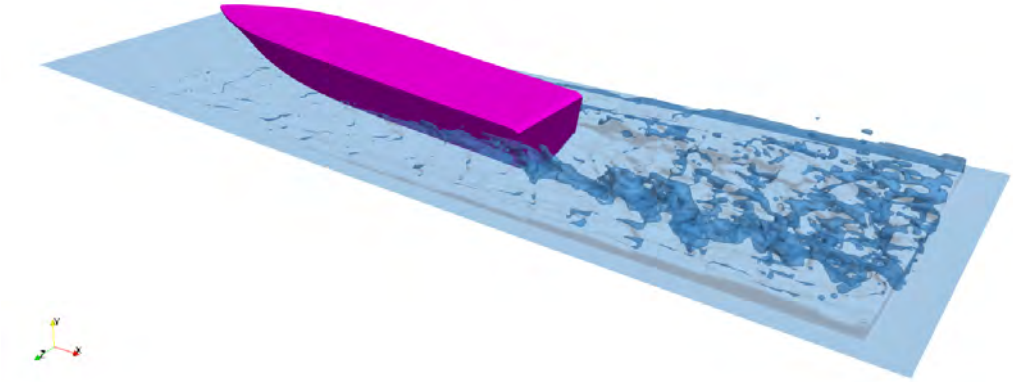


(c) 55kts, 2.2° trim.

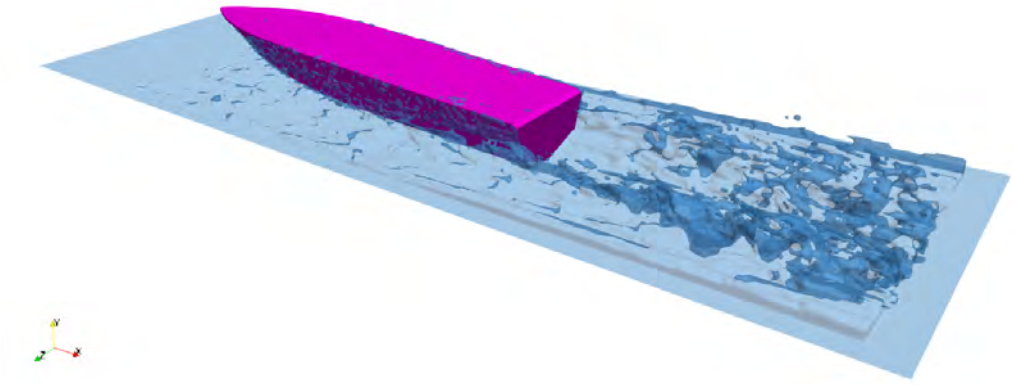
Figure 3.15: The final timestep of each of the simulations is shown from below, with hull geometry (pink) and free surface (blue with 50% opacity) shown.



(a) 35kts, 4° trim.



(b) 45kts, 3° trim.



(c) 55kts, 2.2° trim.

Figure 3.16: The final timestep of each of the simulations is shown from behind, with hull geometry (pink) and free surface (blue with 50% opacity) shown.

3.6 Heave dynamics conclusions and recommendations for future work

The implementation and validation of a heave dynamics capability for the Palabos solver has been demonstrated. The validation against industry data shows good convergence in the $35kts$ and $45kts$ speed regimes ($3.31 \leq Fr \leq 4.26$, using boat beam as reference length), and although the higher $55kts$ speed showed evidence of convergence to a reasonable equilibrium position, this could not be confirmed due to high computational cost and time limitations. The drop test in Section 3.4 exhibited the robustness of the dynamics capabilities as the hull is dropped from a height near the limits of the design conditions expected in service, and successfully returns to an equilibrium position.

This approach does incur high computational costs, as the grid size needs to be driven towards a grid independent value (see Section 2.3) while also keeping the domain size large enough for phenomena such as wake, spray, and bow waves, all while being restricted to a uniform global grid size. This cost is quantified in the grid independence study of Section 2.3, where a four times increase in the resolution parameter results in a 19,346% increase in the compute time required for a unit of physical time simulated. For this reason, a significant step forward in future development of this work to make the method more practical for industry use would be the implementation of local grid refinement. Additionally, implementation of the dynamics natively in the C++ script rather than requiring the Matlab "wrap" solution would be desirable: doing so would allow some degree of parallelism in the dynamics process rather than the constantly reserved single core for running the Matlab script, which spends most of the runtime waiting. The capabilities presented in this section are, nonetheless, useful to industry: accurate evaluation of the equilibrium position of a vessel for a given trim angle and speed.

Chapter 4

Aerodynamic optimisation of an XCat-style catamaran

4.1 XCat racing boats as a platform for aerodynamic optimisation.

The UIM (Union Internationale Motonautique) XCat Class 1 World Championship is an international powerboat racing event hosted around the world. In this event, 12 catamarans powered by 2 400hp engines exceed speeds of 200kmph around a marked course. The category of boats used for this event are referred to as the XCat class, and as with most racing events optimisation of their design is sought within very strict regulations. An XCat racing boat is depicted for reference in Figure 4.1

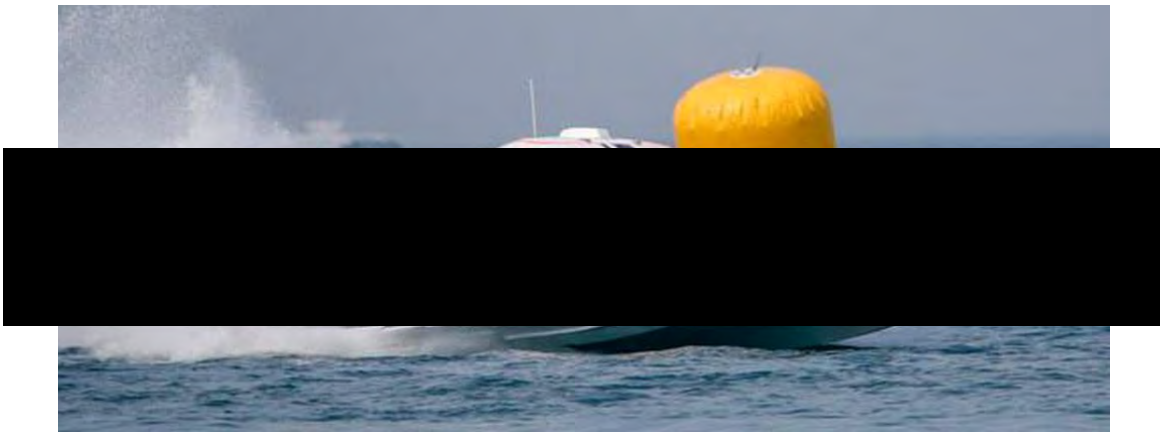
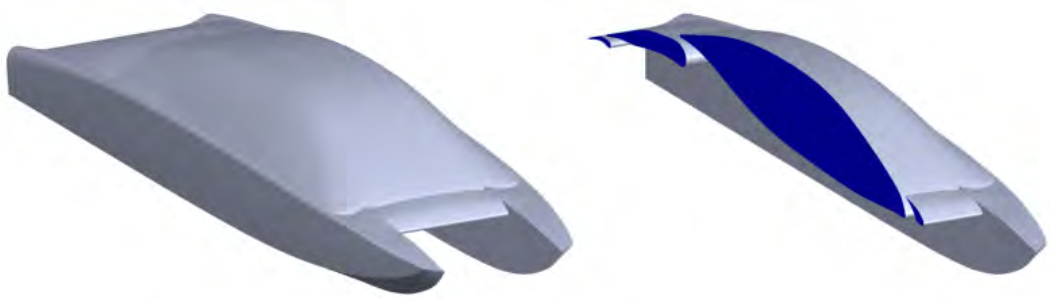
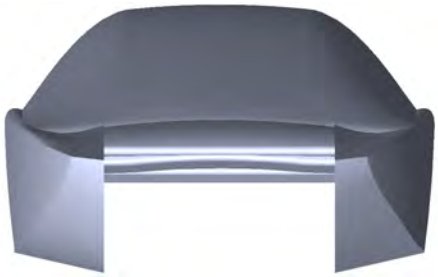


Figure 4.1: A typical XCat racing boat at speed [117].



(a) The modified XCat geometry of the XC10. A cross section (blue) is taken along the centreline to reveal the multi-wing configuration of the centre body.



(b) The frontal area with characteristic "tunnel" is shown from the perspective of oncoming airflow.

Figure 4.2: The area of aerodynamic interest in an XCat-style geometry is depicted. Note that this geometry is modified from an XCat class geometry by industry partners Norson Design and is referred to as the XC10. Modifications include a multi-wing aerofoil cabin and leading/trailing bodies.

Industry partners Norson Design were interested in developing an aerodynamically optimised boat with the XCat class as a platform. The primary rationale behind this venture was the concept that, due to the high speed planing mode of operation, the appreciable aerodynamic effects could be leveraged to minimise drag and increase performance. Specifically, optimisation of the high-pressure "tunnel" region inherent in the catamaran configuration was sought. To do this, Norson Design provided a baseline geometry of a catamaran based loosely on an XCat class vessel. This geometry, referred to as the XC10, is depicted in Figure 4.2.

Aerodynamic optimisation of the XC10 presented several potential avenues of performance improvement including pitch control and stability, but the main focus for initial optimisation was drag reduction. Industry partners wanted to explore the possibility of reducing the hydrodynamic component of drag, in turn reducing the total drag and increasing vessel speed for a given propulsive power. The strategy was to optimise aerodynamic lift generated

Figure 4.3: The A2V project prototype, a similar conceptual prototype vessel that leverages aerodynamic lift for drag reduction [118].

in the tunnel to lift the vessel out of the water, reducing the water-wetted area. A net reduction in drag could be achieved this way in spite of increase in aerodynamic drag so long as it is out-weighted by hydrodynamic drag improvements.

This concept of leveraging aerodynamic lift has precedent in existing projects, for example the Advanced Aerodynamic Vessels (A2V) project [118], which has produced a functioning prototype vessel to demonstrate the approach (see Figure 4.3).

The focus of this work was to develop tools to allow industry to further optimise a potential prototype vessel (the XC10) for aerodynamic properties. To do this, an existing 2D aerodynamic optimisation tool developed at Swansea University, AerOpt [119, 120], was adopted and applied to the design problem. This tool relies on a novel mesh movement scheme not applicable to the meshless LBM, and so 2D RANS simulations were used for the flow modelling. In initial optimisation, the goal was maximised lift from the centre-body of the XC10 geometry. The degree to which these 2D optimisations influenced the performance when extruded onto a 3D geometry were then assessed by

3D CFD experiments.

4.2 The 2D design baseline for optimisation

Although AerOpt is capable of optimising across a global design space with many degrees of freedom, starting from a strong baseline design and limiting the scope of the design space remains beneficial in saving computer run-time and avoiding impractical designs. The multi-disciplinary nature of vehicle design must be considered: the aerodynamic optimisation does not occur independently of other design factors, and consideration needs to be given to structural limitations, minimum internal volumes for crew/passengers/cargo, etc.

Motivation for 2D optimisation arose from a desire to reduce the problem to minimise design time. Not only would a design cycle for 3D optimisation involve 3D CFD experiments, which are time-expensive in themselves, it would mean a large increase in the degrees of freedom in the optimisation process. A 3D optimisation would require more control nodes to parameterise the additional geometric dimension and an additional degree of freedom per control node.

Flow under the centre-structure of the boat is "gated" by the twin hulls on either side, and so spill of flow outside this ducted volume is minimised and flow is expected to be highly axial. Due to the nature of this ducted space, optimisation of the 2D cross section was assumed to translate aerodynamic benefits to the 3D reality of the boat. Losses in this translation are later assessed with a 3D CFD study of the optimised geometry against the baseline.

A 2D cross-sectional baseline geometry was arrived at with input from the following people: George Robson, design engineer for Norson Design with a background in composite materials; Dr Kyle Forster, former Mercedes-AMG Petronas Formula 1 engineer and consultant aerodynamicist; and Dr Ben Evans, doctoral supervisor for this work and CFD aerodynamicist for the Bloodhound Land Speed Record vehicle. The cross section is depicted in Figure 4.4.

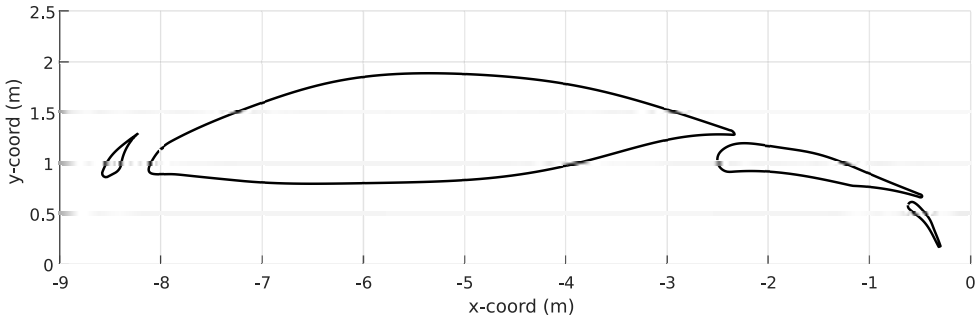


Figure 4.4: The 2D cross section of the XC10 centre body is depicted with the x-axis pointing in the direction of relative airflow. This section is along the centreline of the 3D boat. From left to right the bodies are referred to as the leading control surface, the cabin, the fixed secondary body, and the trailing control surface. The leading and trailing control surfaces are depicted in their high-lift position as found by analysis presented in Section 4.2.1.

The design is notably a multi-wing configuration of four bodies, and is broken down into – from left to right – the leading body, the cabin, the fixed secondary body, and the trailing body.

The design of the cabin is restricted by a minimum volume condition to fit internal components and crew. The cabin shape itself is that of an aerofoil to produce aerodynamic lift. The fixed secondary body behind the cabin is a consequence of splitting the central body. This ventilation in the fixed geometry was made to encourage high-energy flow from inside the tunnel to energise flow over the top and back of the geometry. This was done to mitigate possible separation of flow over the high-camber point of the aerofoil-shaped cabin, especially at high trim angles.

The leading and trailing control surfaces were featured due to a desire from industry to achieve active aerodynamic control of the vehicle in the future, be that in a pre-set trim position or an active response system. Optimisation of these bodies was beyond the scope of this project, and so optimal high-lift positions for each were found where they remained fixed for the later shape optimisation. In Figure 4.4 the control surfaces are in their high-lift configuration. Determination of these high-lift positions is explained in the following section.

4.2.1 Leading and trailing body position optimisation

Optimal high-lift positions of the leading and trailing surfaces were sought. These positions would remain fixed for the later shape optimisation of the fixed central bodies of the 2D baseline. Norson Design were consulted on the scope of the optimisation, and it was determined that three degrees of freedom would be practical: vertical displacement of the leading body, pitch of the leading body, and the pitch of the trailing body (ϕ_{1-3} respectively). See Figure 4.6. Pitch angles are taken about the leading edge of each aerofoil. The datum for displacement and angles are based on the neutral positions in the industry-provided CAD geometry. The limits for each parameter are given in Table 4.1.

Table 4.1: Design space limits relative to the industry-provided baseline.

design parameter	*lower bound*	*upper bound*
ϕ_1	-0.3m	0.05m
ϕ_2	-45°	45°
ϕ_3	-20°	45°

Latin Hypercube Sampling (LHS) was used to scatter 30 combinations of design parameters ϕ_{1-3} throughout the design space. The LHS method allows a near-random scattering through a n-dimensional design space in a manner that ensures a degree of variability not guaranteed by random scattering [121]. The combinations of design parameters are given in Table 4.3, and are visualised in the 3D design space in Figure 4.5.

Each combination of design parameters was then applied to a CAD geometry that was submitted for steady-state simulation using HyperWorks Virtual Wind Tunnel (VWT) [122]. HyperWorks VWT is a CFD software powered by AcuSolve, a general-purpose, finite element based RANS solver. VWT was used in large part due to the rapid setup for such a standard case, versus the longer lead-time that would be associated with writing the C++ code for a Palabos LBM simulation.

The VWT setup parameters are given in Table 4.2, and the results in Table 4.3. Slip wall boundary conditions were applied at the top and side

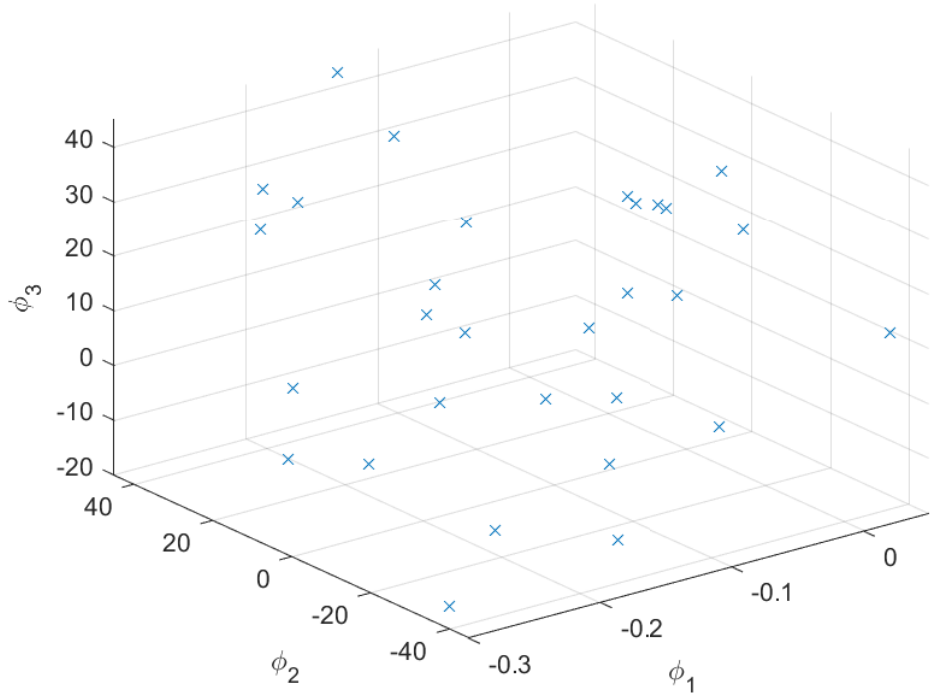


Figure 4.5: The Latin Hypercube Sampling (LHS) points for design parameters ϕ_{1-3} is visually depicted scattered throughout the design space. Exact values for each can be found in Table 4.3.

walls of the domain, no slip walls at the boat geometry and ground plane. The ground plane was also modelled as moving at the inlet speed, which was $100kts$ $(51.5m/s)$, the top design speed of the boat.

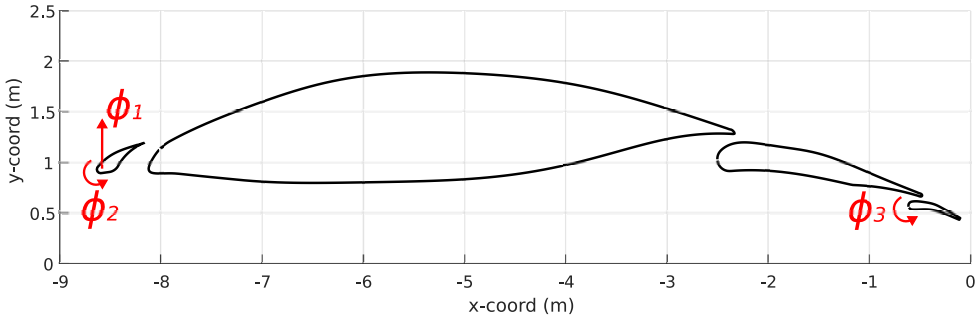


Figure 4.6: The physical context of the design parameters ϕ_{1-3} is shown. The datum for each body is taken from the leading edge of the aerofoil chordline.

Table 4.2: Simulation parameters for the VWT simulations informing leading and trailing body positions.

parameter	*units*	*value*
boat trim angle	$(^\circ)$	2
simulation type	–	steady-state, incompressible RANS
turbulence model	–	Spalart-Allmaras
domain size:		
x range	(m)	$[0.0, 25.0]$
y range	(m)	$[-4.0, 4.0]$
z range	(m)	$[0.0, 5.0]$
mesh properties:		
element count	–	approx. 6.4e6
boundary layer mesh	–	10 layers, layer 1 $y^+ = 1.0$
fluid properties:		
density	(kg/m^3)	1.225
dynamic viscosity	(kg/ms)	$1.8e-5$
inlet velocity	(m/s)	51.5
Reynolds number $(L = 10m)$	–	35e6
freestream Mach number	–	0.15

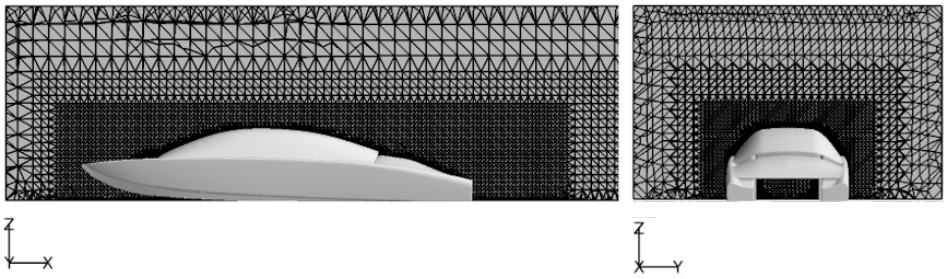
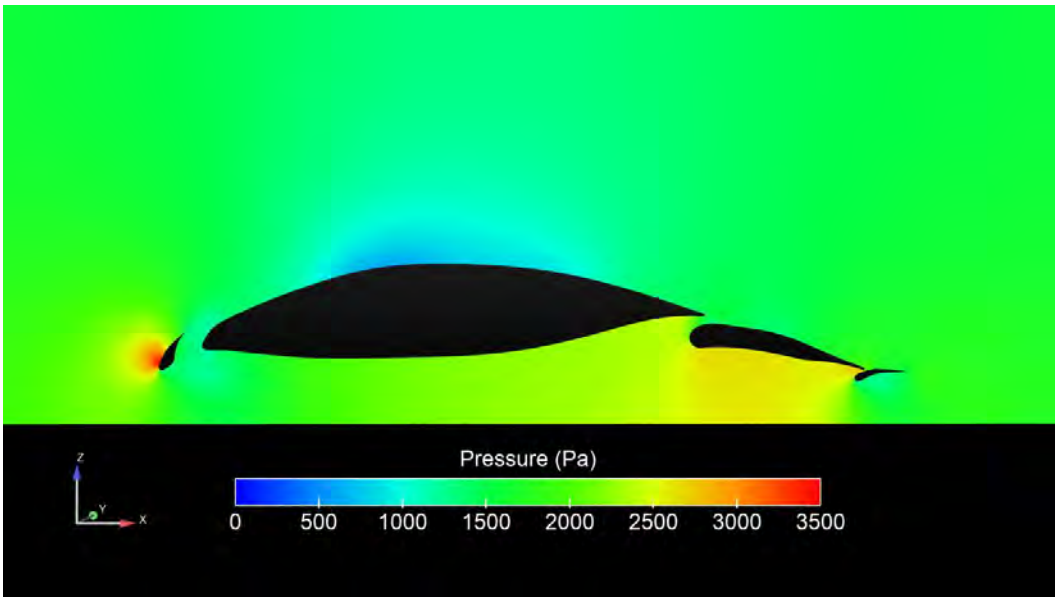


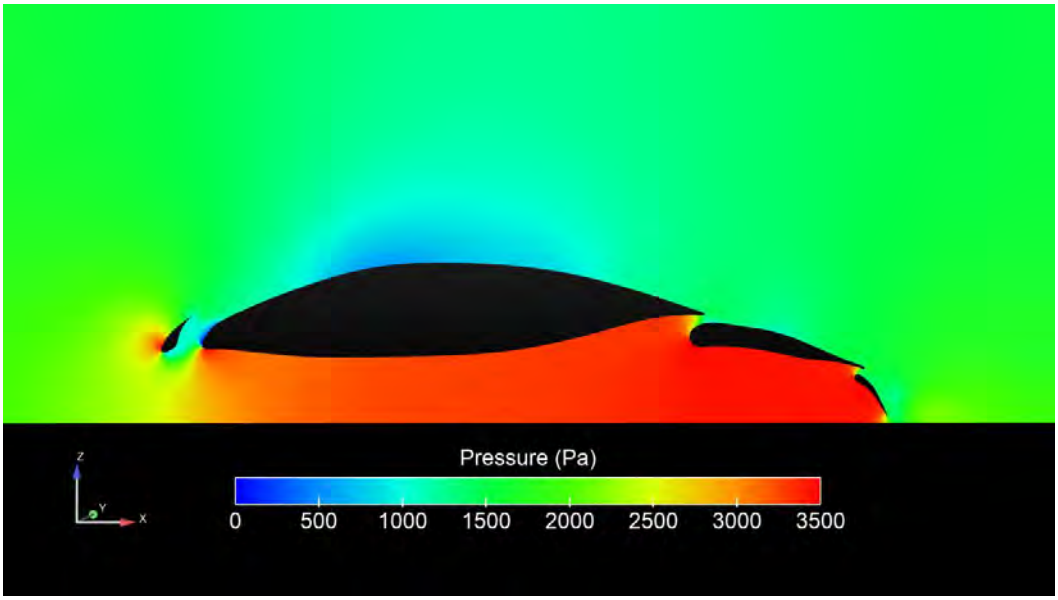
Figure 4.7: The mesh is seen from the side and rear of the boat to depict the mesh refinement zones for the VWT simulations.

Table 4.3: LHS scattered designs and their aerodynamic results as evaluated by Virtual Wind Tunnel. Geometries 12, 22, and 23 have no results due to their ϕ values producing self-intersecting geometries.
S = frontal area, C_D = coefficient of drag, C_L = coefficient of lift, L = lift force.

geometry	ϕ_1	ϕ_2	ϕ_3	S (m^2)	C_D	C_L	L (kN)
1	-0.01763	-3.75207	10.39346	4.357764	0.167	2.392	16.93345
2	-0.10722	-28.8746	41.43307	4.750478	0.035	2.889	22.29489
3	-0.19256	34.49056	26.36133	4.506648	0.109	2.634	19.28368
4	-0.08594	37.24498	-2.05355	4.184257	0.147	1.897	12.89455
5	0.031956	15.68678	16.92107	4.382652	0.19	2.439	17.36478
6	-0.16602	-37.9217	-13.2056	4.09746	0.062	2.043	13.5989
7	-0.19636	-17.0178	-16.4405	4.09746	0.137	1.759	11.7085
8	-0.25921	20.94914	37.73823	4.630644	0.093	2.55	19.18236
9	-0.0702	12.30772	-10.3971	4.10986	0.178	1.952	13.03249
10	-0.23521	-14.6245	30.15641	4.556246	0.108	2.659	19.68095
11	0.004264	-6.92634	-13.8667	4.10986	0.16	2.048	13.67343
12	-0.10104	22.69079	0.289687	–	–	–	–
13	-0.06192	-2.7488	-5.68859	4.134659	0.169	1.982	13.31263
14	-0.12896	31.22401	35.41774	4.612488	0.101	2.734	20.48587
15	-0.27433	8.096882	6.374643	4.233856	0.154	1.78	12.24268
16	0.020737	-44.8187	14.82905	4.357746	0.055	2.616	18.51911
17	-0.11611	-23.7185	23.95427	4.643044	0.064	2.892	21.81331
18	-0.18048	-40.6382	2.483685	4.38956	0.037	2.546	18.15515
19	-0.03103	-19.3529	39.37958	4.729841	0.076	2.85	21.89838
20	-0.25093	24.1543	28.45184	4.593445	0.098	2.61	19.47599
21	-0.21528	-9.37918	5.597893	4.295854	0.15	1.847	12.88953
22	-0.22857	4.304848	-9.12029	–	–	–	–
23	0.04309	29.46097	13.68102	–	–	–	–
24	-0.29454	2.718527	-3.59875	4.134659	0.142	1.346	9.040768
25	-0.03374	44.66157	8.817222	4.333053	0.124	2.181	15.35219
26	-0.15403	-26.7741	21.09875	4.506648	0.082	2.669	19.53992
27	-0.14071	41.74093	44.24382	4.690517	0.08	2.799	21.32771
28	-0.28449	-34.8832	-18.4881	4.09746	0.077	1.21	8.054167
29	-0.05005	-31.3081	33.70299	4.593357	0.048	2.898	21.62465
30	-0.00382	11.8671	21.33995	4.439424	0.181	2.55	18.39024



(a) The lowest lift configuration (geometry 28).



(b) The highest lift configuration (geometry 2).

Figure 4.8: The lowest and highest lift leading and trailing body configurations are shown as simulated in FLITE2D, the solver that was used for the optimisation described in Section 4.1. The 276.8% increase in lift can be seen to arise mainly for the increase in pressure within the tunnel. This is caused by the "open" configuration at the leading body, and "closed" position of the trailing body. Note the pressure plotted is total pressure minus atmospheric pressure.

The highest lift geometry (geometry 2) can be seen to be producing 277% more lift than the lowest lift geometry (geometry 28). The two are compared in Figure 4.8, where it appears that the bulk of the increase in lift comes from increase in the high pressure below the boat rather than the lowering

of pressure across the top. This seems to be caused by the "opening" of the front of the tunnel and "closing" of the rear. This gain in lift is not without cost, as geometry 2 also sees a 220% increase in drag over geometry 28.

The high-lift leading and trailing bodies configuration of geometry 2 was carried forward as the baseline positioning for the AerOpt shape optimisation in Section 4.4.

4.3 AerOpt

AerOpt is an aerodynamic optimisation software developed by Dr David Naumann at Swansea University [119, 120]. AerOpt is an automated, population-based evolutionary optimisation algorithm that parameterises a search domain and design geometry based on user-defined control nodes, and then uses a series of CFD simulations and mesh movement schemes to optimise the geometry for a given objective function. The constituent methods used by AerOpt are described in the following subsections, followed by description of how they are combined to deliver optimised 2D aerodynamic geometries.

4.3.1 Geometry parameterisation and mesh movement

The mesh movement of AerOpt works by directly manipulating the mesh used for the later CFD experiments that assess the aerodynamic properties of the geometries, and so parameterisation is also applied directly to the computational mesh. To do this, mesh nodes are broken down into three types: boundary nodes, which occur along the perimeter of the geometry being studied; domain nodes, which are the remaining nodes in the domain volume that are not located at the boundary; and finally control nodes, which are select boundary nodes that define the later mesh movement. The control nodes are selected by the user, as are the limits of motion for each control node. This scheme is illustrated in Figure 4.9.

The choice of how many control nodes are used, the positions they are placed, and how limited their motion may be, are crucial to the degree of flexibility the geometry has to be optimised. The total degrees of freedom of

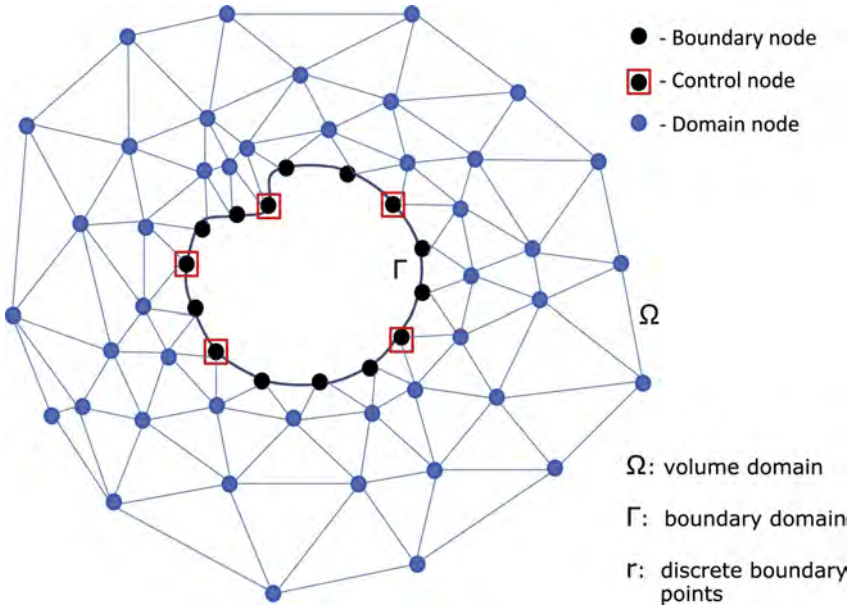


Figure 4.9: The AerOpt mesh parameterisation scheme is depicted in a general case [120]. The mesh is broken down into nodes at the geometry surface, with all other nodes being considered domain nodes. The control nodes are a special user-selected type of nodes that are free to move with their user-defined constraints, indicated by the red bounding boxes.

this control node system defines the dimensionality of the design space, d, and can be expressed as in Equation 4.1.

$$d = \sum_{k=1}^{N_{CN}} f_{CN} \tag{4.1}$$

where N_{CN} is the number of control nodes and f_{CN} is the number of degrees of freedom per control node.

The Delauney Condition requires that for a grid of triangles the circumcircle of each triangle does not contain a point of another triangle [123]. This condition is required for mesh validity for the later CFD simulations that will take place on this mesh. Thus to move the control nodes meaningful distances the boundary and volume nodes must also deform in a way that maintains mesh validity. This is performed as described by Naumann et al. [120].

4.3.2 Optimisation – modified cuckoo search algorithm

The control nodes described serve a dual purpose in parameterising the geometry and as the design parameters for the optimisation process itself. As

the dimensionality of the optimisation problem, defined previously in Equation 4.1, could conceivably grow large for complex aerodynamic bodies, the optimisation method used needed to be suitable for large design spaces and not prone to localised solutions. This motivated the use of an evolutionary algorithm, the Modified Cuckoo Search (MCS) [124].

The objective function informs the optimiser what is desired from the optimisation process. This may be a single factor or combination of weighted factors that are to be maximised or minimised. The fitness of a solution is a single quantity that describes the degree to which a given solution satisfies the objective function. For this work the objective function is a simple maximisation of aerodynamic lift, and so the fitness of each solution is simply the 2D lift coefficient, C_l.

The MCS is a population based algorithm that mimics natural selection in cuckoo bird reproduction. Iterations of the optimisation process are referred to as generations. The first generation is made by sampling the entire design space, producing combinations of design parameters referred to as eggs. The fitness of these eggs are then assessed based on the criteria of the designer: in the case for this work, fitness is aerodynamic lift as assessed by 2D CFD simulations. A fraction of these eggs are then discarded based on their fitness, and new eggs generated by a Lévy flight [125] through the design space replace the discarded eggs. Among the retained eggs, a selection of the best are paired and cross-bred to create a new egg which is retained for the next generation if its fitness exceeds that of the parent eggs. The ratio of best to worst eggs for this work is set at 1:3 based on empirical evidence that this is optimal [71].

4.3.3 CFD simulations – FLITE2D

The assessment of fitness of each geometry generated in the optimisation stage was conducted by a 2D CFD simulations. The solver used to perform these simulations was FLITE, an in-house solver developed at Swansea University. FLITE is an edge-based, vertex-centred finite volume discretised solver of the compressible Reynolds-Averaged Navier-Stokes (RANS) equations [126]. In particular, the 2-dimensional version of FLITE, FLITE2D,

was used. FLITE2D features its own native mesh generator, which is used to generate a 2D unstructured triangular mesh using a Delauney approach [123], which was necessitated by the current implementation of the AerOpt code. A Spalart-Allmaras turbulence model was used based on its applicability to external aerodynamic flows [127]. The primary driver behind the use of FLITE2D for CFD is that FLITE2D and its native mesher were the systems currently programmed into the AerOpt programme. Although, theoretically, AerOpt could be used with any "black box" CFD mesher and solver taking inputs and returning solutions, it was determined that the work required to do this was unnecessary, as FLITE2D was a suitable solver for the case presented.

4.3.4 AerOpt algorithm

The AerOpt process is outlined in Figure 4.10, where the context for each sub-process described is given. This constitutes the design loop followed for each of the design iterations discussed in the upcoming sections.

4.4 XC10 AerOpt optimisations and results

The optimisation of the XC10 geometry took place over several AerOpt optimisation cycles. These are presented in the following sections.

4.4.1 XC10 baseline optimisation

The first AerOpt optimisation used the baseline geometry with the leading and trailing bodies set to the positions determined in Section 4.2.1. This optimisation sought to maximise aerodynamic lift.

The first step in the AerOpt process involves determining the scope of the shape optimisation via the choice of the position, limits, and number of the controls nodes to be used. Control nodes given limits of zero act as anchor points in the mesh movement process, and their placement is as important as control nodes that are free to move. Moveable (free) control nodes should be placed appropriately where deformation is acceptable, and have their limits respect design constraints such as minimum internal volumes and structural

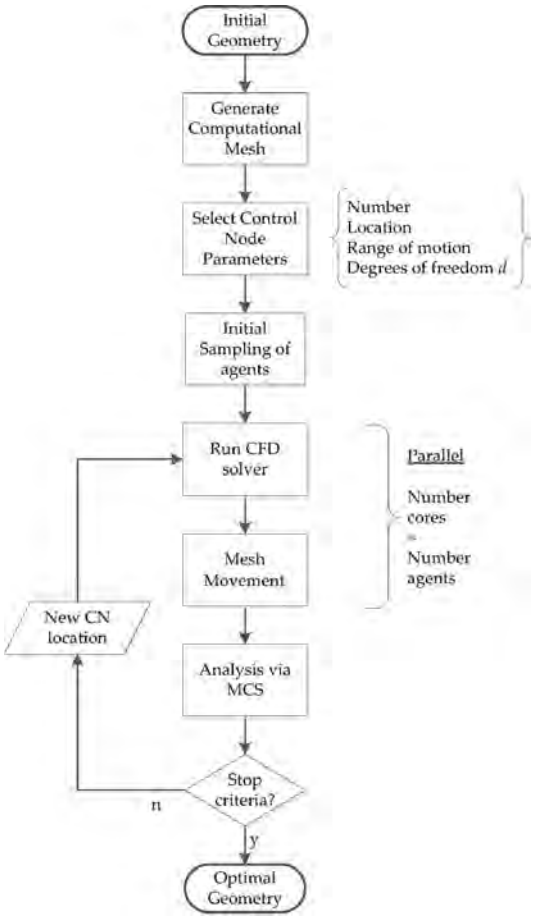


Figure 4.10: AerOpt's procedure is depicted as a flowchart for n number of generations (iterations) [119].

limitations. The position and limits of the control nodes used for optimisation of the baseline geometry are depicted in Figure 4.11.

The parameters for both AerOpt and the FLITE2D simulations are given in Table 4.4, and the control node positions are shown in Figure 4.11. The pre- and post-optimisation meshes and geometry outlines are shown in Figures 4.13 and 4.14 respectively. The process ran on 21 cores, one for each of the 20 nests (thus 20 FLITE2D CFD simulations per generation) plus one for the remaining processes. The requested 500 generations were achieved in 23 hours 22 minutes of wall clock time on the HPCWales Sunbird system.

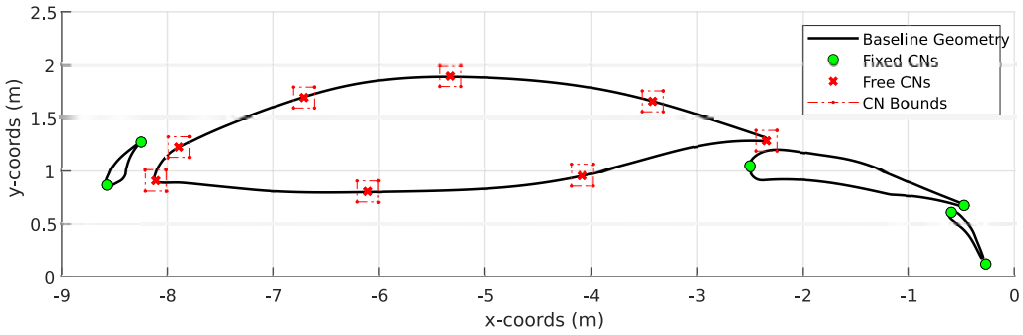


Figure 4.11: The control node placement and bounds are depicted for the optimisation of the baseline geometry.

Table 4.4: AerOpt and FLITE2D parameters for the optimisation of the baseline XC10 geometry with leading and trailing bodies in high-lift configuration.

parameter	*units*	*value*
initial boat geometry	–	XC10 baseline
boat trim angle	$(^\circ)$	2
AerOpt parameters:		
objective function	–	C_l
number of control nodes	–	14
ratio of best:worst nests	–	1:3
number of nests	–	20
number of Lévy steps	–	100
number of generations	–	500
FLITE2D parameters:		
compute cores per simulation	–	1
domain x range	(m)	$[-10.0, 22.0]$
domain y range	(m)	$[0.0, 10.0]$
turbulence model	–	Spalart-Allmaras
Reynolds number, $(L = 10m)$	–	3,500,000
freestream Mach number	–	0.15
ambient temperature	(K)	293
ambient pressure	(Pa)	101,325
specific gas constant	(J/kgK)	273
ratio of specific heats	–	1.4
density	(kg/m^3)	1.225
inlet velocity	(m/s)	51.5
convergence criteria	–	e-2

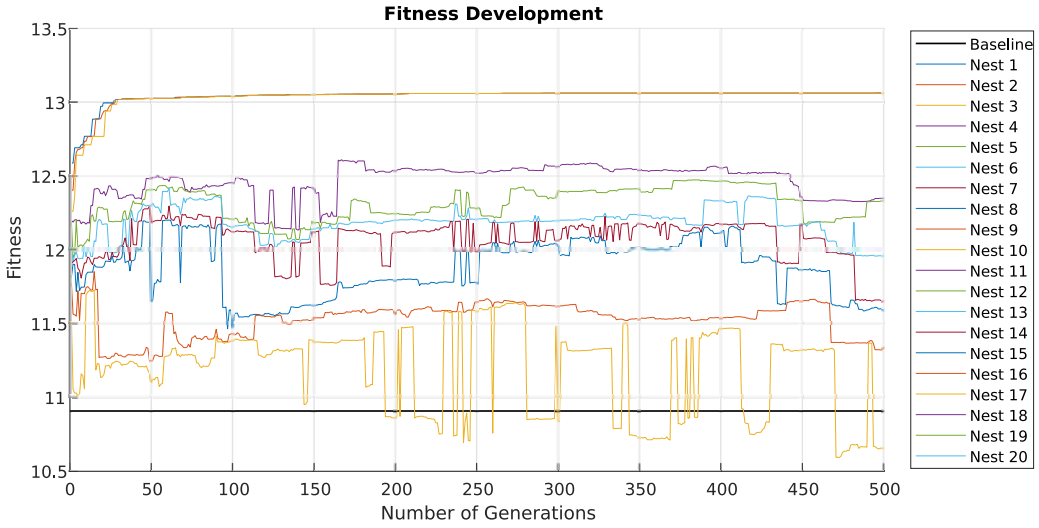
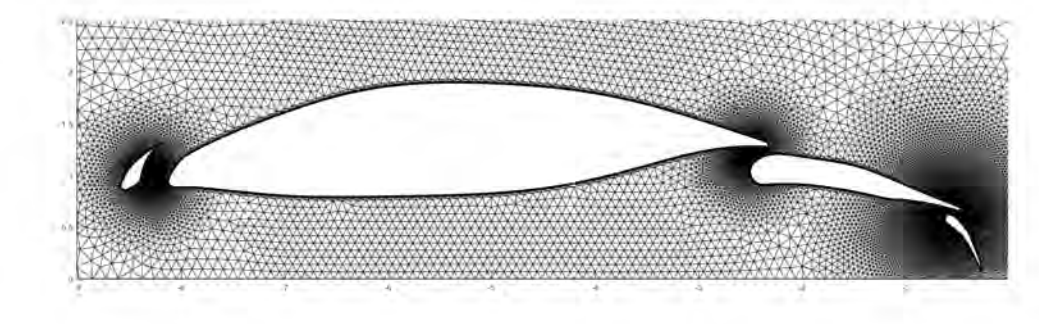
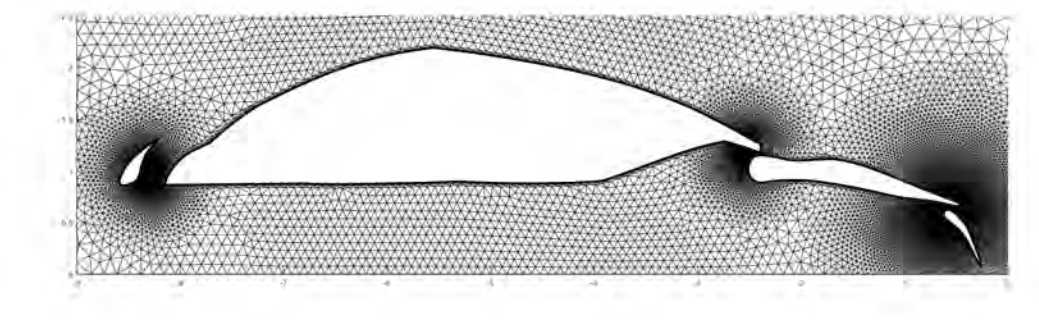


Figure 4.12: The evolution of the nests across generations for the baseline optimisation. In this case fitness is the lift coefficient.



(a) The pre-optimisation mesh.



(b) The post-optimisation mesh.

Figure 4.13: The pre- and post-optimised meshes for the baseline optimisation are shown. The mesh refinement around points of expected flow complexity and the consequences of the mesh movement can be seen.

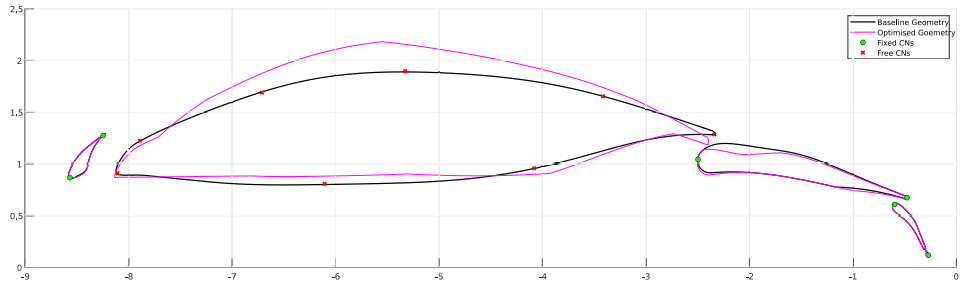


Figure 4.14: The pre- and post-optimisation geometry outlines are compared directly.

The best nest, seen from Figure 4.12, after 500 generations produced a geometry with a 19.76% increase in lift, increasing from $C_l = 10.906$ to $C_l = 13.062$. The geometry changes that achieved this, as can be seen in Figure 4.14, are an increased camber in the main cabin, and flattening of the underside of the cabin that exaggerates the expansion in the tunnel after $x = -4m$. The mesh movement influence of the control node at the rear of the main cabin can be seen to have "jumped" across the gap between bodies and deformed the fixed secondary body, causing a tightening of the ventilation gap between the two. In doing this, a Venturi effect causes faster flow through the constrained gap, feeding energised air to the top-side boundary layer and delaying separation.

These geometry changes make sense for a fitness function of maximised aerodynamic lift. The cabin, which is essentially an aerofoil, saw an increase in camber, a trait that is typically associated with increased lift. The tightening of the ventilation gap is also an exaggeration of a geometric property intended to increase lift. With this in mind, this post-optimisation geometry was carried forward as the baseline for the next AerOpt optimisation.

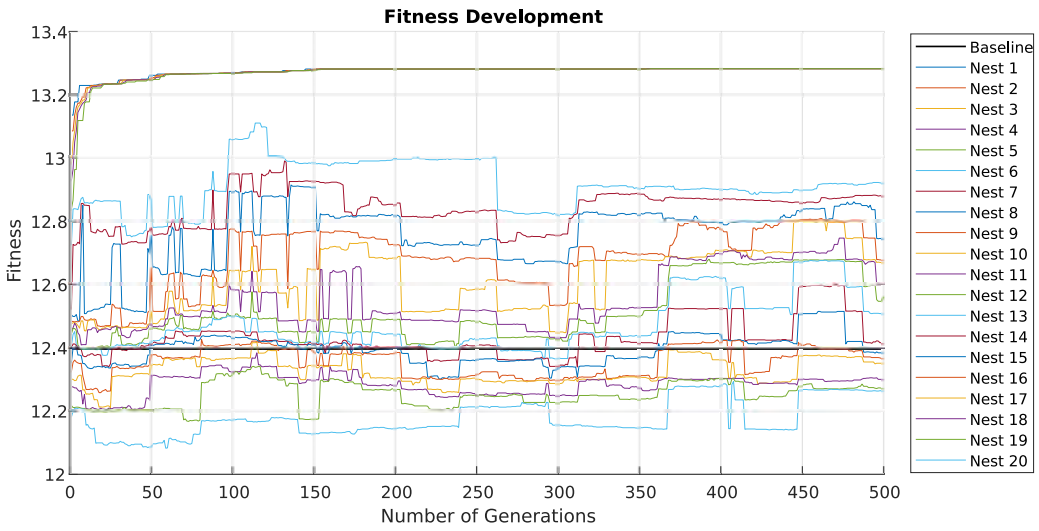


Figure 4.15: The evolution of the nests across generations for the windscreen optimisation. In this case fitness is the lift coefficient.

4.4.2 XC10 windscreen optimisation

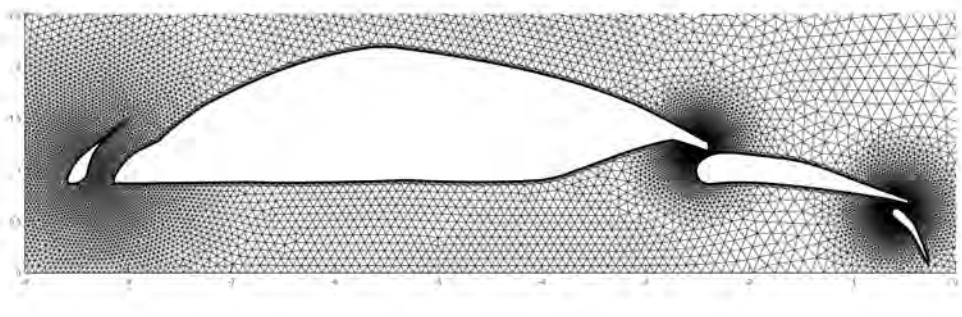
A more focused study was carried out to explore gains available in the flow across the top of the boat. This allowed for concentrated control node placement around a localised area, in this case the windscreen. The optimised baseline geometry from the previous section was used as a starting point with some minor changes: sharp edges, for example at the peak of the cabin, have been smoothed, and the fixed secondary body has been restored to it's original form due to structural constraints. This sees a small drop in the starting C_l to 12.397.

The parameters for both AerOpt and the FLITE2D simulations are given in Table 4.5. The pre- and post-optimisation meshes and geometry outlines with control node positions are shown in Figures 4.16 and 4.17 respectively.

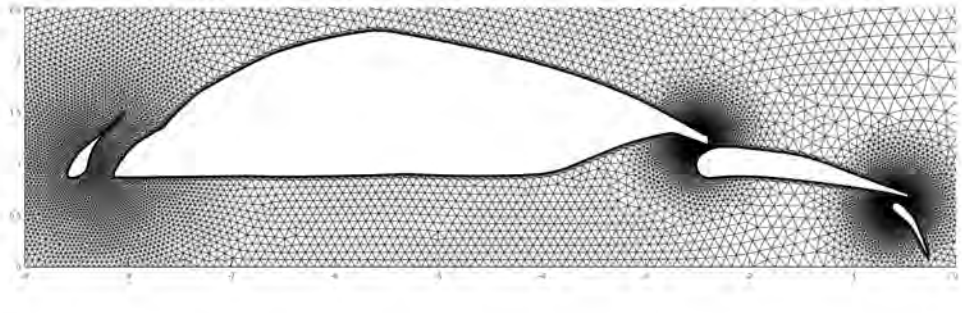
The process ran on 21 cores, one for each of the 20 nests (thus 20 FLITE2D CFD simulations per generation) plus one for the remaining processes. The requested 500 generations were achieved in 26 hours 01 minutes of wall clock time on the HPCWales Sunbird system.

Table 4.5: AerOpt and FLITE2D parameters for the optimisation of the windscreen of the post-optimisation XC10 geometry with leading and trailing bodies in high-lift configuration.

parameter	*units*	*value*
initial boat geometry	–	XC10 baseline, post-optimisation
boat trim angle	$(^\circ)$	2
AerOpt parameters:		
objective function	–	maximise lift
number of control nodes	–	13
ratio of best:worst nests	–	1:3
number of nests	–	20
number of Lévy steps	–	100
number of generations	–	500
FLITE2D parameters:		
compute cores per simulation	–	1
domain x range	(m)	$[-10.0, 22.0]$
domain y range	(m)	$[0.0, 10.0]$
turbulence model	–	Spalart-Allmaras
Reynolds number, ($L = 10m$)	–	3,500,000
freestream Mach number	–	0.15
ambient temperature	(K)	293
ambient pressure	(Pa)	101,325
specific gas constant	(J/kgK)	273
ratio of specific heats	–	1.4
density	(kg/m^3)	1.225
inlet velocity	(m/s)	51.5
convergence criteria	–	e-2



(a) The pre-optimisation mesh.



(b) The post-optimisation mesh.

Figure 4.16: The pre- and post-optimised meshes for the windscreen optimisation are shown. The mesh refinement around points of expected flow complexity and the consequences of the mesh movement can be seen.

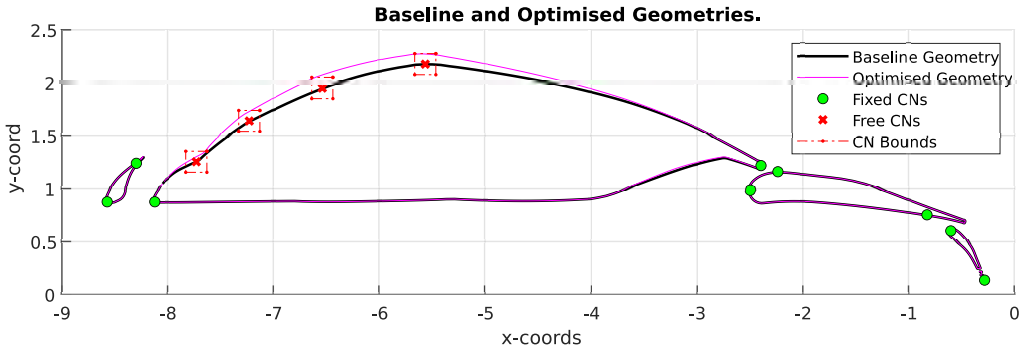


Figure 4.17: The pre- and post-optimisation geometry outlines are compared directly.

The trends seen in the previous optimisation (Section 4.4.1) are seen, restrained to the local windscreen area: the optimiser is driving the aerofoil of the cabin to a higher camber to increase lift. From Figure 4.17 it can be seen that mesh movement has been restricted to the windscreen area.

From Figure 4.15 it can be seen that the geometry increase C_l from 12.397 to 13.282, an increase of 7.14%. An optimal geometry was achieved early in the 500 generations.

4.4.3 XC10 vent optimisation

Similar to the windscreen optimisation, a focused optimisation study was made in the tunnel around the vent between the cabin and fixed secondary body. This was especially necessary as the deformation of the fixed secondary body in the initial baseline optimisation had to be reverted for structural reasons. The optimised baseline geometry from the previous section was used as a starting point with some minor changes: sharp edges, for example at the peak of the cabin, have been smoothed, and the fixed secondary body has been restored to it's original form due to structural constraints. This sees a small drop in the starting C_l to 12.397.

The parameters for both AerOpt and the FLITE2D simulations are given in Table 4.6. The pre- and post-optimisation meshes and geometry outlines with control node positions are shown in Figures 4.19 and 4.20 respectively.

The process ran on 21 cores, one for each of the 20 nests (thus 20 FLITE2D CFD simulations per generation) plus one for the remaining processes. The requested 500 generations were achieved in 26 hours 07 minutes of wall clock time on the HPCWales Sunbird system.

From Figure 4.18 it can be seen that an initial jump in fitness was later exceeded by a secondary jump resulting from cross-breeding with other designs. Fitness increased from C_l of 12.397 to 13.849, a 11.71% increase. Although the control node focus was on the tunnel part of the geometry, some deformation is seen on the top of the cabin (see Figure 4.20).

Table 4.6: AerOpt and FLITE2D parameters for the optimisation of the vent of the post-optimisation XC10 geometry with leading and trailing bodies in high-lift configuration.

parameter	*units*	*value*
initial boat geometry	–	XC10 baseline, post-optimisation
boat trim angle	$(^\circ)$	2
AerOpt parameters:		
objective function	–	maximise lift
number of control nodes	–	13
ratio of best:worst nests	–	1:3
number of nests	–	20
number of Lévy steps	–	100
number of generations	–	500
FLITE2D parameters:		
compute cores per simulation	–	1
domain x range	(m)	$[-10.0, 22.0]$
domain y range	(m)	$[0.0, 10.0]$
turbulence model	–	Spalart-Allmaras
Reynolds number, $(L = 10m)$	–	3,500,000
freestream Mach number	–	0.15
ambient temperature	(K)	293
ambient pressure	(Pa)	101,325
specific gas constant	(J/kgK)	273
ratio of specific heats	–	1.4
density	(kg/m^3)	1.225
inlet velocity	(m/s)	51.5
convergence criteria	–	e-2

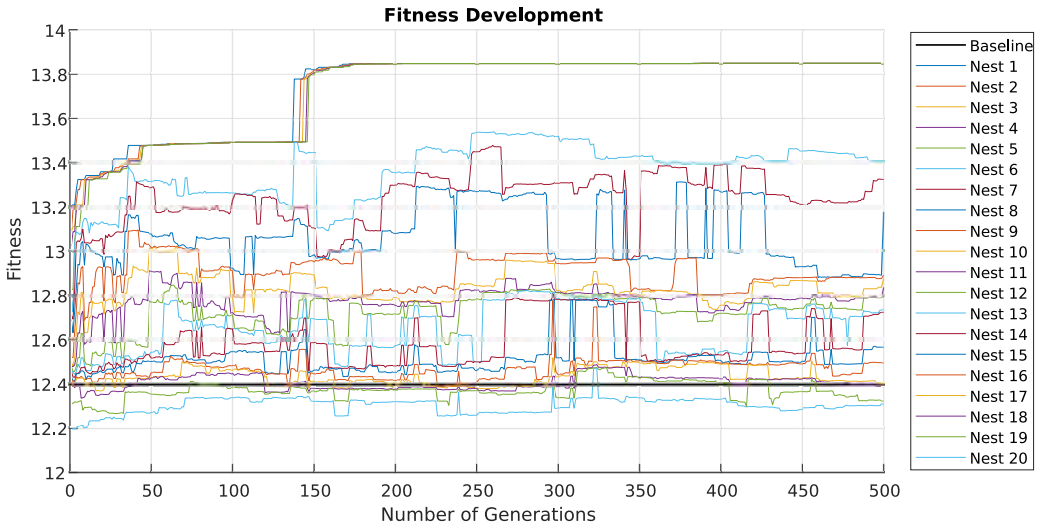
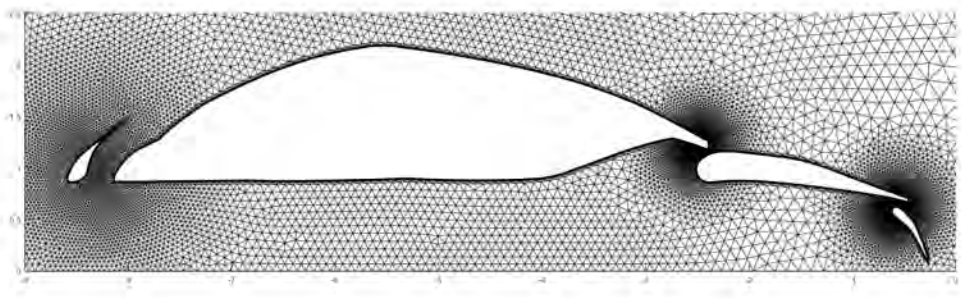
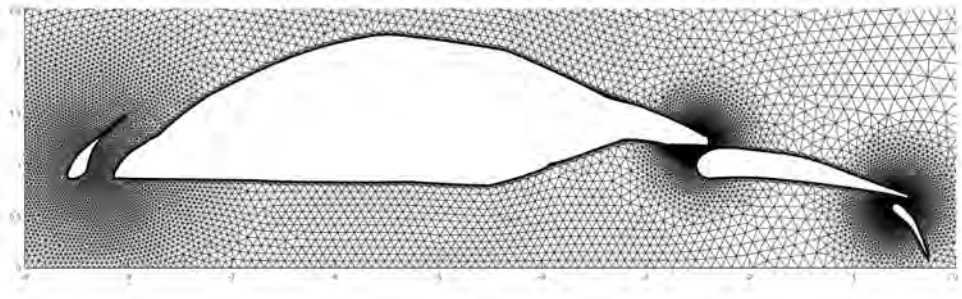


Figure 4.18: The evolution of the nests across generations for the vent optimisation. In this case fitness is the lift coefficient.



(a) The pre-optimisation mesh.



(b) The post-optimisation mesh.

Figure 4.19: The pre- and post-optimised meshes for the vent optimisation are shown. The mesh refinement around points of expected flow complexity and the consequences of the mesh movement can be seen.

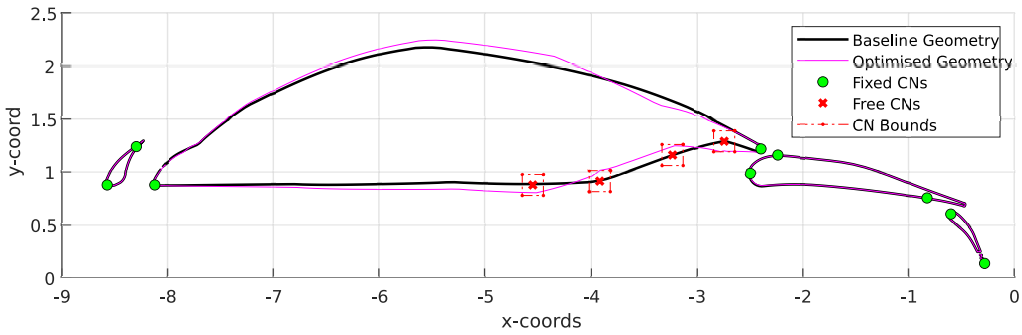


Figure 4.20: The pre- and post-optimisation geometry outlines are compared directly.

4.4.4 XC10 windscreen/vent combination

The optimisations of the windscreen and vent both yielded lift increases with geometry deformation in localised areas. To best take advantage of these, a combined geometry was created, depicted in Figure 4.21. The vent-optimised geometry takes precedence over the windscreen on the under-side of the geometry, and vice versa for the top. This is achieved without much conflict as the local optimisations did not influence the geometry greatly outside their focus.

The combined geometry was submitted for a FLITE2D simulation under conditions matching those used in Sections 4.4.1-4.4.3, and returned a further increased C_l value of 14.280, exceeding both the vent and windscreen optimisations individually. The increase in lift throughout the optimisation process is shown in Figure 4.22.

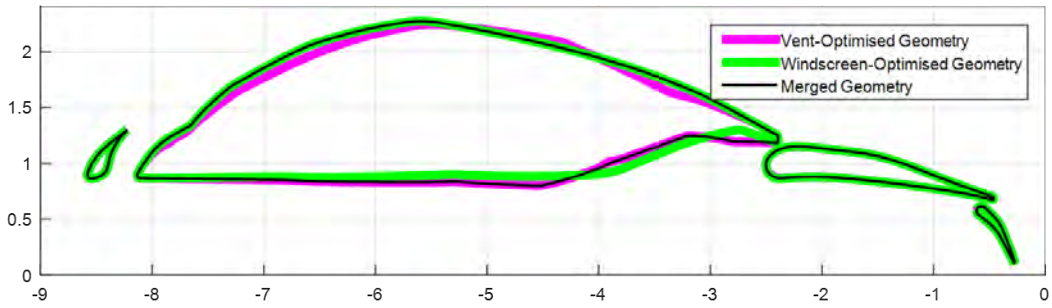


Figure 4.21: The windscreen and vent optimisation results are shown with their resulting combined geometry.

4.4.5 XC10 2D AerOpt optimisation results

The AerOpt optimisation process applied to the 2D XC10 centreline geometry successfully increased the lift coefficient, C_l from 10.906 to 14.280, a 30.9% increase. The evolution of the C_l for each geometry change are graphed in Figure 4.22.

As the fitness function throughout the AerOpt process has been an unweighted focus on maximising aerodynamic lift, these geometry changes have – not unexpectedly – incurred an increase in drag coefficient, C_d, from the baseline value of 0.87 to 1.05, a 21.0% increase. Though this was proportionally less than the 30.9% lift increase, it was still an undesirable performance penalty. The working theory remained, however, that aerodynamic drag increases arising from an increase in aerodynamic lift would be offset by the larger losses in hydrodynamic drag as a consequence of a reduced water-wetted area.

The optimisation presented to this point has been limited to the 2D cross-section, under the assumption that although some losses may occur in translating this to a real 3D geometry, this was minimised by the gated, duct-like nature of the catamaran tunnel, and that losses would be outweighed by the overall improvement. This assumption was analysed in the following section.

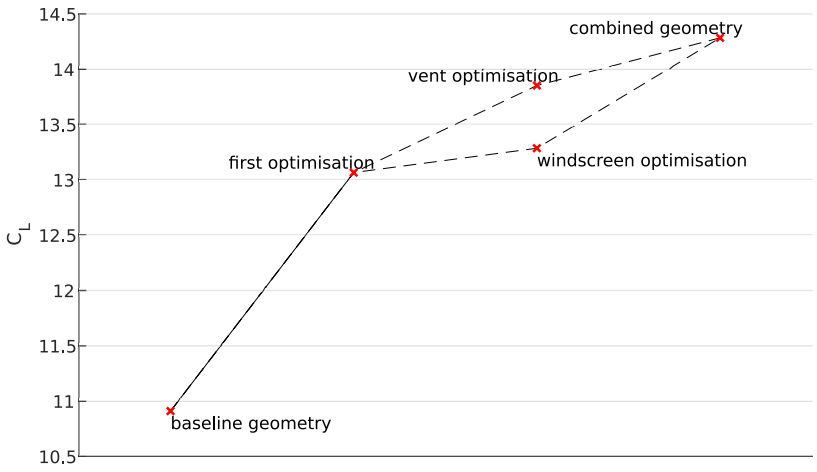


Figure 4.22: The incremental increase in C_l is shown through the 2D AerOpt optimisation process.

4.5 XC10: 3D CFD assessment of 2D optimisation

To assess the degree to which the 2D gains in lift translate back to the real 3D boat geometry, two 3D CFD simulations were carried out using Altair Virtual Wind Tunnel (VWT). Two geometries were simulated, an XC10 geometry with a centrebody made of the extruded baseline geometry presented in Section 4.4.1, and another with the centrebody made up of the extruded optimised geometry.

The geometries can be seen in Figure 4.23. The choice to make the centre body a simplified extrusion of the 2D shapes was made to eliminate the possibility that differences in the tapering of the prisms to more realistic geometries would be the source of the performance differences. Realistically, the 2D optimised shape would inform a 3D geometry that would then be tapered towards the outside of the boat.

The results are presented in Table 4.7, and flow solution results can be seen in Figures 4.24 and 4.25. The 2D optimisation has resulted in a 15.8% increase in aerodynamic lift, though this comes at a cost of a 30.7% increase in aerodynamic drag. This is not in line with the 2D drag cost, which was smaller than the change in lift for the 2D optimisation (30.9% increase in lift for 21.0% increase in drag). Some of this is due to the blocky prismatic shape of the centrebody, which causes a lot of separation on its side due to its sharp edges (see Figure 4.24). In practice, the optimised 2D cross section should inform a 3D geometry rather than be simply extruded, reducing drag related to separation at this face.

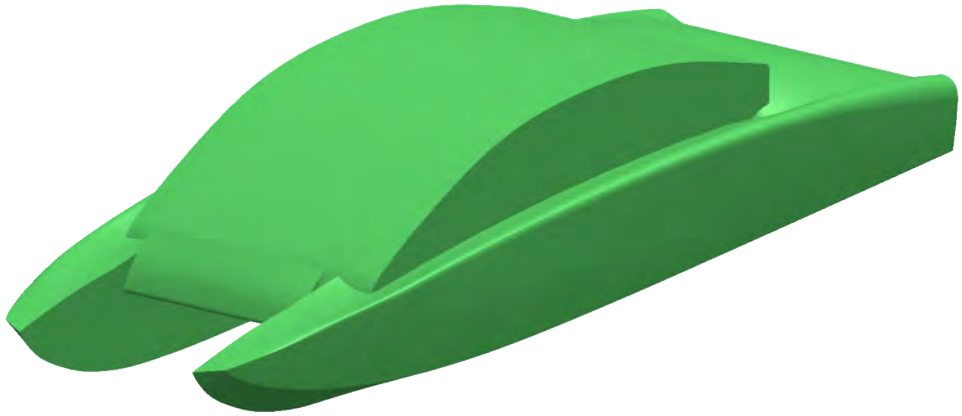
This study was predicated on the idea that aerodynamic lift could offset hydrodynamic drag. For this reason, aerodynamic drag incurred by an increase in the aerodynamic lift of the boat should be considered in absolute terms, as it is expected that the resulting reduction in hydrodynamic drag will greatly exceed it. At $51.5m/s$, the estimated top speed of the XC10, this yields an absolute lift increase of 3,700N at the cost of 1,050N in drag.

Table 4.7: Results of the VWT testing of the 3D extruded geometries. Note that a unitary reference area of $S = 1m$ is used in calculation of the aerodynamic coefficients.

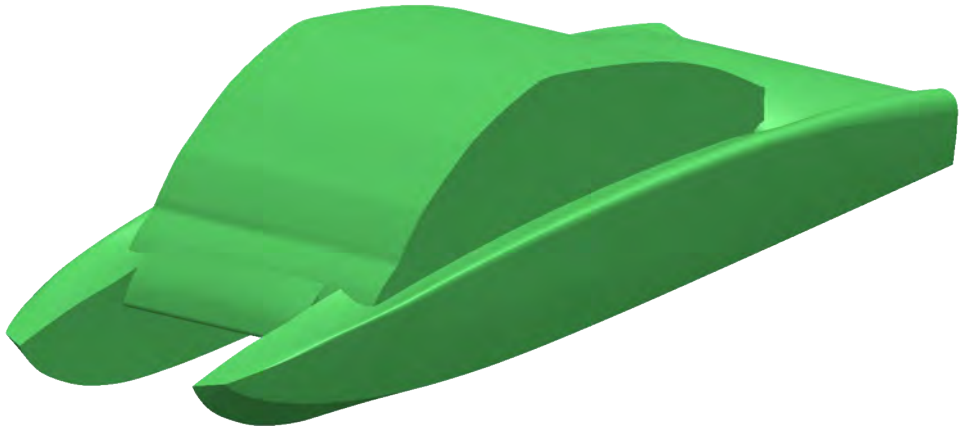
	baseline	*optimised*	*% change*
2-dimensional			
C_l	10.91	14.28	+30.9%
C_d	0.87	1.05	+21.0%
3-dimensional			
C_L	14.07	16.30	+15.8%
C_D	2.08	2.72	+30.7%

Table 4.8: Simulation parameters for the VWT simulations assessing changes between 2D and 3D modelling.

parameter	*units*	*value*
boat trim angle	$(^\circ)$	2
simulation type	–	steady-state, incompressible RANS
turbulence model	–	Spalart-Allmaras
domain size:		
x range	(m)	$[0.0, 25.0]$
y range	(m)	$[-4.0, 4.0]$
z range	(m)	$[0.0, 5.0]$
mesh properties:		
element count	–	approx. $10e6$
boundary layer mesh	–	20 layers, layer 1 $y^+ = 1.0$
fluid properties:		
density	(kg/m^3)	1.225
dynamic viscosity	(kg/ms)	$1.8e-5$
inlet velocity	(m/s)	51.5
Reynolds number ($L = 10m$)	–	$35e6$
freestream Mach number	–	0.15

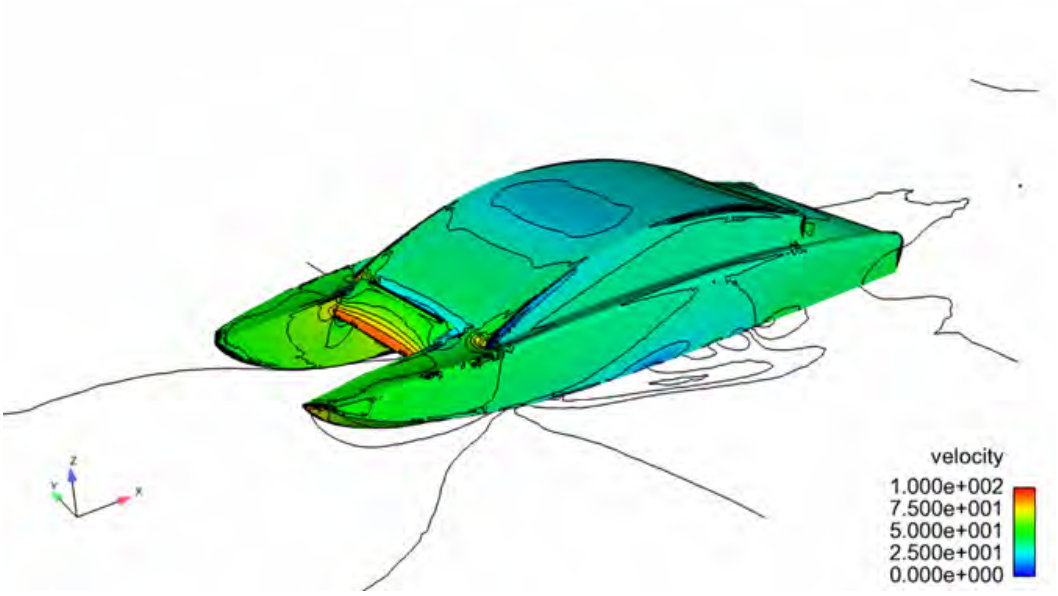


(a) The 3D geometry based on an extrusion of the 2D baseline geometry.

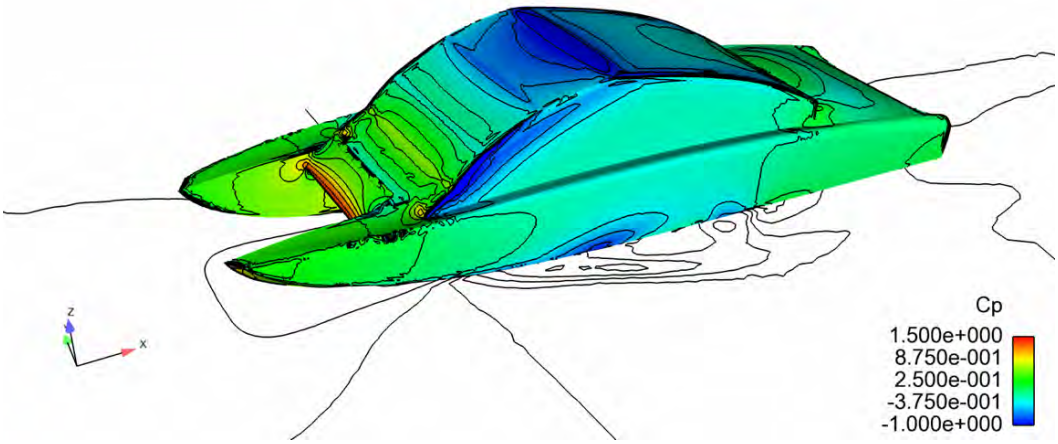


(b) The 3D geometry based on an extrusion of the 2D optimised geometry.

Figure 4.23: The two 3D geometries are compared. Each is based on an extrusion of the 2D baseline and optimised geometry respectively. The basic extrusion is used rather than a realistic form tapered towards the outside of the boat in order to best assess the 3D performance of the 2D optimisation.

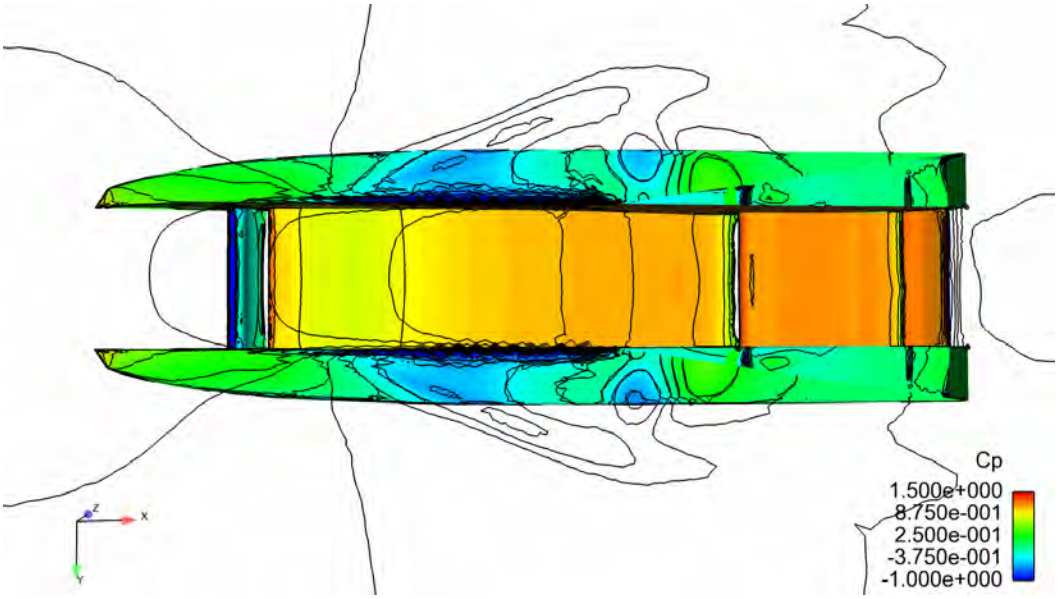


(a) The baseline 3D geometry.

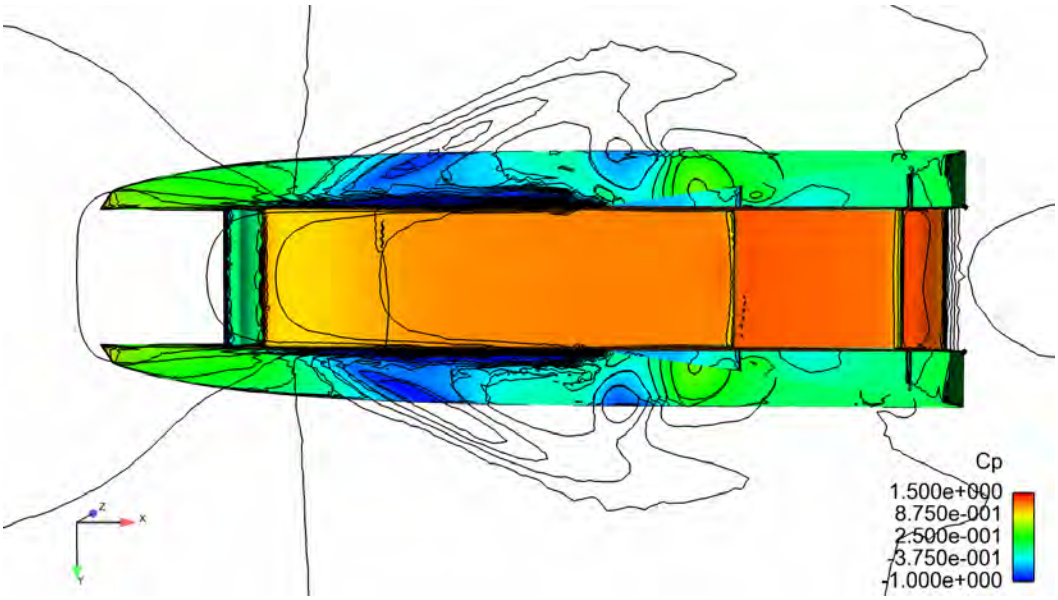


(b) The optimised 3D geometry.

Figure 4.24: The velocity profile from the VWT simulations for both geometries is shown. Large low velocity regions at the sharp edge of the extruded centrebody side indicate separation of flow.



(a) The baseline 3D geometry is seen from below.



(b) The optimised 3D geometry is seen from below.

Figure 4.25: The pressure coefficient distribution from the VWT simulations for both geometries is shown. An increase in pressure in the tunnel is visible in the optimised version.

4.6 XC10 optimisation conclusions and recommendations for future work

The AerOpt design process has successfully delivered a 30.9% improvement in lift generation of a 2D geometry of a high speed catamarans centrebody lift, translating to a 15.8% increase in lift for an extruded 3D equivalent.

The method is easily tailor-able to more complex, balanced objective functions (for example taking a weighted drag or aerodynamic pitch moment into account) depending on industry requirements. The method can run on parallel computer architectures and is relatively inexpensive, taking roughly one day of runtime across 21 cores per optimisation.

The method shows promise for performance optimisation of a real catamaran, though certain additional steps should be taken. A thorough analysis of the assumed total drag reduction by increasing the aerodynamic lift and reducing the water-wetted area should be made, ideally by using the hydrodynamic modelling capabilities detailed in Chapters 2 and 3. Longitudinal centre of pressure should be included in the objective function to avoid geometries that are impractically prone to longitudinal instability. A weighted drag term would also be desirable, though the optimum weighting would rely on an assessment of the trade-off between the aerodynamic lift and hydrodynamic drag, which would vary for different hullform configurations and designs.

Chapter 5

Thesis conclusions

5.1 Thesis conclusions

The objective of this work was the development of tools for commercial boat design by an industry partner, specifically in the more challenging arena of high-speed planing boats. Notable areas of opportunity were discovered in a literature review of the extant software, in particular costly commercial licensing and the current inefficient parallelisation on future exascale HPC clusters. For this reason, LBM (and Palabos specifically) was selected as the method of choice for this work due to its open-source nature and proven scalability to large numbers of compute nodes.

The development of the Palabos boatHullFormSolver programme delivers an open-source, license-free tool that has been validated in its capability to model the dynamic heave equilibrium of a planing monohull at high-speed. The full capabilities of boatHullFormSolver are reflected on in Section 5.1.1. Although no licensing cost is required to operate this tool, operational costs in terms of computational resources are a consideration which are explored in detail in Section 5.1.2. Recommendations for future work are given in Section 5.1.3.

Also presented in this thesis was a study on the aerodynamic optimisation of the centre-section of a high-speed catamaran. This work focused more on the design problem specific to a particular boat geometry presented by Norson Design, and as a consequence made best use of available tools to

turnaround a design to industry partners; it is for this reason that a RANS solver (FLITE2D) was used as a departure from LBM/Palabos. The aerodynamic shape optimisation was successful in that it delivered significant improvements in the objective functions (30.9% lift increase in 2D), however the originally-planned experimental testing on a scale model to determine the resultant total drag reduction could not be achieved with industry during the project lifespan.

5.1.1 Capabilities of the Palabos boatHullFormSolver code

In Chapter 2 and 3 the capabilities of the water-phase CFD performed by the Palabos boatHullFormSolver are presented. These are summarised as follows:

- Open source and free of licensing costs.
- A standard computational domain and boundary conditions that define a "virtual tow tank".
- The ability to import a user-defined .STL geometry that is pre-processed in parallel with no meshing required.
- Single-phase free surface flow modelling based on the Palabos lattice Boltzmann kernel, executed in parallel.
- Simple user customisation of the parameters most likely to be changed between different simulations via an .XML file that does not require a recompile.
- Both static and dynamic boat geometries (dynamics limited to heave motion only).
- Determination of heave equilibrium of a high speed monohull validated against industry data.

5.1.2 Commercial viability of the Palabos boatHullFormSolver code

Regarding the delivery of a cost-effective tool to marine industry, an examination of the costs associated needs to be presented. The open-source nature of the underlying Palabos libraries means that no software licensing is necessary, which provides an inherent advantage over costly commercial software. Commercial CFD software costs can vary greatly depending on the wide range of package options available; some capabilities are added as optional extras, discounts may be applied for research collaboration, and support technical support packages may be required at cost. For this reason the cost to industry of licensing a CFD package is case-dependent. During the initial search for an appropriate CFD solution for this work, XFlow [76] was explored as an option and a quote was provided by FlowHD for a 128 core license at a commercial rate of £83,456, with a 50% discount applied due to being for academic research. License costs on the order of tens of thousands of pounds would simply be prohibitive for many small and medium size boat design companies. The free and open-source nature of the Palabos solver eliminates this cost.

The Palabos boatHullFormSolver code is computationally intensive, and compute time has an associated cost. Amazon Web Services (AWS) provide HPC services for commercial CFD, with costs being service-dependent. A cost estimation from AWS for the running of OpenFOAM (also open source) in the London region on a 64-bit Amazon Machine Image with 16GB memory and 8 virtual cores is $0.488USD per hour [129]. The $35kts$ and $45kts$ dynamic heave cases presented in Section 3.5, for example, ran for 144 hours of wall clock time on 700 cores on the Swansea Sunbird HPC cluster. Assuming a simple scaling of number of cores and runtime to the AWS service, the equivalent cost would be around $6,148.8USD. This is the estimated cost for a single simulation determining the boat heave equilibrium at a single velocity and trim angle combination. Whether this is cost-effective is dependent on the value placed on that information by industry, however it can be seen that the running of the presented software is far from cheap.

Some of the flaws that drive the computational expense of boatHullFormSolver are identified in the following section; if achieved, these changes could greatly improve the commercial viability of the boatHullFormSolver code.

5.1.3 Recommended future work for boatHullFormSolver

The boatHullFormSolver as presented in this work is computationally expensive, despite the underlying scalability of the Palabos solver discussed in Section 1.5.3. The cost of this expense is detailed in the context of industry use in the following Section 5.1.2. There are a number of areas in which improvement could drastically improve the accuracy and computational cost of the solver.

Firstly, the parallelism is undermined by the use of the serial Matlab script used to achieve the dynamics; this Matlab script reserves a compute node that spends the whole compute wall-clock time processing the Matlab script in serial, with the vast majority of this time spent waiting on the Palabos simulations to finish. Tasks such as reading of the geometry vertices, forces and pressures, and manipulation of the hullform geometry are all undertaken by this Matlab script. By implementing these processes natively in the C++ script of boatHullFormSolver.cpp, the parallel potential of the solver can be better taken advantage of. This is especially the case as the current implementation requires an initialisation of the Palabos simulation for every timestep of the dynamics solver: initialisation of Palabos simulations is known to have significantly worse parallel performance as compared to the compute stage [89]. The reason for not doing this in the first place was the expected additional time it would take to deliver this more complex implementation, and uncertainty that it would be deliverable in the project lifespan.

As mentioned in Section 2.2.1, no wall functions are implemented in the LES scheme. This is an obvious weakness in the solvers ability to accurately solve the flow conditions in the near-wall areas, and remains as a crucial piece of future work to ensure the solver is useful in general, though especially for modelling factors heavily influenced by the boundary layer, such as drag. As described in Section 2.2.1, this could be achieved by varying the Smagorinsky

constant, depending on spatial proximity to a wall, which is permitted by local nature of the dynamics class `SmagorinskyDynamics` [100]. This was attempted, but not successfully achieved, in this work.

Another improvement that would see increased accuracy at a reduced computational cost is grid refinement. Due to the limitation of a constant global grid size, simulations were unnecessarily finely resolved in areas far from the complex near-hull flow, incurring high computational cost for little to no gain. As a corollary, refinement around the bow spray and wake regions would allow for capture of the finer free surface interface behaviour such as whisker spray, which in the current implementation of the solver fall into subgrid scales and are not captured. This basic premise was understood at the outset, however was not pursued as FlowKit indicated grid refinement was a capability they were currently working on, and it was determined that reproducing the same capability in parallel would be a waste of time and effort. Unfortunately, FlowKit did not develop such a capability for the free surface modules used in this work, though mesh refinement is being actively worked on and may be readily applicable in the near future [128].

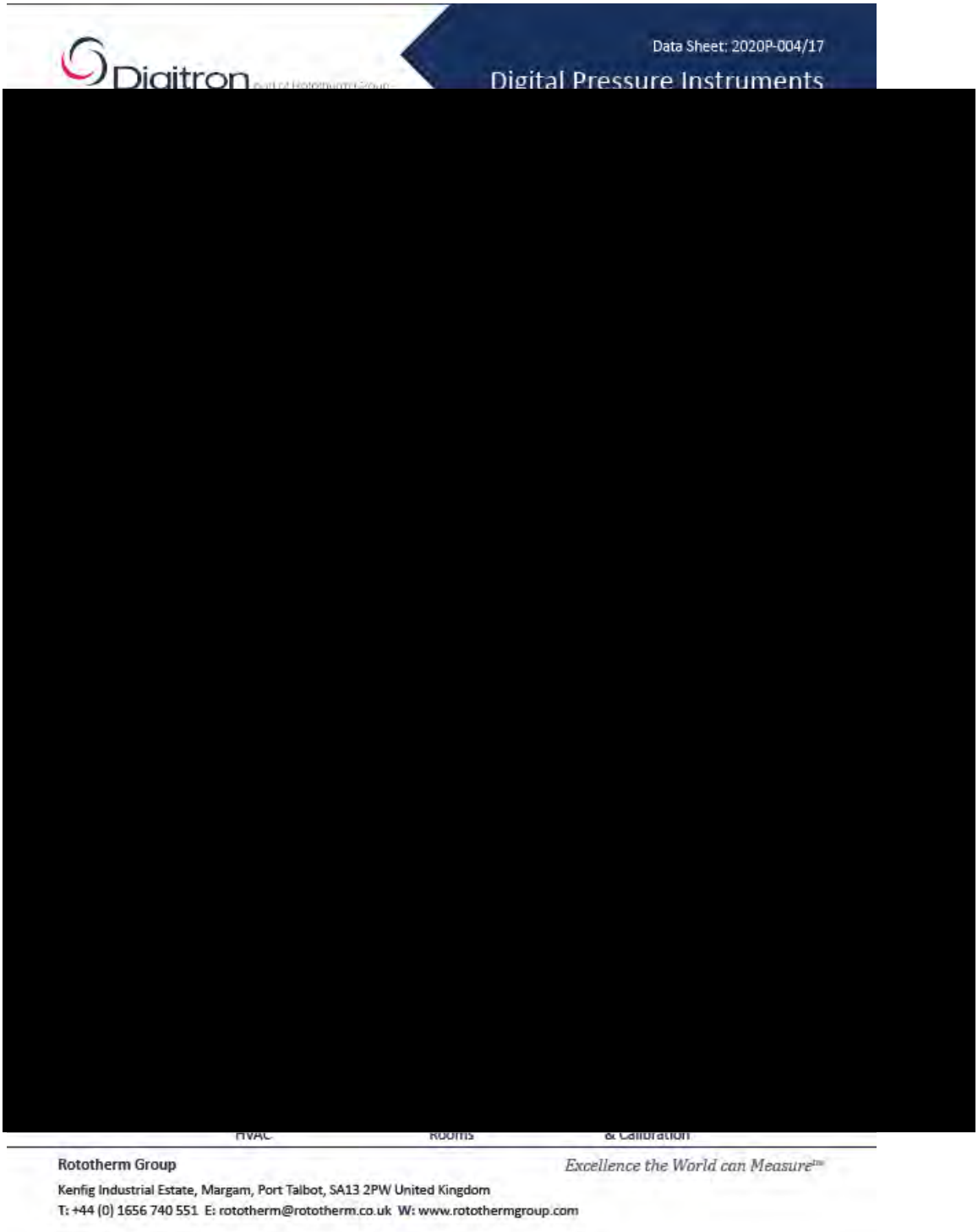
5.1.4 Closing statement

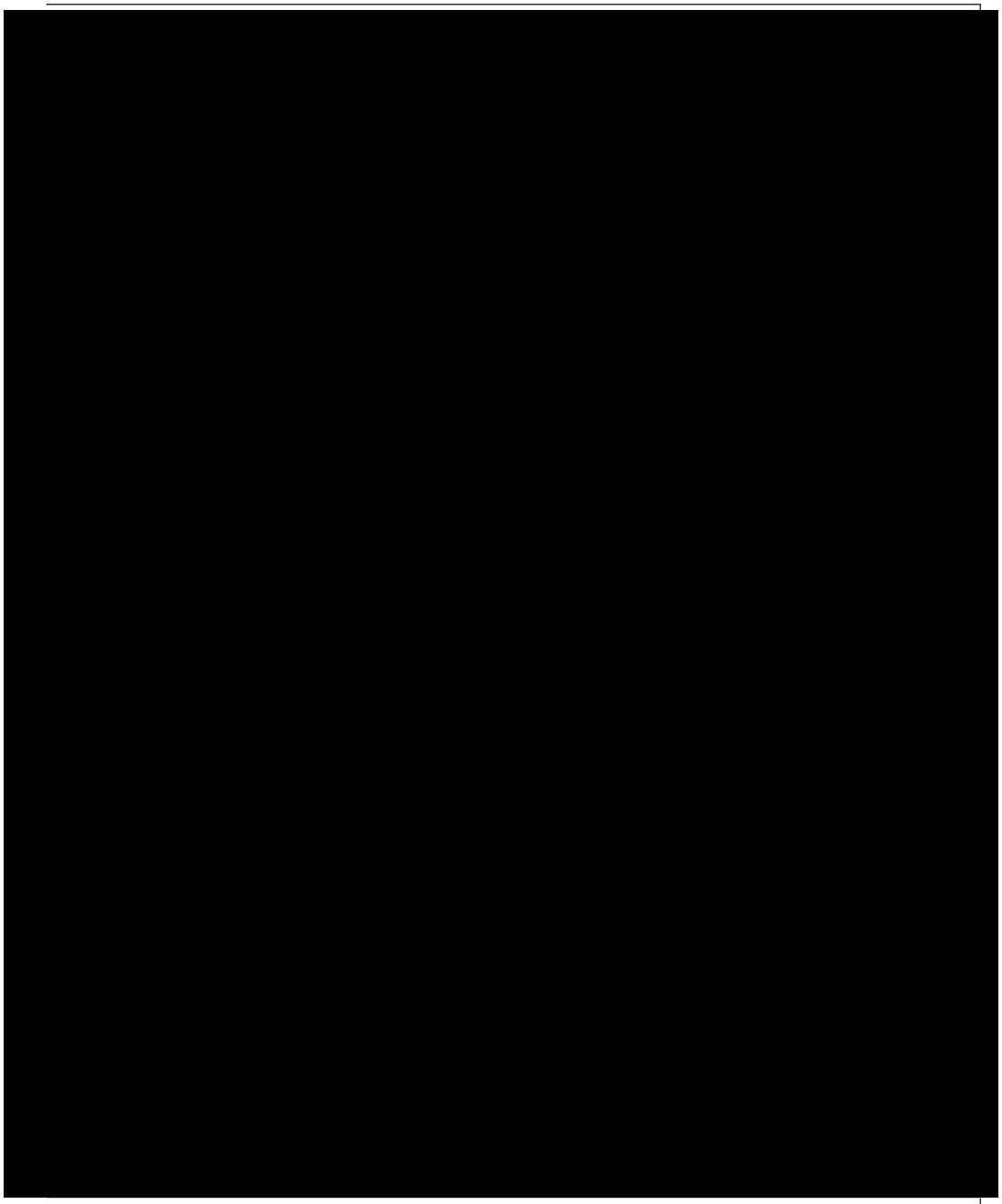
This thesis describes the development and testing of a "virtual tow tank" for the modelling and design of high-speed marine craft. The method shows good prediction of the heave equilibrium of high-speed hullform against industry data. It is reliant on purely open source and free software. Although it is computationally expensive in its current implementation, there exist obvious avenues to improving this in future work.

Appendix A

Appendices

A.1 Digitron 2020P Manometer Specifications

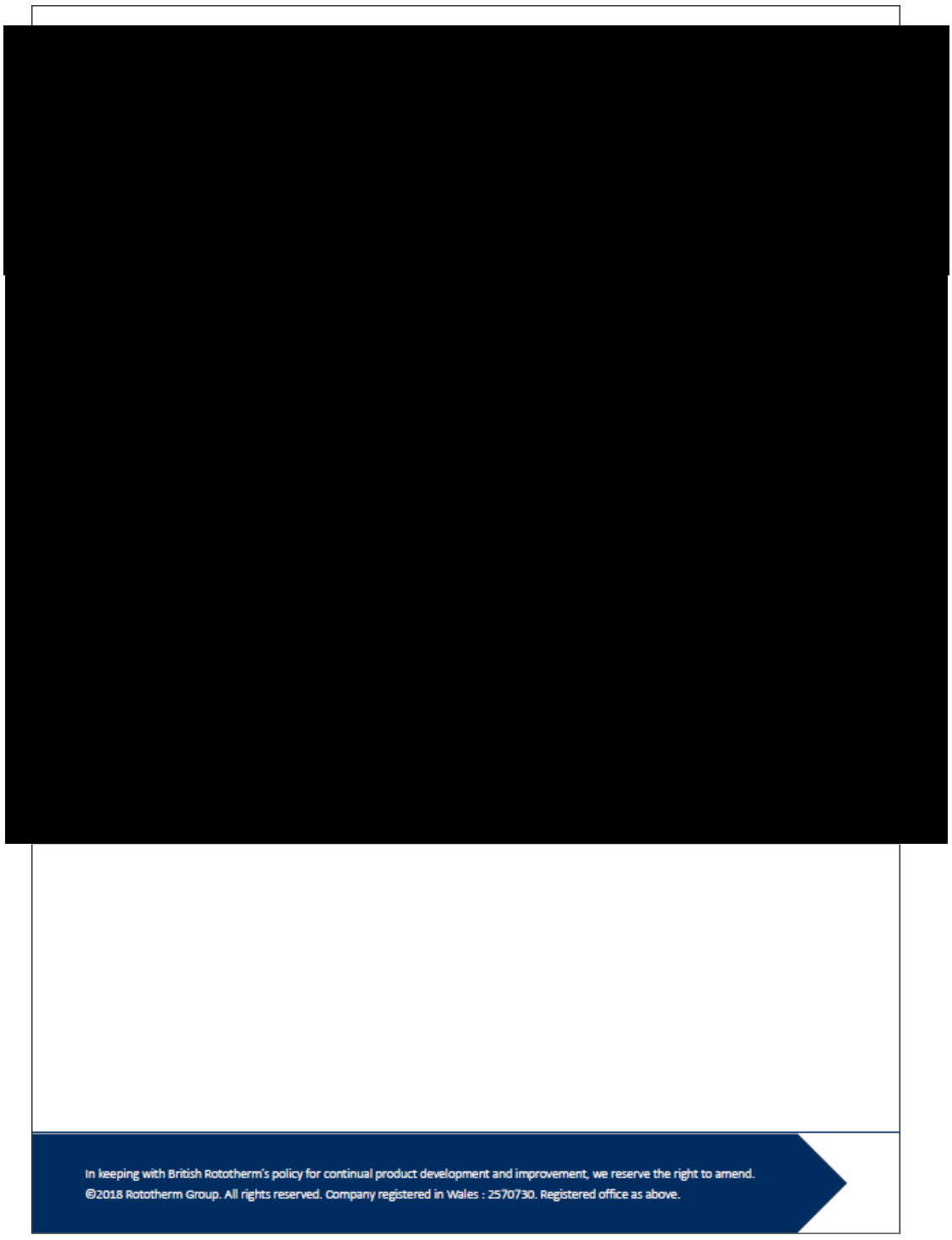




Rototherm Group *Excellence the World can Measure™*

Kenfig Industrial Estate, Margam, Port Talbot, SA13 2PW United Kingdom

T: +44 (0) 1656 740 551 **E:** rototherm@rototherm.co.uk **W:** www.rototherm group.com



A.2 Matlab files

A.2.1 jobSubmitter.m

```
%% RIGID BODY DYNAMICS JOB SUBMISSION SCRIPT.
% Handles job submission of the Palabos boatHullFormSolver executable and
% applies forward Euler method for rigid body dynamics.

% Instructions:
% The files the user will need to edit:
%     - <geometry_name>.stl
%         Have this in the working directory. Also create a copy of this
%         file of the format "<geometry_name>_init.stl". The init file will
%         not be overwritten, and is used to reset the position of the
%         other .stl after a completed run.
%     - params_template.xml
%         This file configures all the paramters for the Palabos sims.
%     - batchfile.job
%     - /tmp directory
%         Ensure this directory exists before running or an error will be
%         thrown. Simulation outputs will be written here.
%
% The user will also routinely need to edit the parameters in the following
% section before each submission.

%% USER INPUT - CHECK THESE BEFORE EACH RUN
% General.
filenameSTL = 'dv15';  % .stl filename, no file extension.
restarting = false;    % Restarting from a continue.xml file?

% Rigid body dynamics parameters.
maxTime =  5.0;    % Duration of dynamic simulation.
     dt =  0.005;  % Timestep of dynamic simulation (i.e. length of each
                   % constituent Palabos sim).
CofG = [-5.0;      % x-component.
         0.8;      % y-component.
         0.0;];    % z-component.
m_boat = 10000;    % Mass of boat (kg).
Iyy = 1;           % Second moment of area.

%% PRE-SUBMISSION.
% Calculate number of submits.
numOfSims = maxTime/dt;

% Read params_template for necessary values.
[charL, resolution, uLB, uRef, outIter, cpIter] = ...
    funcReadParamVals('params_template.xml');

% Calculate Palabos timesteps.
dx_palabos = charL/(resolution-1);
dt_palabos = uLB*dx_palabos/uRef;

% Calculate number of iterations required per Palabos sim to reach dynamics ...
    dt for each sim.
max_iter_per_sim = round(dt/dt_palabos);   % Each Palabos sim will run for ...
    this many iters to reach user-set dynamics dt.
stat_iter = round(max_iter_per_sim/100)+1; % Rate of Palabos iteration ...
    status write to output file.
```

```
out_iter  = round(max_iter_per_sim/5);     % Rate of Palabos iteration full ...
    output to disk.
cp_iter   = max_iter_per_sim;              % Rate of checkpointing (once at ...
    the end of each sim).

% If restarting, find the time at which the last sim finished.
if restarting
    [restartIter,restartTime] = funcReadLastSim(restarting);
else
    restartIter = 0; restartTime = 0;
end

% Initial values for dynamics.
    z = [0,0];     z_dot = [0,0];
theta = [0,0]; theta_dot = [0,0];

%% SUBMISSION LOOP
for i = 1+restartIter : numOfSims+restartIter

    % Calculate sim-specific iteration to terminate.
    max_iter_this_sim = max_iter_per_sim*i;
    cp_iter = max_iter_this_sim;
    out_iter = max_iter_this_sim;

    % Stream values to params.xml file from params_template.xml file.
    system('rm -f params.xml');
    system(sprintf('sed ''s/MAX_ITER/%d/; s/STAT_ITER/%d/; s/OUT_ITER/%d/; ...
        s/CP_ITER/%d/; s/YVEL/%d/;'' params_template.xml > ...
        params.xml',max_iter_this_sim+1,stat_iter,out_iter,cp_iter,z_dot(2) ...
        )); % +1 to max_iter_this_sim to ensure checkpointing occurs correctly.
    system(sprintf('echo ...
        ------------------------------------------------------------------------\n'
    system(sprintf('echo ITERATION %d:\n',i));
    system(sprintf('echo sim-%08d: running...',i));
    system(sprintf('echo sim-%08d: iterations: %d - ...
        %d\n',i,max_iter_this_sim-max_iter_per_sim,max_iter_this_sim));
    system(sprintf('echo sim-%08d:       time: %.3f - %.3f ...
        s\n',i,dt*i-dt,dt*i));

    % Execute solver.
    % On initial sim, do not use continue.xml as an input unless restarting ...
        from existing continue.xml file.
    if i == 1+restartIter && ~restarting
        system(sprintf('srun ./boatHullFormSolver params.xml > ...
            sim-%08d.out',i));
    else
        system(sprintf('srun ./boatHullFormSolver params.xml continue.xml > ...
            sim-%08d.out',i));
    end

    system(sprintf('echo sim-%08d: COMPLETE.',i));

    % Delete old checkpoint files (requires knowing last iter num).
    if i+restartIter > 1 % No checkpoint files on initial loop.
        % Get and format iteration number of sim i-1.
        iter_num_old = max_iter_this_sim-max_iter_per_sim;
        iter_num_old = num2str(iter_num_old,'%08.0f');
        % Use this to remove the old checkpoint file.
        system(sprintf('rm checkpoint_%s_*',iter_num_old));
    end
```

```
    % Move sim ouputs to /tmp.
    system(sprintf('mv sim-%08d.out tmp/',i));

    % DYNAMICS.
    % Read .stl vertices and vertex normals.
    [vertices, vertexNormals] = funcReadVertices(filenameSTL);
    % Read forces and pressures.
    [x_forceAvg, y_forceAvg, pressureCoords, pressureConnecs, pressureData] ...
       = ...
       funcReadForcesAndPressures(vertices,vertexNormals,max_iter_this_sim,out_iter);
    M_z = 0; % Not implemented.

    % Determine boat motion.
    [theta_diff, theta, theta_dot, z_diff, z, z_dot] = funcMoveBoat(dt, ...
       y_forceAvg, M_z, m_boat, Iyy, z, z_dot, theta, theta_dot);

    % Tranform current STL CofG to origin.
    [CofG, iterationNum] = funcManipulateSTL(filenameSTL, CofG, theta_diff, ...
       z_diff);
end

%% POST-PROCESSING
% Replace moved .stl file with initial .stl file for next run.
system(['cp ',filenameSTL,'_init.stl ',filenameSTL,'.stl']);
% Fill in Palabos output iterations that do not have a corresponding ...
   boat.stl file.
funcFillBoatStlGaps();
% Move the params.xml file to /tmp.
system('mv params.xml tmp/');

fprintf('\n\n');
fprintf('           %%-----------------------------------------%%\n');
fprintf('           The auto submission process is COMPLETE:-\n');
fprintf('           \n');
fprintf('                          Geometry: %s\n', filenameSTL);
fprintf('           \n');
fprintf('                    Starting time: %f s\n', restartTime);
fprintf('                   Finishing time: %f s\n', restartTime+maxTime);
fprintf('                      Total time: %f s\n', maxTime);
fprintf('             Time between restarts: %f s\n', dt);
fprintf('              Total number of sims: %d\n', numOfSims);
fprintf('           \n');
fprintf('           Palabos Stats:\n');
fprintf('                       Palabos dt: %f s\n', dt_palabos);
fprintf('                       Palabos dx: %f s\n', dx_palabos);
fprintf('                       Resolution: %d\n', resolution);
fprintf('             Characteristic length: %f m\n', charL);
fprintf('                     Lattice speed: %f m/s\n', uLB);
fprintf('                   Reference speed: %f m/s\n', uRef);
fprintf('            Rate of output to disk: %d iterations\n', out_iter);
fprintf('           \n');
fprintf('           \n');
fprintf('           %%-----------------------------------------%%\n');
fprintf('\n\n');
```

A.2.2 funcReadParamVals.m

```
function [characteristicLength, resolution, uLB, uRef, outIter, cpIter] = ...
    funcReadParamVals(paramsFilename)
% Search through .xml file to find important values for calculating ...
    iteration steps.

fid = fopen(paramsFilename,'r');
if fid == -1
    error('ERROR: cannot find params_template.xml.');
end

% Search for parameter name then extract the relevant number or string
% following that regexp. Note format should be:
%
%   <varName> varVal <varName>
%
% Note the spaces between the equals are important as the third space
% delimited item is taken.

% characteristicLength.
while ~feof(fid)
    line = fgetl(fid);
    lineIndex = regexp(line,'<characteristicLength>');
    if lineIndex
        characteristicLength = strsplit(line);
        characteristicLength = str2double(characteristicLength{3});
        break
    else
        % Not correct line, keep searching.
    end
end
frewind(fid);

% resolution.
while ~feof(fid)
    line = fgetl(fid);
    lineIndex = regexp(line,'<resolution>');
    if lineIndex
        resolution = strsplit(line);
        resolution = str2double(resolution{3});
        break
    else
        % Not correct line, keep searching.
    end
end
frewind(fid);

% uLB.
while ~feof(fid)
    line = fgetl(fid);
    lineIndex = regexp(line,'<uLB>');
    if lineIndex
        uLB = strsplit(line);
        uLB = str2double(uLB{3});
        break
    else
        % Not correct line, keep searching.
    end
```

```
end
frewind(fid);

% uRef.
while ~feof(fid)
    line = fgetl(fid);
    lineIndex = regexp(line,'<uRef>');
    if lineIndex
        uRef = strsplit(line);
        uRef = str2double(uRef{3});
        break
    else
        % Not correct line, keep searching.
    end
end
frewind(fid);

% outIter.
while ~feof(fid)
    line = fgetl(fid);
    lineIndex = regexp(line,'<outIter>');
    if lineIndex
        outIter = strsplit(line);
        outIter = str2double(outIter{3});
        break
    else
        % Not correct line, keep searching.
    end
end
frewind(fid);

% cpIter.
while ~feof(fid)
    line = fgetl(fid);
    lineIndex = regexp(line,'<cpIter>');
    if lineIndex
        cpIter = strsplit(line);
        cpIter = str2double(cpIter{3});
        break
    else
        % Not correct line, keep searching.
    end
end
frewind(fid);
```

A.2.3 funcReadLastSim.m

```
function [restartIter,restartTime] = funcReadLastSim(restarting)
% Reads the iteration value of the existing continue.xml file in the ...
    instance of restarting.

% Find latest sim-xxxxxxxx.out file in /tmp and open for reading.
simFiles = dir('tmp/sim-*');
simFiles = struct2cell(simFiles);
% Check if sim files exist in case of restarting.
if restarting && isempty(simFiles)
    fprintf('%s: No previous sim data found, check "restarting" ...
       value.\n',mfilename);
end
% Pick out restart iteration from sim files.
numSimFiles = size(simFiles);
simFilesLast = simFiles{1,numSimFiles(2)};
restartIter = str2num(cell2mat(regexp(simFilesLast,'\d*','Match')));

% Open file.
fid = fopen(sprintf('tmp/%s',simFilesLast),'r');
if fid == -1
    error('ERROR: cannot find latest sim file in tmp/');
end

% Skip to end of file.
offset = -750;
numLines = 10;
fseek(fid,offset,'eof');
text = cell(numLines,1);
for i = 1:numLines
    text(i) = {fgetl(fid)};
end

% Concatenate into one string.
textCat = '';
for i = 1: numLines
    textCat = strcat(textCat,text(i));
end
textCat = textCat{1};

% Search through to find 't = ' pattern and identify the time following it
% using regexp.
restartTime = str2double(regexp(textCat,'(?<= t = [^0-9]*)[0-9]*\.?[0-9]+', ...
    'match'));
% regexp usage:
% (?<= t = [^0-9]*) : start matching after 't = ' followed by non-number
% characters.
% [0-9]*            : then match 0 or more characters in 0-9 range.
% \.?               : allow for an optional decimal point.
% [0-9]+            : catch the number characters after the decimal.
```

A.2.4 funcReadVertices.m

```
function [vertices, vertexNormals] = funcReadVertices(filenameSTL)
%% Read vertices and their normals from the active .stl file.

% Read .stl file.
[vertices, faces, faceNormals, nameSTL] = stlRead([filenameSTL,'.stl']);

% Get vertex normals.
fprintf('%s: Reading .stl vertices and vertex normals.\n',mfilename);
vertexNormals = zeros(size(vertices));
% Add the normals for each point of every triangle.
vertexNormals(faces(:,1),:) = vertexNormals(faces(:,1),:) + faceNormals;
vertexNormals(faces(:,2),:) = vertexNormals(faces(:,2),:) + faceNormals;
vertexNormals(faces(:,3),:) = vertexNormals(faces(:,3),:) + faceNormals;
% Calculate vertex normals.
vertexNormals = vertexNormals./repmat(sqrt(sum(vertexNormals.^2,2)),[1,3]);
vertexNormals = -vertexNormals; % Invert to point inward of volume.

end
```

A.2.5 funcReadForcesAndPressures.m

```
function [x_forceAvg, y_forceAvg, pressureCoords, pressureConnecs, ...
    pressureData] = ...
    funcReadForcesAndPressures(vertices,vertexNormals,max_iter_this_sim,out_iter)
%% Reads forces and pressures from Palabos outputs.
% Forces are contained in total_force_on_boat.dat file in /tmp.
% Pressures are contained in boat_pressure_xxxxxxxx.vtk (xxxxxxxx =
% iteration number of last output).

%% READ FORCES.
fid = fopen('tmp/total_force_on_boat.dat');
if fid == -1
    error('%s: ERROR tmp/total_forces_on_boat.dat not found.',mfilename);
end
fprintf('%s: Reading forces...\n',mfilename);

% Determine how many outputs are made during this Palabos simulation.
num_outputs_this_sim = floor(max_iter_this_sim/out_iter);
% Take the last 10% of force values to be averaged.
avgRange = floor(num_outputs_this_sim*0.1);
% Rounded down, but must avoid zero val.
if avgRange == 0
    avgRange = 1;
end

% Count total lines in force file.
totalLines = 0;
while ~feof(fid)
    line = fgetl(fid);
    totalLines = totalLines + 1;
end
beginRange = totalLines - avgRange;

% Rewind and count through to averaging range, pick values to be averaged.
frewind(fid);
lineCount = 0; avgIndex = 1;
x_forcesToAvg = zeros(avgRange,1);
y_forcesToAvg = zeros(avgRange,1);
while ~feof(fid)
    if lineCount < beginRange
        line = fgetl(fid);
        lineCount = lineCount + 1;
    else
        line = strsplit(fgetl(fid));
        x_forcesToAvg(avgIndex,1) = str2double(line{1,3});
        y_forcesToAvg(avgIndex,1) = str2double(line{1,4});
        lineCount = lineCount + 1;
        avgIndex = avgIndex + 1;
    end
end

% Average the values.
x_forceAvg = sum(x_forcesToAvg)/length(x_forcesToAvg);
y_forceAvg = sum(y_forcesToAvg)/length(y_forcesToAvg);
fprintf('%s: The average DRAG for the last %d Palabos iterations is ...
    %fN.\n',mfilename,length(x_forcesToAvg),x_forceAvg);
fprintf('%s: The average LIFT for the last %d Palabos iterations is ...
    %fN.\n',mfilename,length(y_forcesToAvg),y_forceAvg);
```

```
%% READ PRESSURES.
% First need to identify which is the latest boat pressure .vtk file.
% Format is:    boat_pressure_xxxxxxxx.vtk
% Where xxxxxxxx is an eight-digit number of the last Palabos iteration at
% which output files were written.

% Find latest pressure.vtk file.
pressureFiles = dir('tmp/boat_pressure_*');
filenamePressure = pressureFiles(length({pressureFiles.name})).name;

% Open this file to read pressure data.
fid = fopen(['tmp/',filenamePressure]);
if fid == -1
    error('%s: ERROR tmp/boat_pressure_xxxxxxxx.vtk not found.',mfilename);
end

% Read number of points.
regexpFound = [];
while isempty(regexpFound)
    line = fgetl(fid);
    regexpFound = regexp(line,'POINTS');
    if regexpFound
        % Found line for numPoints.
        numPoints = strsplit(line);
        numPoints = str2double(numPoints{2});
    end
end

% Read all point coords.
fprintf('%s: Reading pressure coords...\n',mfilename);
pressureCoords = zeros(numPoints,3);
for i = 1:numPoints
    line = strsplit(fgetl(fid));
    pressureCoords(i,:) = str2double(line);
end

% Skip a line.
fgetl(fid);

% Read number of connectivities.
line = fgetl(fid);
numConnecs = strsplit(line);
numConnecs = str2double(numConnecs{2});

% Read connectivities.
fprintf('%s: Reading connectivities...\n',mfilename);
pressureConnecs = zeros(numConnecs,4);
for i = 1:numConnecs
    line = strsplit(fgetl(fid));
    pressureConnecs(i,:) = str2double(line);
end
pressureConnecs = pressureConnecs(:,2:4); % First column is redundant.

% Read up to "POINT_DATA" and check that numPoint == numPressures.
regexpFound = [];
while isempty(regexpFound)
    line = fgetl(fid);
    regexpFound = regexp(line, 'POINT_DATA');
    if regexpFound
```

```
        % Found line for numPressures.
        numPressures = strsplit(line);
        numPressures = str2double(numPressures{2});
    end
end
if numPoints ~= numPressures
    error('%s: Number of points does not match number of pressures in ...
       boat_pressure.vtk file',mfilename);
end
% Skip two lines.
fgetl(fid); fgetl(fid);

fprintf('%s: Reading pressure data...\n',mfilename);
% Read all pressure data.
pressureData = zeros(numPoints,1);
for i = 1:numPoints
    line = strsplit(fgetl(fid));
    pressureData(i,:) = str2double(line);
    if mod(i,1000000) == 0
      fprintf('%s: 1,000,000 of %d pressures read.\n',mfilename,numPoints);
    end
end

fclose(fid);

end
```

A.2.6 funcMoveBoat.m

```
function [theta_diff, theta, theta_dot, z_diff, z, z_dot] = funcMoveBoat(dt, ...
    y_forceAvg, M_z, m_boat, Iyy, z, z_dot, theta, theta_dot)
% Determine 2DoF boat motion (heave and pitch) given the global timestep and ...
    force/moment.

% Accel due to grav at sea level.
g = 9.81;

% Previous values of z and z_dot are 1x2 vectors:
%     z(1) = z_i    and    z(2) = z_i+1
% And likewise for z_dot.

% Heave.
        F_z = y_forceAvg;
delta_z_dot = ((F_z-m_boat*g)/m_boat)*dt;
   z_dot(2) = z_dot(1) + delta_z_dot;
       z(2) = z(1) + z_dot(1)*dt;

% funcManipulateSTL.m operates about the boat's previous position, not ...
    global coords, so the difference between z values is needed.
z_diff = z(2) - z(1);

% Iterate z values.
z_dot(1) = z_dot(2);
    z(1) = z(2);

% Pitch (NOT IMPLEMENTED).
theta_diff = 0;
```

A.2.7 funcManipulateSTL.m

```
function [CofG, iterationNum] = funcManipulateSTL(filenameSTL, CofG, theta, z)
%% Manipulates the .stl geometry by rotating and transforming in z-direction ...
   according to dynamics.
%  1. Read in .stl file (binary or ASCII).
%  2. Move whole .stl so that the CofG is at the origin.
%  3. Perform rotation.
%  4. Tranform back to pre-rotation position.
%  5. Perform z translation.
%  6. Tranform CofG also.
%  7. Write to new .stl file and store a copy for post-processing later.

%% 1. Read .stl file.
[vertices, faces, faceNormals, nameSTL] = stlRead([filenameSTL,'.stl']);

%% 2. Move whole .stl file so that the CofG is at the origin.
vertices(:,1) = vertices(:,1) - CofG(1);
vertices(:,2) = vertices(:,2) - CofG(2);
vertices(:,3) = vertices(:,3) - CofG(3);

%% 3. Perform rotation.
% Define rotation matrix for 3D rotation about the z-axis.
Rz = [ cosd(theta), sind(theta), 0  ;
      -sind(theta), cosd(theta), 0  ;
                 0,           0, 1 ];
% Apply rotation to the vertices.
vertices = vertices * Rz';

%% 4. Return to pre-rotation position.
vertices(:,1) = vertices(:,1) + CofG(1);
vertices(:,2) = vertices(:,2) + CofG(2);
vertices(:,3) = vertices(:,3) + CofG(3);

%% 5. Z-translation.
vertices(:,2) = vertices(:,2) + z;

%% 6. Transform CofG identically (no rotation required as rotation is ...
   effectively about CofG).
CofG(2) = CofG(2) + z;

%% 7. Write .stl files.
% Write an updated .stl file out in working directory.
stlWrite([filenameSTL,'.stl'], faces, vertices);
% Get iteration number from latest interface file to be written.
interfaceFiles = dir('tmp/interface_*');
iterationNum = interfaceFiles(length({interfaceFiles.name})).name;
iterationNum = iterationNum(regexp(iterationNum,'\d'));
% Store a copy in /stl_stored for paraview post-processing.
filenameSTL_iter = [filenameSTL,'_',iterationNum];
stlWrite(['tmp/',filenameSTL_iter,'.stl'], faces, vertices);

% Print iteration number out while we have it.
fprintf('     %s: Palabos iteration: %s\n',mfilename,num2str(iterationNum));

end
```

A.2.8 funcFillBoatStlGaps.m

```
function [] = funcFillBoatStlGaps()
% Boat geometry .stl files are only created and save when boat movement
% occurs. This can leave gaps in the output files where pressures .vtk
% files or interface .stl files do not have a corresponding boat.stl file.
% This function fixes that as a post-processing step.
%     1. Read from params.xml the frequency of Palabos output to disk and
%        boat.stl file name.
%     2. Loop through all iteration numbers, noting where matching boat.stl
%        files occur and where spaces are.
%     3. For every iteration number a boat.stl file does not exist for,
%        copy to a file of that iteration number name the previous existing
%        boat.stl.

% 1. Read from params.xml.
fid = fopen('params.xml');
if fid == -1
    error('ERROR(funcFillBoatStlGaps.m): params.xml not found.');
end

% Get boat.stl name and frequency of Palabos output to disk.
while ~feof(fid)
    line = fgetl(fid);
    lineBoatName = regexp(line,'boatStl');
    lineOutIter  = regexp(line,'outIter');
    if lineBoatName
        boatName = strsplit(line);
        boatName = boatName{3};
    elseif lineOutIter
        outIter = strsplit(line);
        outIter = str2double(outIter{3});
    else
        % If neither found, do nothing.
    end
end

% Remove .stl extension from boatName.
boatName = boatName(1:length(boatName)-4);

% Count number of output files in /tmp.
interfaceFiles = dir('tmp/interface_*');
numPalabosOutputs = (length({interfaceFiles.name}));

% List boat.stl files in /tmp.
boatStlFiles = dir(['tmp/',boatName,'_*']);
boatStlFileNames = struct2cell(boatStlFiles);
boatStlFileNames = boatStlFileNames(1,:)';

% 2. Loop through all iterations checking if a corresponding boat.stl file
% exists.
matchingBoatStl = zeros(numPalabosOutputs,1); % Pre-allocating.
for i = 1:numPalabosOutputs
    palabosIterNum = i*outIter-outIter;
    palabosIterNumPadded = num2str(palabosIterNum,'%08.0f');
    matchedIndex = regexp(boatStlFileNames,palabosIterNumPadded);
    if isempty(cell2mat(matchedIndex))
        % No boat.stl file exists for this palabos iteration - index.
        matchingBoatStl(i,1) = false;
```

```
    else
        % A boat.stl files already exists for this iteration - index.
        matchingBoatStl(i,1) = true;
    end
end

% 3. Fill in non-matching cases by copying forward the previous boat.stl file.
% Note: a special exception will be to be made for the first gap.
previousMatch = boatName; j = 1;
for i = 1:length(matchingBoatStl)
    palabosIterNum = i*outIter-outIter;
    palabosIterNumPadded = num2str(palabosIterNum,'%08.0f');
    if matchingBoatStl(i,1)
        % Match - set the corresponding filename as the new latest match.
        previousMatch = cell2mat(boatStlFileNames(j,1));
        j = j + 1;
    else
        % No match - copy previousMatch file to a new boat.stl name with
        % appropriate iteration number.
        if strcmp(previousMatch,boatName)
            % Special case, copy main boat.stl file to iternumbered file.
            system(['cp ',boatName,'.stl ...
               tmp/',boatName,'_',palabosIterNumPadded,'.stl']);
        else
            system(['cp tmp/',previousMatch,' ...
               tmp/',boatName,'_',palabosIterNumPadded,'.stl']);
        end
    end
end
```

A.3 Palabos files

A.3.1 boatHullFormSolver.cpp

```
/* This file is part of the Palabos library.
 *
 * Copyright (C) 2011-2017 FlowKit Sarl
 * Route d'Oron 2
 * 1010 Lausanne, Switzerland
 * E-mail contact: contact@flowkit.com
 *
 * The most recent release of Palabos can be downloaded at
 * <http://www.palabos.org/>
 *
 * The library Palabos is free software: you can redistribute it and/or
 * modify it under the terms of the GNU Affero General Public License as
 * published by the Free Software Foundation, either version 3 of the
 * License, or (at your option) any later version.
 *
 * The library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU Affero General Public License for more details.
 *
 * You should have received a copy of the GNU Affero General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */
```

```
 emph# emphinclude "palabos3D.h"
 emph# emphinclude "palabos3D.hh"

 emph# emphinclude "pypal/headers3D.h"
 emph# emphinclude "pypal/headers3D.hh"

 emph# emphinclude <algorithm>
 emph# emphinclude <cstdlib>
 emph# emphinclude <cmath>
 emph# emphinclude <string>

using namespace plb;

typedef double T;

 emph# emphdefine WAVE_ABSORPTION

 emph# emphifdef WAVE_ABSORPTION
 emph# emphdefine DESCRIPTOR descriptors::AbsorbingWaveD3Q19Descriptor
 emph# emphelse
 emph# emphdefine DESCRIPTOR descriptors::D3Q19Descriptor
 emph# emphendif

// Descriptor used for post-processing smoothing
 emph# emphdefine SM_DESCRIPTOR descriptors::AdvectionDiffusionD3Q7Descriptor

 emph# emphdefine OUT     0
 emph# emphdefine IN      1

struct SimulationParameters {

    /*
     * Parameters set by the user.
     * All user input variables and all data in external input files must be ...
        in the same system of units.
     */

    T yVel;
    std::vector<T> xDomain;                     // Extent in the x-direction ...
       of the physical simulation domain.
    std::vector<T> yDomain;                     // Extent in the y-direction ...
       of the physical simulation domain.
    std::vector<T> zDomain;                     // Extent in the z-direction ...
       of the physical simulation domain.
    T inletAbsorbingZoneWidth;                  // Absorbing zone widths.
    T outletAbsorbingZoneWidth;
    T lateralAbsorbingZoneWidth;
    T topAbsorbingZoneWidth;
    T fluidHeight;                              // Initial height of the fluid.

    std::string boatStl;                        // Name of the file with the ...
       boat hull form geometry.

    T rho;                                      // Fluid density in physical ...
       units.
    T nu;                                       // Fluid kinematic viscosity ...
       in physical units.
    T surfaceTension;                           // Surface tension ...
       coefficient in physical units.
```

```
T inflationParameter;                          // Parameter for the voxelizer.

T characteristicLength;                        // Length to define dx.
plint resolution;                              // Total number of lattice ...
    nodes in the characteristic length.

T inletVelocity;                               // Inlet BC.

T u_Ref;                                       // Reference velocity.
T u_LB;                                        // Lattice velocity.

T A;                                           // Wave generation force ...
    parameters.
T P;
std::vector<T> xWaveDomain;
std::vector<T> yWaveDomain;
std::vector<T> zWaveDomain;

plint maxIter;                                 // Maximum number of iterations.

T cSmago;                                      // Smagorinsky parameter.

bool strongRepelling;                          // Parameter for the ...
    Immersed Boundary method.

T ambientPressure;                             // Absolute pressure at ...
    infinity.

std::string outDir;                            // Output directory.
plint statIter;                                // Number of iterations for ...
    terminal output.
plint outIter;                                 // Number of iterations for ...
    disk output.
plint cpIter;                                  // Number of iterations for ...
    checkpointing.
plint abIter;                                  // Number of iterations for ...
    checking for user-driven program abortion.

bool excludeInteriorForOutput;                 // Exclude the interior of ...
    the boat for pressure and force output or not?
int numPresLaplaceIter;                        // Number of Laplacian ...
    smoothing iterations for the pressure post-processing.

bool outputInDomain;                           // Save data on disk in a ...
    volume domain or not?
Cuboid<T> outputCuboid;                        // Volume domain for disk ...
    output.

bool outputOnSlices;                           // Save data on disk on a ...
    set of slices or not?
std::vector<T> xPositions;                     // Positions of the x-slices ...
    for output.
std::vector<T> xyRange;                        // y range of the x-slices.
std::vector<T> xzRange;                        // z range of the x-slices.
std::vector<T> yPositions;                     // Positions of the y-slices ...
    for output.
std::vector<T> yzRange;                        // z range of the y-slices.
std::vector<T> yxRange;                        // x range of the y-slices.
std::vector<T> zPositions;                     // Positions of the z-slices ...
```

```
        for output.
    std::vector<T> zxRange;                         // x range of the z-slices.
    std::vector<T> zyRange;                         // y range of the z-slices.

    std::string abortFileName;                      // File for signaling ...
        program abortion.
    std::string xmlContinueFileName;                // XML file for restarting.
    std::string baseFileName;                       // Basename of the ...
        checkpoint files.
    bool useParallelIO;                             // For a desktop PC this ...
        should be "false", for a cluster "true".

    /*
     * Parameters NOT set by the user.
     */

    T yVel_LB;
    plint nx, ny, nz;
    plint fluidHeight_LB;
    Box3D fullDomain;
    Box3D bottom, top;
    Box3D initialFluidDomain;
    T dx;
    T dt;
    T rho_LB;
    Array<plint,6> numAbsorbingCells;
    plint totalNumAbsorbingCells;
    T surfaceTension_LB;
    T inletVelocity_LB;
    T initialVelocity_LB;
    T gravity_LB;
    T A_LB;
    T P_LB;
    Box3D waveDomain;
    T omega;
    bool incompressibleModel;
    Array<T,3> physicalLocation;
    RawConnectedTriangleMesh<T>* connectedMesh;
    std::vector<Array<T,3> > vertices;
    std::vector<T> areas;
    std::vector<int> flags;
    Box3D outputDomain;
    std::vector<Box3D> xSlices;
    std::vector<Box3D> ySlices;
    std::vector<Box3D> zSlices;
    std::vector<Box3D> allOutputDomains;
    std::vector<std::string> allOutputDomainNames;
    bool saveDynamicContent;
    plint fileNamePadding;
} param;

T toPhys(T lbVal, plint direction, T dx, Array<T,3> const& location)
{
    PLB_ASSERT(direction >= 0 && direction <= 2);
    return (lbVal * dx + location[direction]);
}

Array<T,3> toPhys(Array<T,3> const& lbVal, T dx, Array<T,3> const& location)
{
    return (lbVal * dx + location);
```

```
}

T toLB(T physVal, plint direction, T dx, Array<T,3> const& location)
{
    PLB_ASSERT(direction >= 0 && direction <= 2);
    return (physVal - location[direction]) / dx;
}

Array<T,3> toLB(Array<T,3> const& physVal, T dx, Array<T,3> const& location)
{
    return (physVal - location) / dx;
}

void setParameters(std::string xmlInputFileName)
{
    XMLreader document(xmlInputFileName);

    document["geometry"]["simulationDomain"]["x"].read(param.xDomain);
    plbIOError(param.xDomain.size() != 2 || ...
       util::lessEqual(param.xDomain[1], param.xDomain[0]),
            "The x-extent of the simulation domain is wrong.");
    document["geometry"]["simulationDomain"]["y"].read(param.yDomain);
    plbIOError(param.yDomain.size() != 2 || ...
       util::lessEqual(param.yDomain[1], param.yDomain[0]),
            "The y-extent of the simulation domain is wrong.");
    document["geometry"]["simulationDomain"]["z"].read(param.zDomain);
    plbIOError(param.zDomain.size() != 2 || ...
       util::lessEqual(param.zDomain[1], param.zDomain[0]),
            "The z-extent of the simulation domain is wrong.");

    document["geometry"]["inletAbsorbingZoneWidth"].read(param.inletAbsorbingZoneWidth);
    document["geometry"]["outletAbsorbingZoneWidth"].read(param.outletAbsorbingZoneWidth);
    document["geometry"]["lateralAbsorbingZoneWidth"].read(param.lateralAbsorbingZoneWidth
    document["geometry"]["topAbsorbingZoneWidth"].read(param.topAbsorbingZoneWidth);

    document["geometry"]["fluidHeight"].read(param.fluidHeight);

    document["geometry"]["boatStl"].read(param.boatStl);

    document["fluid"]["rho"].read(param.rho);
    document["fluid"]["nu"].read(param.nu);
    document["fluid"]["surfaceTension"].read(param.surfaceTension);

    document["solver"]["yVel"].read(param.yVel);
    document["solver"]["inflationParameter"].read(param.inflationParameter);
    plbIOError(param.inflationParameter < (T) 0 || param.inflationParameter ...
       > (T) 1,
            "The inflationParameter must take values between 0 and 1.");

    document["solver"]["characteristicLength"].read(param.characteristicLength);
    document["solver"]["resolution"].read(param.resolution);

    document["solver"]["inletVelocity"].read(param.inletVelocity);

    document["solver"]["uRef"].read(param.u_Ref);
    document["solver"]["uLB"].read(param.u_LB);

    document["solver"]["A"].read(param.A);
    document["solver"]["P"].read(param.P);
    document["solver"]["waveDomain"]["x"].read(param.xWaveDomain);
```

```
plbIOError(param.xWaveDomain.size() != 2 || ...
    util::lessEqual(param.xWaveDomain[1], param.xWaveDomain[0]),
        "The x-extent of the wave domain is wrong.");
document["solver"]["waveDomain"]["y"].read(param.yWaveDomain);
plbIOError(param.yWaveDomain.size() != 2 || ...
    util::lessEqual(param.yWaveDomain[1], param.yWaveDomain[0]),
        "The y-extent of the wave domain is wrong.");
document["solver"]["waveDomain"]["z"].read(param.zWaveDomain);
plbIOError(param.zWaveDomain.size() != 2 || ...
    util::lessEqual(param.zWaveDomain[1], param.zWaveDomain[0]),
        "The z-extent of the wave domain is wrong.");

document["solver"]["maxIter"].read(param.maxIter);

document["solver"]["cSmago"].read(param.cSmago);

document["solver"]["strongRepelling"].read(param.strongRepelling);

document["solver"]["ambientPressure"].read(param.ambientPressure);

std::string outDir;
document["output"]["outDir"].read(outDir);
if (outDir[outDir.size() - 1] != '/') {
    outDir += '/';
}
param.outDir = outDir;
abortIfCannotCreateFileInDir(param.outDir, "plb-checkfile.txt");

document["output"]["statIter"].read(param.statIter);
document["output"]["outIter"].read(param.outIter);
document["output"]["cpIter"].read(param.cpIter);
document["output"]["abIter"].read(param.abIter);

document["output"]["excludeInteriorForOutput"].read(param.excludeInteriorForOutput);
document["output"]["numPresLaplaceIter"].read(param.numPresLaplaceIter);

document["output"]["outputInDomain"].read(param.outputInDomain);
if (param.outputInDomain) {
    std::vector<plint> x, y, z;
    document["output"]["outputDomain"]["x"].read(x);
    plbIOError(x.size() != 2 || util::lessEqual(x[1], x[0]), "The ...
       x-extent of the outputDomain is wrong");
    document["output"]["outputDomain"]["y"].read(y);
    plbIOError(y.size() != 2 || util::lessEqual(y[1], y[0]), "The ...
       y-extent of the outputDomain is wrong");
    document["output"]["outputDomain"]["z"].read(z);
    plbIOError(z.size() != 2 || util::lessEqual(z[1], z[0]), "The ...
       z-extent of the outputDomain is wrong");
    param.outputCuboid.lowerLeftCorner[0] = x[0];
    param.outputCuboid.lowerLeftCorner[1] = y[0];
    param.outputCuboid.lowerLeftCorner[2] = z[0];
    param.outputCuboid.upperRightCorner[0] = x[1];
    param.outputCuboid.upperRightCorner[1] = y[1];
    param.outputCuboid.upperRightCorner[2] = z[1];
}

document["output"]["outputOnSlices"].read(param.outputOnSlices);
if (param.outputOnSlices) {
    document["output"]["outputSlices"]["xSlices"]["xPositions"].read(param.xPositions)
    document["output"]["outputSlices"]["xSlices"]["yRange"].read(param.xyRange);
```

```
        plbIOError(param.xyRange.size() != 2 || ...
            util::lessEqual(param.xyRange[1], param.xyRange[0]),
                "The y-range of the x-slices is wrong");
        document["output"]["outputSlices"]["xSlices"]["zRange"].read(param.xzRange);
        plbIOError(param.xzRange.size() != 2 || ...
            util::lessEqual(param.xzRange[1], param.xzRange[0]),
                "The z-range of the x-slices is wrong");

        document["output"]["outputSlices"]["ySlices"]["yPositions"].read(param.yPositions)
        document["output"]["outputSlices"]["ySlices"]["zRange"].read(param.yzRange);
        plbIOError(param.yzRange.size() != 2 || ...
            util::lessEqual(param.yzRange[1], param.yzRange[0]),
                "The z-range of the y-slices is wrong");
        document["output"]["outputSlices"]["ySlices"]["xRange"].read(param.yxRange);
        plbIOError(param.yxRange.size() != 2 || ...
            util::lessEqual(param.yxRange[1], param.yxRange[0]),
                "The x-range of the y-slices is wrong");

        document["output"]["outputSlices"]["zSlices"]["zPositions"].read(param.zPositions)
        document["output"]["outputSlices"]["zSlices"]["xRange"].read(param.zxRange);
        plbIOError(param.zxRange.size() != 2 || ...
            util::lessEqual(param.zxRange[1], param.zxRange[0]),
                "The x-range of the z-slices is wrong");
        document["output"]["outputSlices"]["zSlices"]["yRange"].read(param.zyRange);
        plbIOError(param.zyRange.size() != 2 || ...
            util::lessEqual(param.zyRange[1], param.zyRange[0]),
                "The x-range of the z-slices is wrong");
    }

    document["output"]["abortFileName"].read(param.abortFileName);
    document["output"]["xmlContinueFileName"].read(param.xmlContinueFileName);
    document["output"]["baseFileName"].read(param.baseFileName);
    document["output"]["useParallelIO"].read(param.useParallelIO);
}

void computeOutputDomain(Cuboid<T> const& cuboid, Box3D& box)
{
    if (!param.outputInDomain) {
        return;
    }

    Array<T,3> llc = cuboid.lowerLeftCorner;
    Array<T,3> urc = cuboid.upperRightCorner;

    plint x0 = util::roundToInt(toLB(llc[0], 0, param.dx, ...
        param.physicalLocation));
    plint y0 = util::roundToInt(toLB(llc[1], 1, param.dx, ...
        param.physicalLocation));
    plint z0 = util::roundToInt(toLB(llc[2], 2, param.dx, ...
        param.physicalLocation));

    plint x1 = util::roundToInt(toLB(urc[0], 0, param.dx, ...
        param.physicalLocation));
    plint y1 = util::roundToInt(toLB(urc[1], 1, param.dx, ...
        param.physicalLocation));
    plint z1 = util::roundToInt(toLB(urc[2], 2, param.dx, ...
        param.physicalLocation));

    PLB_ASSERT(x1 >= x0 && y1 >= y0 && z1 >= z0);
```

```
    box = Box3D(x0, x1, y0, y1, z0, z1);

 emph# emphifdef PLB_DEBUG
    bool intersectionOK =
 emph# emphendif
        intersect(box, param.fullDomain, box);
    PLB_ASSERT(intersectionOK);
}

void computeOutputSlices()
{
    if (!param.outputOnSlices) {
        return;
    }

    {
        param.xSlices.clear();

        plint y0 = util::roundToInt(toLB(param.xyRange[0], 1, param.dx, ...
           param.physicalLocation));
        plint y1 = util::roundToInt(toLB(param.xyRange[1], 1, param.dx, ...
           param.physicalLocation));
        plint z0 = util::roundToInt(toLB(param.xzRange[0], 2, param.dx, ...
           param.physicalLocation));
        plint z1 = util::roundToInt(toLB(param.xzRange[1], 2, param.dx, ...
           param.physicalLocation));
        PLB_ASSERT(y1 >= y0 && z1 >= z0);

        for (size_t i = 0; i < param.xPositions.size(); i++) {
            plint xPos = util::roundToInt(toLB(param.xPositions[i], 0, ...
               param.dx, param.physicalLocation));
            plint x0 = xPos - 1;
            plint x1 = xPos + 1;
            param.xSlices.push_back(Box3D(x0, x1, y0, y1, z0, z1));

 emph# emphifdef PLB_DEBUG
            bool intersectionOK =
 emph# emphendif
                intersect(param.xSlices.back(), param.fullDomain, ...
                   param.xSlices.back());
            PLB_ASSERT(intersectionOK);
        }
    }

    {
        param.ySlices.clear();

        plint z0 = util::roundToInt(toLB(param.yzRange[0], 2, param.dx, ...
           param.physicalLocation));
        plint z1 = util::roundToInt(toLB(param.yzRange[1], 2, param.dx, ...
           param.physicalLocation));
        plint x0 = util::roundToInt(toLB(param.yxRange[0], 0, param.dx, ...
           param.physicalLocation));
        plint x1 = util::roundToInt(toLB(param.yxRange[1], 0, param.dx, ...
           param.physicalLocation));

        PLB_ASSERT(z1 >= z0 && x1 >= x0);

        for (size_t i = 0; i < param.yPositions.size(); i++) {
            plint yPos = util::roundToInt(toLB(param.yPositions[i], 1, ...
```

```
                param.dx, param.physicalLocation));
            plint y0 = yPos - 1;
            plint y1 = yPos + 1;
            param.ySlices.push_back(Box3D(x0, x1, y0, y1, z0, z1));

 emph# emphifdef PLB_DEBUG
            bool intersectionOK =
 emph# emphendif
                intersect(param.ySlices.back(), param.fullDomain, ...
                    param.ySlices.back());
            PLB_ASSERT(intersectionOK);
        }
    }

    {
        param.zSlices.clear();

        plint x0 = util::roundToInt(toLB(param.zxRange[0], 0, param.dx, ...
            param.physicalLocation));
        plint x1 = util::roundToInt(toLB(param.zxRange[1], 0, param.dx, ...
            param.physicalLocation));
        plint y0 = util::roundToInt(toLB(param.zyRange[0], 1, param.dx, ...
            param.physicalLocation));
        plint y1 = util::roundToInt(toLB(param.zyRange[1], 1, param.dx, ...
            param.physicalLocation));

        PLB_ASSERT(x1 >= x0 && y1 >= y0);

        for (size_t i = 0; i < param.zPositions.size(); i++) {
            plint zPos = util::roundToInt(toLB(param.zPositions[i], 2, ...
                param.dx, param.physicalLocation));
            plint z0 = zPos - 1;
            plint z1 = zPos + 1;
            param.zSlices.push_back(Box3D(x0, x1, y0, y1, z0, z1));

 emph# emphifdef PLB_DEBUG
            bool intersectionOK =
 emph# emphendif
                intersect(param.zSlices.back(), param.fullDomain, ...
                    param.zSlices.back());
            PLB_ASSERT(intersectionOK);
        }
    }
}

void orderAllOutputDomains()
{
    param.allOutputDomains.clear();
    param.allOutputDomainNames.clear();

    if (param.outputInDomain) {
        param.allOutputDomains.push_back(param.outputDomain);
        param.allOutputDomainNames.push_back("domain");
    }

    if (param.outputOnSlices) {
        size_t numXdigits = util::val2str(param.xSlices.size()).length();
        for (size_t i = 0; i < param.xSlices.size(); i++) {
            param.allOutputDomains.push_back(param.xSlices[i]);
            param.allOutputDomainNames.push_back(createFileName("slice_x_", ...
```

```
                i, numXdigits+1));
        }

        size_t numYdigits = util::val2str(param.ySlices.size()).length();
        for (size_t i = 0; i < param.ySlices.size(); i++) {
            param.allOutputDomains.push_back(param.ySlices[i]);
            param.allOutputDomainNames.push_back(createFileName("slice_y_", ...
                i, numYdigits+1));
        }

        size_t numZdigits = util::val2str(param.zSlices.size()).length();
        for (size_t i = 0; i < param.zSlices.size(); i++) {
            param.allOutputDomains.push_back(param.zSlices[i]);
            param.allOutputDomainNames.push_back(createFileName("slice_z_", ...
                i, numZdigits+1));
        }
    }
}

void setDerivedParameters()
{
    // Derived quantities.
    Cuboid<T> fullCuboid(Array<T,3>(param.xDomain[0], param.yDomain[0], ...
        param.zDomain[0]),
                         Array<T,3>(param.xDomain[1], param.yDomain[1], ...
                             param.zDomain[1]));

    param.dx = param.characteristicLength / (T) (param.resolution - 1);

    T lx = fullCuboid.x1() - fullCuboid.x0();
    T ly = fullCuboid.y1() - fullCuboid.y0();
    T lz = fullCuboid.z1() - fullCuboid.z0();
    param.physicalLocation = Array<T,3>(fullCuboid.x0(), fullCuboid.y0(), ...
        fullCuboid.z0());
    param.nx = util::roundToInt(lx / param.dx);
    param.ny = util::roundToInt(ly / param.dx) + 1;
    param.nz = util::roundToInt(lz / param.dx);
    param.fullDomain = Box3D(0, param.nx - 1, 0, param.ny - 1, 0, param.nz - 1);

    param.bottom = Box3D(0, param.nx - 1, 0,            0,            0, ...
        param.nz - 1);
    param.top    = Box3D(0, param.nx - 1, param.ny - 1, param.ny - 1, 0, ...
        param.nz - 1);

    param.initialFluidDomain = param.fullDomain;
    param.initialFluidDomain.y0++;
    param.fluidHeight_LB = util::roundToInt(param.fluidHeight / param.dx);
    param.initialFluidDomain.y1 = param.fluidHeight_LB - 1;

    param.saveDynamicContent = true;
    param.fileNamePadding = 8;

    param.dt = param.u_LB / param.u_Ref * param.dx;

    param.rho_LB = (T) 1;

    param.yVel_LB = param.yVel * (param.dt / param.dx);

    param.inletVelocity_LB = param.inletVelocity * (param.dt / param.dx);
```

```
    param.initialVelocity_LB = param.inletVelocity_LB;

    param.surfaceTension_LB = (param.rho_LB/param.rho) * param.dt*param.dt / ...
        (param.dx*param.dx*param.dx) * param.surfaceTension;
    param.gravity_LB = (T) 9.81 * (param.dt*param.dt / param.dx);

    param.numAbsorbingCells[0] = ...
       util::roundToInt(param.inletAbsorbingZoneWidth / param.dx);
    param.numAbsorbingCells[1] = ...
       util::roundToInt(param.outletAbsorbingZoneWidth / param.dx);
    param.numAbsorbingCells[2] = 0;
    param.numAbsorbingCells[3] = ...
       util::roundToInt(param.topAbsorbingZoneWidth / param.dx);
    param.numAbsorbingCells[4] = ...
       util::roundToInt(param.lateralAbsorbingZoneWidth / param.dx);
    param.numAbsorbingCells[5] = ...
       util::roundToInt(param.lateralAbsorbingZoneWidth / param.dx);

    param.totalNumAbsorbingCells = 0;
    for (int iZone = 0; iZone < 6; iZone++) {
        param.totalNumAbsorbingCells += param.numAbsorbingCells[iZone];
    }

    param.A_LB = param.A;
    param.P_LB = param.P / param.dt;
    param.waveDomain.x0 = util::roundToInt(toLB(param.xWaveDomain[0], 0, ...
       param.dx, param.physicalLocation));
    param.waveDomain.x1 = util::roundToInt(toLB(param.xWaveDomain[1], 0, ...
       param.dx, param.physicalLocation));
    param.waveDomain.y0 = util::roundToInt(toLB(param.yWaveDomain[0], 1, ...
       param.dx, param.physicalLocation));
    param.waveDomain.y1 = util::roundToInt(toLB(param.yWaveDomain[1], 1, ...
       param.dx, param.physicalLocation));
    param.waveDomain.z0 = util::roundToInt(toLB(param.zWaveDomain[0], 2, ...
       param.dx, param.physicalLocation));
    param.waveDomain.z1 = util::roundToInt(toLB(param.zWaveDomain[1], 2, ...
       param.dx, param.physicalLocation));
 emph# emphifdef PLB_DEBUG
    bool intersectionOK =
 emph# emphendif
        intersect(param.waveDomain, param.fullDomain, param.waveDomain);
    PLB_ASSERT(intersectionOK);

    T nu_LB = param.nu * param.dt / (param.dx * param.dx);
    param.omega = (T) 1 / (DESCRIPTOR<T>::invCs2 * nu_LB + (T) 0.5);

    computeOutputDomain(param.outputCuboid, param.outputDomain);
    computeOutputSlices();
    orderAllOutputDomains();
}

T waveForce(T t)
{
    static T pi = std::acos((T) -1);
    return param.A_LB * std::fabs(param.gravity_LB) * std::sin((T) 2 * pi * ...
       t / param.P_LB);
}

void setupBoatGeometry(bool operateOnOuterBorder, MultiScalarField3D<int>& tags)
{
```

```
TriangleSet<T>* triangleSet = new TriangleSet<T>(param.boatStl);
triangleSet->translate(-param.physicalLocation);
triangleSet->scale((T) 1 / param.dx);

RawConnectedTriangleMesh<T>* mesh = 0;
{
    RawTriangleMesh<T> rawMesh = triangleSetToRawTriangleMesh(*triangleSet);
    mesh = new ...
       RawConnectedTriangleMesh<T>(MeshConnector<T>(rawMesh).generateConnectedMesh());
}
inflate(*mesh, param.inflationParameter);
plint borderWidth = 1;
MultiNTensorField3D<int>* voxelMatrix = meshToVoxel(*mesh, ...
   tags.getBoundingBox(), borderWidth);
delete mesh; mesh = 0;

setToConstant(tags, voxelMatrix->scalarView(), voxelFlag::inside, ...
   tags.getBoundingBox(), (int) IN);
setToConstant(tags, voxelMatrix->scalarView(), voxelFlag::innerBorder, ...
   tags.getBoundingBox(), (int) IN);
if (operateOnOuterBorder) {
    setToConstant(tags, voxelMatrix->scalarView(), ...
       voxelFlag::outerBorder, tags.getBoundingBox(), (int) IN);
}
delete voxelMatrix; voxelMatrix = 0;

plint maxRefinements = 100;
T targetLength = (T) 1;
bool succeeded = triangleSet->refineRecursively(targetLength, ...
   maxRefinements);
if (!succeeded) {
    pcout << std::endl;
    pcout << "WARNING: The target maximum triangle edge length " << ...
       targetLength
          << " for the immersed surface was not reached after " << ...
             maxRefinements
          << " refinement iterations." << std::endl;
    pcout << std::endl;
    exit(1);
}

{
    RawTriangleMesh<T> rawMesh = triangleSetToRawTriangleMesh(*triangleSet);
    delete triangleSet; triangleSet = 0;
    param.connectedMesh = new ...
       RawConnectedTriangleMesh<T>(MeshConnector<T>(rawMesh).generateConnectedMesh());
}

plint numVertices = param.connectedMesh->getNumVertices();
param.vertices.resize(numVertices);
param.areas.resize(numVertices);
param.flags.resize(numVertices, (int) 1);
plint uniqueVertexIdTag = param.connectedMesh->getVertexTag("UniqueID");
RawConnectedTriangleMesh<T>::PVertexIterator ...
   vertexIterator(param.connectedMesh->vertexIterator());
while (!vertexIterator->end()) {
    RawConnectedTriangleMesh<T>::PVertex vertex = vertexIterator->next();
    plint iVertex = vertex->tag(uniqueVertexIdTag);
    param.vertices[iVertex] = vertex->get();
    param.areas[iVertex] = vertex->area();
```

```
    }
    pcout << "The number of vertices on the boat surface (after refinement) ...
        is: " << numVertices << std::endl;
}

void initialRhoU(plint iX, plint iY, plint iZ, T& rho, Array<T,3>& u)
{
    rho = param.rho_LB;
    if (iY <= param.fluidHeight_LB) {
        rho = param.rho_LB + DESCRIPTOR<T>::invCs2 * ...
            std::fabs(param.gravity_LB) * (param.fluidHeight_LB - iY);
    }
    u = Array<T,3>(param.initialVelocity_LB, (T) 0, (T) 0);
}

template <typename T>
class VelFunction {
public:
    T yVel_LB = param.yVel_LB;
    Array<T,3> operator()(pluint id)
    {
        return Array<T,3>(0.0, yVel_LB, 0.0);
    }
};

void printSimulationParameters()
{
    pcout << "Physical simulation domain: [" << param.xDomain[0] << ", " << ...
        param.xDomain[1] << "] x ["
                                              << param.yDomain[0] << ", " << ...
                                                 param.yDomain[1] << "] x ["
                                              << param.zDomain[0] << ", " << ...
                                                 param.zDomain[1] << "]" << ...
                                                 std::endl;
    pcout << "inletAbsorbingZoneWidth = " << param.inletAbsorbingZoneWidth ...
        << std::endl;
    pcout << "outletAbsorbingZoneWidth = " << param.outletAbsorbingZoneWidth ...
        << std::endl;
    pcout << "lateralAbsorbingZoneWidth = " << ...
        param.lateralAbsorbingZoneWidth << std::endl;
    pcout << "topAbsorbingZoneWidth = " << param.topAbsorbingZoneWidth << ...
        std::endl;
    pcout << "fluidHeight = " << param.fluidHeight << std::endl;

    pcout << "boatStl = " << param.boatStl << std::endl;

    pcout << "rho = " << param.rho << std::endl;
    pcout << "nu = " << param.nu << std::endl;
    pcout << "surfaceTension = " << param.surfaceTension << std::endl;

    pcout << "inflationParameter = " << param.inflationParameter << std::endl;
    pcout << "characteristicLength = " << param.characteristicLength << ...
        std::endl;
    pcout << "resolution = " << param.resolution << std::endl;
    pcout << "inletVelocity = " << param.inletVelocity << std::endl;
    pcout << "u_Ref = " << param.u_Ref << std::endl;
    pcout << "u_LB = " << param.u_LB << std::endl;
    pcout << "maxIter = " << param.maxIter << std::endl;
    pcout << "cSmago = " << param.cSmago << std::endl;
    pcout << "strongRepelling = " << (param.strongRepelling ? "true" : ...
```

```
        "false") << std::endl;
    pcout << "ambientPressure = " << param.ambientPressure << std::endl;

    pcout << "outDir = " << param.outDir << std::endl;
    pcout << "statIter = " << param.statIter << std::endl;
    pcout << "outIter = " << param.outIter << std::endl;
    pcout << "cpIter = " << param.cpIter << std::endl;
    pcout << "abIter = " << param.abIter << std::endl;
    pcout << "excludeInteriorForOutput = " << ...
        (param.excludeInteriorForOutput ? "true" : "false") << std::endl;
    pcout << "numPresLaplaceIter = " << param.numPresLaplaceIter << std::endl;
    pcout << "abortFileName = " << param.abortFileName << std::endl;
    pcout << "xmlContinueFileName = " << param.xmlContinueFileName << std::endl;
    pcout << "baseFileName = " << param.baseFileName << std::endl;
    pcout << "useParallelIO = " << (param.useParallelIO ? "true" : "false") ...
       << std::endl;

    pcout << "incompressibleModel = " << (param.incompressibleModel ? "true" ...
        : "false") << std::endl;
    pcout << "fluidHeight_LB = " << param.fluidHeight_LB << std::endl;
    pcout << "rho_LB = " << param.rho_LB << std::endl;
    pcout << "surfaceTension_LB = " << param.surfaceTension_LB << std::endl;
    pcout << "gravity_LB = " << param.gravity_LB << std::endl;
    pcout << "inletVelocity_LB = " << param.inletVelocity_LB << std::endl;
    pcout << "initialVelocity_LB = " << param.initialVelocity_LB << std::endl;
    for (int iZone = 0; iZone < 6; iZone++) {
        pcout << "numAbsorbingCells[" << iZone << "] = " << ...
            param.numAbsorbingCells[iZone] << std::endl;
    }
    T Re = param.u_Ref * param.characteristicLength / param.nu;
    pcout << "Reynolds number: Re = " << Re << std::endl;
    pcout << "omega = " << param.omega << std::endl;
    pcout << "tau = " << 1.0 / param.omega << std::endl;
    pcout << "dx = " << param.dx << std::endl;
    pcout << "dt = " << param.dt << std::endl;
    pcout << "dt / dx = " << param.dt / param.dx << std::endl;
    pcout << "dt / (dx * dx) = " << param.dt / (param.dx * param.dx) << ...
       std::endl;
    pcout << "physicalLocation = (" << param.physicalLocation[0] << ", " << ...
       param.physicalLocation[1] << ", "
          << param.physicalLocation[2] << ")" << std::endl;
    pcout << std::endl;
}

void applyAbsorbingZones(FreeSurfaceFields3D<T,DESCRIPTOR>& fields, ...
   MultiScalarField3D<T>& targetVolumeFraction,
        MultiTensorField3D<T,3>& targetVelocity)
{
    if (param.totalNumAbsorbingCells == 0) {
        return;
    }

    std::vector<MultiBlock3D*> args;
    args.push_back(&fields.volumeFraction);
    args.push_back(&targetVolumeFraction);
    args.push_back(&fields.rhoBar);
    args.push_back(&fields.mass);
    args.push_back(&fields.j);
    args.push_back(&targetVelocity);
    applyProcessingFunctional(new FreeSurfaceSpongeZone3D<T,DESCRIPTOR>(
```

```
                param.nx + 1, param.ny, param.nz + 1, param.numAbsorbingCells,
                param.incompressibleModel),
            fields.volumeFraction.getBoundingBox(), args);
}

void writeVTK(FreeSurfaceFields3D<T,DESCRIPTOR>& fields, ...
    MultiTensorField3D<T,3>& force, MultiScalarField3D<T>& smoothVF,
        MultiScalarField3D<T>& pressure, Box3D const& outputDomain, ...
           ParallelVtkImageOutput3D<T>& vtkOut)
{
    // CAUTION: If more entries are added to the VTK file in this function, ...
       the variable "numEntries" in
    //          the "writeResults" function must be adapted accordingly.

    vtkOut.writeData<float>(*extractSubDomain(pressure, outputDomain), ...
       "pressure", 1.0);
    vtkOut.writeData<float>(*copyConvert<int,T>(fields.flag, outputDomain), ...
       "flag", 1.0);
    MultiScalarField3D<T>* normalVF = ...
       extractSubDomain(fields.volumeFraction, outputDomain).release();
    boundScalarField<T>(*normalVF, (T) 0, (T) 1);
    vtkOut.writeData<float>(*normalVF, "volumeFraction", 1.0);
    delete normalVF; normalVF = 0;
    vtkOut.writeData<float>(*extractSubDomain(smoothVF, outputDomain), ...
       "volumeFractionFiltered", 1.0);
    std::auto_ptr<MultiTensorField3D<T,3> > v = ...
       freeSurfaceComputeForcedVelocity(fields.lattice, force, fields.flag, ...
       outputDomain);
    vtkOut.writeData<3,float>(*v, "velocity", param.dx / param.dt);
}

void writeResults(FreeSurfaceFields3D<T,DESCRIPTOR>& fields, ...
    MultiTensorField3D<T,3>& force,
        MultiScalarField3D<int>& tags, plint iIter)
{
    MultiScalarField3D<T>* smoothVF = ...
       generateMultiScalarField<T>(fields.volumeFraction, 1).release();
    copy(fields.volumeFraction, *smoothVF, ...
       fields.volumeFraction.getBoundingBox());
    lbmSmoothenInPlace<T,SM_DESCRIPTOR>(*smoothVF);
    lbmSmoothenInPlace<T,SM_DESCRIPTOR>(*smoothVF);
    boundScalarField<T>(*smoothVF, (T) 0, (T) 1);

    T pressureScale = param.rho * (param.dx * param.dx) / (param.dt * ...
       param.dt) * DESCRIPTOR<T>::cs2;
    T pressureOffset = param.ambientPressure -
        param.rho_LB * param.rho * (param.dx * param.dx) / (param.dt * ...
           param.dt) * DESCRIPTOR<T>::cs2;
    std::auto_ptr<MultiScalarField3D<T> > pressure = ...
       computeDensity(fields.lattice);
    pressure->periodicity().toggle(0, true);
    pressure->periodicity().toggle(1, false);
    pressure->periodicity().toggle(2, true);

    Box3D smoothDomain(param.fullDomain);
    smoothDomain.y0++;
    smoothDomain.y1--;
    for (int iSmooth = 0; iSmooth < param.numPresLaplaceIter; iSmooth++) {
        lbmSmoothenInPlace<T,SM_DESCRIPTOR>(*pressure, smoothDomain);
    }
```

```
multiplyInPlace(*pressure, pressureScale);
addInPlace(*pressure, pressureOffset);

// CAUTION: If more entries are added to the VTK file in the "writeVTK" ...
   function, the variable
//          "numEntries" must be adapted accordingly here.

plint numEntries = 5;

for (size_t iDomain = 0; iDomain < param.allOutputDomains.size(); ...
   iDomain++) {
    Box3D outputDomain = param.allOutputDomains[iDomain];
    std::string domainName = param.allOutputDomainNames[iDomain];
    std::string fname = createFileName(param.outDir + domainName + "_", ...
       iIter, param.fileNamePadding);
    ParallelVtkImageOutput3D<T> vtkOut(fname, numEntries, param.dx, ...
       param.physicalLocation);
    writeVTK(fields, force, *smoothVF, *pressure, outputDomain, vtkOut);
}

// Use a marching-cube algorithm to reconstruct the free surface and ...
   write an STL file.

std::vector<T> isoLevels;
isoLevels.push_back((T) 0.5);
typedef TriangleSet<T>::Triangle Triangle;
std::vector<Triangle> triangles;
Box3D marchingCubeDomain(param.fullDomain);
marchingCubeDomain.x0 += 2;
marchingCubeDomain.x1 -= 2;
marchingCubeDomain.y0 += 1;
marchingCubeDomain.y1 -= 1;
isoSurfaceMarchingCube(triangles, *smoothVF, isoLevels, marchingCubeDomain);
delete smoothVF; smoothVF = 0;
TriangleSet<T> set(triangles);
set.scale(param.dx);
set.translate(param.physicalLocation);
set.writeBinarySTL(createFileName(param.outDir + "interface_", iIter, ...
   param.fileNamePadding) + ".stl");

// Export the pressure on the surface of the boat.

RawConnectedTriangleMesh<T> mesh(*param.connectedMesh);
plint envelopeWidth = 1;
std::auto_ptr<MultiScalarField3D<int> > mask = ...
   generateMultiScalarField<int>((MultiBlock3D&) fields.flag, ...
   envelopeWidth);
setToConstant(*mask, mask->getBoundingBox(), (int) 0);
setToConstant(*mask, fields.flag, (int) freeSurfaceFlag::fluid, ...
   fields.flag.getBoundingBox(), (int) 1);
setToConstant(*mask, fields.flag, (int) freeSurfaceFlag::interface, ...
   fields.flag.getBoundingBox(), (int) 1);
if (param.excludeInteriorForOutput) {
    setToConstant(*mask, tags, (int) IN, tags.getBoundingBox(), (int) 0);
}
for (plint iLayer = 0; iLayer < 3; iLayer++) {
    applyProcessingFunctional(new ...
       MaskedNTensorNeumannInLayersFunctional3D<T>(0, 1, -1),
            pressure->getBoundingBox(), pressure->nTensorView(), ...
```

```
                    mask->nTensorView());
    }
    nTensorFieldToMesh(pressure->nTensorView(), mesh, "pressure");
    mesh.scale(param.dx);
    mesh.translate(param.physicalLocation);
    std::string fname = createFileName(param.outDir + "boat_pressure_", ...
        iIter, param.fileNamePadding) + ".vtk";
    writeVTK(mesh, fname);
}


int main(int argc, char *argv[])
{
    plbInit(&argc, &argv);

    std::cout.precision(10);

    // Command-line arguments

    if (argc != 2 && argc != 3) {
        pcout << "Usage: " << argv[0] << " xml-input-file-name ...
            [xml-continue-file-name]" << std::endl;
        exit(1);
    }

    std::string xmlInputFileName;
    xmlInputFileName = std::string(argv[1]);
    abortIfCannotOpenFileForReading(xmlInputFileName);

    std::string xmlRestartFileName;
    bool continueSimulation = false;
    if (argc == 3) {
        xmlRestartFileName = std::string(argv[2]);
        continueSimulation = true;
    }

    int nproc = global::mpi().getSize();

    global::timer("init").start();
    global::timer("totalTime").start();

    // Set the simulation parameters.

    setParameters(xmlInputFileName);
    global::IOpolicy().activateParallelIO(param.useParallelIO);

    setDerivedParameters();

    // Setup the geometry.

    pcout << "Total number of lattice cells: " << param.nx * param.ny * ...
        param.nz << std::endl;
    MultiScalarField3D<int> tags(param.nx, param.ny, param.nz, (int) OUT);
    tags.periodicity().toggle(0, true);
    tags.periodicity().toggle(1, false);
    tags.periodicity().toggle(2, true);

    pcout << "Setting up the boat geometry." << std::endl;
    bool operateOnOuterBorder = true;
    setupBoatGeometry(operateOnOuterBorder, tags);
```

```
emph# emphifdef PLB_DEBUG
    {
        VtkImageOutput3D<T> vtkOut(param.outDir + "tags", param.dx, ...
            param.physicalLocation);
        vtkOut.writeData<float>(*copyConvert<int,T>(tags), "tags", 1.0);
    }
emph# emphendif

    // Free-surface blocks.

emph# emphifdef WAVE_ABSORPTION
    Dynamics<T,DESCRIPTOR>* dynamics = new WaveAbsorptionDynamics<T,DESCRIPTOR>(
            new SmagorinskyDynamics<T,DESCRIPTOR>(
                new TruncatedTRTdynamics<T,DESCRIPTOR>(param.omega, 1.1), ...
                    param.omega, param.cSmago));
    param.incompressibleModel = false;
    pcout << "Dynamics: Wave Absorption Smagorinsky TRT." << std::endl;
emph# emphelse
    Dynamics<T,DESCRIPTOR>* dynamics = new SmagorinskyDynamics<T,DESCRIPTOR>(
            new TruncatedTRTdynamics<T,DESCRIPTOR>(param.omega, 1.1), ...
                param.omega, param.cSmago);
    param.incompressibleModel = false;
    pcout << "Dynamics: Smagorinsky TRT." << std::endl;
emph# emphendif

    T contactAngle = (T) -1;
    plint numIBIterations = 4;
    int repelInterface = 0;
    if (param.strongRepelling) {
        repelInterface = 1; // 3;
    }
    FreeSurfaceFields3D<T,DESCRIPTOR> ...
        fields(tags.getMultiBlockManagement().getSparseBlockStructure(),
            dynamics->clone(), param.rho_LB, param.surfaceTension_LB, ...
                contactAngle, Array<T,0>::zero(),
            numIBIterations, param.vertices, param.areas, param.flags, ...
                VelFunction<T>(),
            repelInterface);

    fields.periodicityToggle(0, true);
    fields.periodicityToggle(1, false);
    fields.periodicityToggle(2, true);

    MultiTensorField3D<T,3> force((MultiBlock3D&) tags);
    force.periodicity().toggle(0, true);
    force.periodicity().toggle(1, false);
    force.periodicity().toggle(2, true);
    Array<T,3> bodyForce_LB = Array<T,3>((T) 0, -param.gravity_LB, (T) 0);
    setToConstant<T,3>(force, force.getBoundingBox(), bodyForce_LB);
    if (!util::isZero(param.A_LB)) {
        bodyForce_LB = Array<T,3>((T) 0, -param.gravity_LB + waveForce((T) ...
            0), (T) 0);
        setToConstant<T,3>(force, param.waveDomain, bodyForce_LB);
    }

    // Initialization

    pcout << "Setting up initial condition." << std::endl;
```

```
    setToConstant(fields.flag, fields.flag.getBoundingBox(), (int) ...
        freeSurfaceFlag::empty);
    setToConstant(fields.flag, param.initialFluidDomain, (int) ...
        freeSurfaceFlag::fluid);
    setToConstant(fields.flag, tags, (int) IN, tags.getBoundingBox(), (int) ...
        freeSurfaceFlag::empty);
    setToConstant(fields.flag, param.top, (int) freeSurfaceFlag::wall);

    initializeAtEquilibrium(fields.lattice, force, ...
        fields.lattice.getBoundingBox(), initialRhoU);
    computeRhoBarJ(fields.lattice, fields.rhoBar, fields.j, ...
        fields.lattice.getBoundingBox());

    freeSurfaceAddForceToMomentum<T,DESCRIPTOR>(fields.lattice, ...
        fields.rhoBar, fields.j, fields.flag, force,
            fields.lattice.getBoundingBox());

    bool useConstRho = false;
    bool useZeroMomentum = false;
    bool initializeCell = false;
    fields.defaultInitialize(useConstRho, useZeroMomentum, initializeCell);

    Array<bool,3> reflectOnAxis;
    reflectOnAxis[0] = false;
    reflectOnAxis[1] = true;
    reflectOnAxis[2] = false;
    defineDynamics(fields.lattice, param.bottom, new ...
        SpecularReflection<T,DESCRIPTOR>(reflectOnAxis, param.rho_LB));
    setToConstant(fields.flag, param.bottom, (int) freeSurfaceFlag::slipWall);

emph# emphifdef WAVE_ABSORPTION
    if (param.totalNumAbsorbingCells != 0) {
        setGenericExternalScalar(fields.lattice, ...
            fields.lattice.getBoundingBox(), ...
            DESCRIPTOR<T>::ExternalField::sigmaBeginsAt,
                WaveAbsorptionSigmaFunction3D<T>(fields.lattice.getBoundingBox(), ...
                    param.numAbsorbingCells, param.omega));
emph# emphifdef PLB_DEBUG
        {
            VtkImageOutput3D<T> vtkOut(param.outDir + "sigma", param.dx, ...
                param.physicalLocation);
            vtkOut.writeData<float>(*computeExternalScalar(fields.lattice, ...
                DESCRIPTOR<T>::ExternalField::sigmaBeginsAt),
                    "sigma", 1.0);
        }
emph# emphendif
        setExternalScalar(fields.lattice, fields.lattice.getBoundingBox(), ...
            DESCRIPTOR<T>::ExternalField::rhoBarBeginsAt, (T) 0);
        setExternalVector(fields.lattice, fields.lattice.getBoundingBox(), ...
            DESCRIPTOR<T>::ExternalField::uBeginsAt,
                Array<T,3>(param.inletVelocity_LB, (T) 0, (T) 0));
    }
emph# emphendif

    plint targetEnvelopeWidth = 1;
    std::auto_ptr<MultiScalarField3D<T> > targetVolumeFraction = ...
        generateMultiScalarField<T>(
            (MultiBlock3D&) fields.volumeFraction, targetEnvelopeWidth);
    copy(fields.volumeFraction, *targetVolumeFraction, ...
        fields.volumeFraction.getBoundingBox());
```

```
std::auto_ptr<MultiTensorField3D<T,3> > targetVelocity = ...
    generateMultiTensorField<T,3>(
         (MultiBlock3D&) fields.j, targetEnvelopeWidth);
setToConstant<T,3>(*targetVelocity, targetVelocity->getBoundingBox(), ...
    Array<T,3>(param.initialVelocity_LB, (T) 0, (T) 0));

applyAbsorbingZones(fields, *targetVolumeFraction, *targetVelocity);

plint iniIter = 0;

printSimulationParameters();

// Checkpointing.

std::vector<MultiBlock3D*> checkpointBlocks;
checkpointBlocks.push_back(&fields.lattice);
checkpointBlocks.push_back(&fields.mass);
checkpointBlocks.push_back(&fields.flag);
checkpointBlocks.push_back(&fields.volumeFraction);
checkpointBlocks.push_back(&fields.outsideDensity);
checkpointBlocks.push_back(&fields.rhoBar);
checkpointBlocks.push_back(&fields.j);
checkpointBlocks.push_back(&tags);
checkpointBlocks.push_back(&force);
checkpointBlocks.push_back(targetVolumeFraction.get());
checkpointBlocks.push_back(targetVelocity.get());

if (continueSimulation) {
    pcout << "Reading state of the simulation from file: " << ...
        xmlRestartFileName << std::endl;
    loadState(checkpointBlocks, iniIter, param.saveDynamicContent, ...
        xmlRestartFileName);
    fields.lattice.resetTime(iniIter);
    pcout << std::endl;
}

// File preparation.

FILE* fpEnergy = 0;
if (global::mpi().isMainProcessor()) {
    std::string fileName = param.outDir + "average_energy.dat";
    fpEnergy = fopen(fileName.c_str(), continueSimulation ? "a" : "w");
    PLB_ASSERT(fpEnergy != 0);
}

FILE* fpForce = 0;
if (global::mpi().isMainProcessor()) {
    std::string fileName = param.outDir + "total_force_on_boat.dat";
    fpForce = fopen(fileName.c_str(), continueSimulation ? "a" : "w");
    PLB_ASSERT(fpForce != 0);
}

global::mpi().barrier();
global::timer("init").stop();

pcout << "The full initialization phase took " << ...
    global::timer("init").getTime() << " seconds on "
      << nproc << " processes." << std::endl;
```

```
// Starting iterations.

pcout << std::endl;
pcout << "Starting simulation." << std::endl;
pcout << std::endl;
bool stopExecution = false;
plint iIter = 0;
for (iIter = iniIter; iIter < param.maxIter && !stopExecution; iIter++) {
    if (iIter != iniIter && (iIter % param.statIter == 0 || iIter == ...
       param.maxIter - 1)) {
        global::timer("io").start();
        pcout << "==== At iteration " << iIter
              << ", t = " << iIter * param.dt << std::endl;
        T energy = freeSurfaceComputeAverageForcedEnergy(fields.lattice, ...
           force, fields.flag) *
            param.rho * (param.dx * param.dx) / (param.dt * param.dt);
        if (!util::isFiniteNumber(energy)) {
            if (global::mpi().isMainProcessor()) {
                fprintf(stdout, "The simulation is unstable. Aborting ...
                   ...\n");
            }
```

```
        global::mpi().barrier();
        exit(1);
    }
    pcout << "Average kinetic energy: " << energy << std::endl;

    bool isCompressible = !param.incompressibleModel;
    MultiNTensorField3D<T>* stress = ...
       pypal_computeStress(fields.lattice, ...
       fields.lattice.getBoundingBox(),
            param.rho_LB, isCompressible);

    plint envelopeWidth = 1;
    std::auto_ptr<MultiScalarField3D<int> > mask = ...
       generateMultiScalarField<int>((MultiBlock3D&) fields.flag, ...
       envelopeWidth);

    setToConstant(*mask, mask->getBoundingBox(), (int) 0);
    setToConstant(*mask, fields.flag, (int) freeSurfaceFlag::fluid, ...
       fields.flag.getBoundingBox(), (int) 1);
    setToConstant(*mask, fields.flag, (int) ...
       freeSurfaceFlag::interface, fields.flag.getBoundingBox(), ...
       (int) 1);
    if (param.excludeInteriorForOutput) {
        setToConstant(*mask, tags, (int) IN, tags.getBoundingBox(), ...
           (int) 0);
    }
    for (plint iLayer = 0; iLayer < 3; iLayer++) {
        applyProcessingFunctional(new ...
           MaskedNTensorNeumannInLayersFunctional3D<T>(0, 1, -1),
                stress->getBoundingBox(), *stress, mask->nTensorView());
    }

    Array<T,3> force_LB = surfaceForceIntegral(*stress, ...
       *param.connectedMesh);
    delete stress; stress = 0;

    T forceConversion = param.rho * util::sqr(util::sqr(param.dx)) / ...
       util::sqr(param.dt);

    pcout << "Total force on the boat: ";
    pcout << "(" << forceConversion * force_LB[0] << ", "
                 << forceConversion * force_LB[1] << ", "
                 << forceConversion * force_LB[2] << ")" << std::endl;

    if (global::mpi().isMainProcessor()) {
        fprintf(fpEnergy, "% .8e\t% .8e\n", (double) (iIter * ...
           param.dt), (double) energy);
        fflush(fpEnergy);
        fprintf(fpForce, "% .8e\t% .8e\t% .8e\t% .8e\n", (double) ...
           (iIter * param.dt),
                (double) (forceConversion * force_LB[0]), (double) ...
                   (forceConversion * force_LB[1]),
                (double) (forceConversion * force_LB[2]));
        fflush(fpForce);
    }
    global::timer("io").stop();

    if (iIter != iniIter) {
        pcout << "Time per iteration: " << ...
           global::timer("iterations").getTime() / (double) (iIter - ...
```

```
            iniIter)
              << std::endl;
        pcout << "Time per iteration considering the last " << ...
            param.statIter << " iterations: "
              << global::timer("statIterations").getTime() / ...
                  (double) param.statIter << std::endl;
        global::timer("statIterations").reset();
        pcout << "Total run time: " << ...
            global::timer("totalTime").getTime() << std::endl;
        pcout << "Total I/O time: " << global::timer("io").getTime() ...
            << std::endl;
    }
}

if (iIter % param.outIter == 0 || iIter == param.maxIter - 1) {
    global::timer("io").start();
    pcout << "== Output to disk at iteration: " << iIter << std::endl;
    writeResults(fields, force, tags, iIter);
    pcout << std::endl;
    global::timer("io").stop();
}

if (param.cpIter > 0 && iIter % param.cpIter == 0 && iIter != ...
    iniIter) {
    global::timer("io").start();
    pcout << "Saving the state of the simulation at iteration: " << ...
        iIter << std::endl;
    saveState(checkpointBlocks, iIter, param.saveDynamicContent, ...
        param.xmlContinueFileName,
            param.baseFileName, param.fileNamePadding);
    pcout << std::endl;
    global::timer("io").stop();
}

if (iIter % param.abIter == 0) {
    stopExecution = abortExecution(param.abortFileName, ...
        checkpointBlocks, iIter,
            param.saveDynamicContent, param.xmlContinueFileName,
            param.baseFileName, param.fileNamePadding);

    if (stopExecution) {
        pcout << "Aborting execution at iteration: " << iIter << ...
            std::endl;
        pcout << std::endl;
    }
}

global::timer("iterations").start();
global::timer("statIterations").start();
fields.lattice.executeInternalProcessors();
fields.lattice.evaluateStatistics();
fields.lattice.incrementTime();

if (!util::isZero(param.A_LB)) {
    bodyForce_LB = Array<T,3>((T) 0, -param.gravity_LB + ...
        waveForce((T) (iIter + 1)), (T) 0);
    setToConstant<T,3>(force, param.waveDomain, bodyForce_LB);
}
freeSurfaceAddForceToMomentum<T,DESCRIPTOR>(fields.lattice, ...
    fields.rhoBar, fields.j, fields.flag, force,
```

```
            fields.lattice.getBoundingBox());

    applyAbsorbingZones(fields, *targetVolumeFraction, *targetVelocity);

    global::timer("iterations").stop();
    global::timer("statIterations").stop();
}

// Solver execution statistics.

global::mpi().barrier();
global::timer("totalTime").stop();
global::timer("iterations").stop();
global::timer("io").stop();

double totT = global::timer("totalTime").getTime();
double iniT = global::timer("init").getTime();
double itT  = global::timer("iterations").getTime();
double ioT  = global::timer("io").getTime();

double iniPC = !util::isZero(totT) ? iniT / totT * 100.0 : 0.0;
double itPC  = !util::isZero(totT) ? itT  / totT * 100.0 : 0.0;
double ioPC  = !util::isZero(totT) ? ioT  / totT * 100.0 : 0.0;

pcout << "The solver finished executing successfully with " << nproc << ...
   " processes." << std::endl;
pcout << "The total running time of the solver, was " << totT << " ...
   seconds." << std::endl;
pcout << "The " << iIter - iniIter << " iterations of the solver, took " ...
   << itT << " seconds (" << itPC << "%)." << std::endl;
pcout << "The total I/O time, was " << ioT << " seconds (" << ioPC << ...
   "%)." << std::endl;

FILE *fpExecStat = 0;
if (global::mpi().isMainProcessor()) {
    std::string fileName = param.outDir + "solver_execution_statistics.txt";
    fpExecStat = fopen(fileName.c_str(), continueSimulation ? "a" : "w");
    PLB_ASSERT(fpExecStat != 0);
    if (continueSimulation) {
        fprintf(fpExecStat, "\nSummary (after restarting) of execution ...
           of the solver: %s\n\n", argv[0]);
    } else {
        fprintf(fpExecStat, "Summary of execution of the solver: ...
           %s\n\n", argv[0]);
    }
    fprintf(fpExecStat, "Number of processes: %d\n", nproc);
    fprintf(fpExecStat, "Total execution time ...
                                    : %g s\n", totT);
    fprintf(fpExecStat, "Total time of the initialization phase ...
                 : %g s,\t%g%% of the total time.\n", iniT, iniPC);
    fprintf(fpExecStat, "Total time of the pure solution phase (no ...
       output): %g s,\t%g%% of the total time.\n", itT, itPC);
    fprintf(fpExecStat, "Total time of I/O ...
                                       : %g s,\t%g%% of the total ...
       time.\n", ioT, ioPC);
    fclose(fpExecStat);
}

if (global::mpi().isMainProcessor()) {
    fclose(fpForce);
```

```
        fclose(fpEnergy);
    }

    // Create a finish file on completion (Jack).
    // std::string emptyFile = "finished";
    // fopen(emptyFile.c_str(), "w" );

    delete param.connectedMesh;
    delete dynamics;

    return 0;
}
```

Bibliography

[1] Pritchard James. From Shipwright to Naval Constructor : The Professionalization of 18th-Century French Naval Shipbuilders. Technology and Culture. 2012;28(1):1–25.

[2] Brown DK. The way of a ship in the midst of the sea: the life and work of William Froude. Periscope Publishing Ltd.; 2006.

[3] Froude W. On the rolling of ships. Nature. 1861;45(1172):559–560.

[4] Russell JS. The Wave-Line Principle of Ship-Construction. Transactions of the Institution of Naval Architects. 1860;12.

[5] Swan and Raven models used by William Froude on River Dart in 1867. Science Museum Group Collection - Objects. 2009. Available from: `https://collection.sciencemuseumgroup.org.uk/objects/co8040652/one-of-four-original-swan-and-raven-models-used-by-froude-study-models`

[6] Preston A. Strike Craft. Hong Kong: Bison Books Ltd; 1982.

[7] Coastal Motor Boat (CMB 4). Imperial War Museum. Available from: `https://www.iwm.org.uk/collections/item/object/30004029`.

[8] World War 1 at Sea - Ships of the Royal Navy, 1914-1919;. Available from: `https://www.naval-history.net/WW1NavyBritishShips-Dittmar6MBNos.htm`.

[9] Caponnetto M. Practical CFD simulations for planing hulls. International Conference on High Performance Marine Vehicles (HIPER' 01). 2001. Available from:

http://www.caponnetto-hueber.com/Papers/Practical{_}CFD{_}simulations{_}for{_}planing{_}hulls{_}HIPER{_}2001.pdf.

[10] Ahmed YM. Determining Ship Resistance Using Computational Fluid Dynamics (CFD). Journal of Transport System Engineering. 2015;2(1):20–25.

[11] Bertram V. Practical Ship Hydrodynamics. Elsevier/Butterworth-Heinemann; 2012.

[12] Ang JH, Goh C, Li Y. Key Challenges and Opportunities in Hull Form Design Optimisation for Marine and Offshore Applications. In: 21st International Conference on Automation and Computing (ICAC). Glasgow, UK; 2015. .

[13] Sharma R, Kim Tw, Storch RL, Hopman HJJ, Erikstad SO. Challenges in computer applications for ship and floating structure design and analysis. Computer-Aided Design. 2012;44(3):166–185.

[14] Palaniappan K, Jameson A. Bodies having minimum pressure drag in supersonic flow - Investigating nonlinear effects. Collection of Technical Papers - AIAA Applied Aerodynamics Conference. 2004;2:1224–1229.

[15] Ferreiro LD, Pollara A. Contested Waterlines: The Wave-Line Theory and Shipbuilding in the Nineteenth Century. Technology and Culture. 2016;57(2):414–444.

[16] Lewis EV. Principles of naval architecture: Volume II - resistance, propulsion, and vibration. vol. 2; 1988.

[17] Rosén A. Loads and Responses for Planing Craft in Waves; 2004. Available from: http://sh.diva-portal.org/smash/get/diva2:14900/FULLTEXT01.

[18] Gregory D, Beach T. Resistance Measurements of Typical Planing Boat Appendages; 1979. December.

[19] Mansoori M, Fernandes AC, Ghassemi H. Interceptor design for optimum trim control and minimum resistance of planing boats. Applied Ocean Research. 2017;69:100–115. Available from: http://dx.doi.org/10.1016/j.apor.2017.10.006.

[20] Larsson L, Raven HC. Ship Resistance and Flow - Principles of Naval Architecture. Alexandria, VA: SNAME; 2010.

[21] Begovic E, Bertorello C. Resistance assessment of warped hullform. Ocean Engineering. 2012;56:28–42. Available from: http://dx.doi.org/10.1016/j.oceaneng.2012.08.004.

[22] De Marco A, Mancini S, Miranda S, Scognamiglio R, Vitiello L. Experimental and numerical hydrodynamic analysis of a stepped planing hull. Applied Ocean Research. 2017;64:135–154. Available from: http://dx.doi.org/10.1016/j.apor.2017.02.004.

[23] Clement EP. Effects of Longitudinal Bottom Spray Strips on Planing Boat Resistance. Washington DC: Department of the Navy - David Taylor Model Basin; 1964.

[24] Masri J, Dala L, Huard B. A Review of the Analytical Methods used for Seaplanes Performance Prediction. Aircraft Engineering and Aerospace Technology. 2019;91(6):820–833. Available from: https://doi.org/10.1108/aeat-07-2018-0186.

[25] Sottorf W. Experiments with Planing Surfaces. NACA Technical Memorandum 739. 1934:1–37.

[26] Shoemaker JM. Tank tests of flat and vee-bottom planing surfaces. NACA Technical Note 509. 1934:1–52.

[27] Sambraus A. Planing-surface Tests at Large Froude Numbers - Airfoil Comparison. NACA Technical Memorandum 848. 1938:1–28.

[28] Savitsky D, Neidinger JW. Wetted area and center of pressure of planing surfaces at very low speed coefficients. Stevens Institute of Technology; 1954.

[29] Chambliss DB, Boyd GM. The planing characteristics of two V-shaped prismatic surfaces having angles of deadrise of 20° and 40°. NACA Technical Note 2876. 1953.

[30] Savitsky D, Others. Hydrodynamic design of planing hulls. Marine Technology and SNAME News. 1964;1(04):71–95.

[31] Savitsky D, Brown PW, Others. Procedures for hydrodynamic evaluation of planing hulls in smooth and rough water. Marine Technology. 1976;13(4):381–400.

[32] Blount DL, Fox DL. Small-Craft Power Prediction. Marine Technology. 1976;13(1):14–45.

[33] Savitsky D, DeLorme MF, Datla R. Inclusion of Whisker Spray Drag in Performance Prediction Method for High-Speed Planing Hulls. Marine Technology. 2007;44(1):35–56. Available from: http://www.ingentaconnect.com/content/sname/mt/2007/00000044/00000001/art00004.

[34] Brizzolara S, Serra F. Accuracy of CFD codes in the prediction of planing surfaces hydrodyamic characteristics. The 2nd International Conference on Marine Research and Transportation. 2007:147–158. Available from: http://www.icmrt07.unina.it/Proceedings/Papers/B/14.pdf.

[35] Stern F, Yang J, Wang Z, Sadat-hosseini H, Mousaviraad M, Bhushan S, et al. Computational Ship Hydrodynamics : Nowadays and Way Forward. 2012;(August):26–31.

[36] von Kerczek CH, MD SAIA. A New Generalized Cross-Flow Momentum Integral Method for Three-Dimensional Ship Boundary Layers. Annapolis: Defense Technical Information Center; 1982.

[37] von Kerczek CH, Christoph G, Stern F. Further Developments of the Momentum Integral Method for Ship Boundary Layers. Annapolis: Scientific Applications Inc.; 1984.

[38] Stern F. Effects of Waves on the Boundary Layer of a Surface-Piercing Body. Iowa City: Iowa Institute of Hydraulic Research, University of Iowa; 1985.

[39] Stern F, Yoo SY, Patel VC. Interactive and large-domain solutions of higher-order viscous-flow equations. AIAA Journal. 1988 sep;26(9):1052–1060. Available from: http://dx.doi.org/10.2514/3.10011.

[40] Tahara Y, Stern F, Rosen B. An interactive approach for calculating ship boundary layers and wakes for nonzero froude number. Journal of Computational Physics. 1992;98(1):33–53. Available from: http://www.sciencedirect.com/science/article/pii/002199919290171T.

[41] Tahara Y, Stern F. A large-domain approach for calculating ship boundary layers and wakes and wave fields for nonzero Froude number. Journal of Computational Physics. 1996;127(2):398–411.

[42] Paterson EG, Wilson RV, Stern F. General-Purpose Parallel Unsteady RANS Ship Hydrodynamics Code : Cfdship-Iowa. IIHR - Hydroscience and Engineering Department, University of Iowa; 2003. 432.

[43] Carrica PM, Wilson RV, Noack RW, Stern F. Ship motions using single-phase level set with dynamic overset grids. Computers and Fluids. 2007;36(9):1415–1433.

[44] CFDShip Iowa Homepage; 2017. Available from: http://www.iihr.uiowa.edu/shiphydro/.

[45] Hirt CW, Nichols BD. Volume of fluid (VOF) method for the dynamics of free boundaries. Journal of Computational Physics. 1981;39(1):201–225.

[46] Michael TJ. Development and Validation of Multi-process Cavitation Model. University of Iowa; 2013.

[47] Stern F, Wang Z, Yang J, Sadat-Hosseini H, Mousaviraad M, Bhushan S, et al. Recent progress in CFD for naval architecture and ocean

engineering. Journal of Hydrodynamics. 2015;27(1):1–23. Available from: http://dx.doi.org/10.1016/S1001-6058(15)60452-8.

[48] Wilson W, Quezon T, Trinh V, Sarles C, Li J, Tools EA. HPCMP CREATE-SH Integrated Hydrodynamic Design Environment;(December 2016):47–56.

[49] Wilson W, Gorski J, Quezon T, Trinh V. No Title. In: SNAME World Maritime Technology Conference (WMTC). Providence, Rhode Island, USA; 2015. .

[50] Bethesda W, Wilson W, Hendrix D, Noblesse F, Gorski J. Validation of Resistance Predictions Using Total Ship Drag (TSD). Bethesda: Naval Surface Warfare Center, Carderock Division; 2011.

[51] Hess JL, Smith AMO. Calculation of potential flow about arbitrary bodies. Progress in Aerospace Sciences. 1967;8(C):1–138.

[52] WYATT DC. Development and assessment of a nonlinear wave prediction methodology for surface vessels. Journal of ship research;44(2):96–107. Available from: http://cat.inist.fr/?aModele=afficheN{\&}cpsidt=1139101.

[53] Kring DC, Milewski WM, Fine NE. Validation of a NURBS-Based BEM for Multihull Ship Seakeeping. In: 25th Symposium on Naval Hydrodynamics. St John's; 2004. .

[54] Larsson L, Regnstron B, Broberg L, Li D, Janson C. Failures, Fantasies and Feats in the Theoretical/Numerical Prediction of Ship Performance. Washington D.C.: National Academies Press; 1998.

[55] Numeca - FINE Marine Homepage;. Available from: http://www.numeca.com/product/finemarine.

[56] Chen L, He G, Wang D, Zhang J. Computation of Wave-Making Resistance on High Speed Catamaran Using FINE / Marine. 2015:880653.

[57] Dudson E, Harries S. Hydrodynamic Fine-Tuning of a Pentamaran for High-Speed Sea Transportation Services. FAST 2005 the 8th International conference on Fast Sea Transportation. 2005;(June).

[58] STAR CCM+ Homepage;. Available from: https://mdx.plm.automation.siemens.com/star-ccm-plus.

[59] Donnelly DJ, Neu WL. Numerical Simulation of Flow About a Surface-Effect Ship. FAST 2011 11th International Conference on Fast Sea Transportation. 2011;1(September):57–64.

[60] Fonfach JMa, Sutulo S, Guedes Soares C. Numerical study of ship-to-ship interaction forces on the basis of various flow models. Second International Conference on Ship Manoeuvring in Shallow and Confined Water: Ship to Ship Interaction. 2011;(May):137–146.

[61] ANSYS Fluent Homepage;. Available from: https://www.ansys.com/products/fluids/ansys-fluent.

[62] Open FOAM Homepage;. Available from: http://www.openfoam.com/.

[63] Jasak H. Open FOAM : Open sourcce CFD in research and industry. International Journal of Naval Architecture and Ocean Engineering. 2009;1(2):89–94.

[64] Park S, Park SW, Rhee SH, Lee SB, Choi JE, Kang SH. Investigation on the wall function implementation for the prediction of ship resistance. International Journal of Naval Architecture and Ocean Engineering. 2013;5(1):33–46. Available from: http://dx.doi.org/10.2478/IJNAOE-2013-0116.

[65] McNamara G, Alder B. Analysis of the lattice Boltzmann treatment of hydrodynamics. Physica A: Statistical Mechanics and its Applications. 1993 mar;194(1-4):218–228. Available from: http://linkinghub.elsevier.com/retrieve/pii/0378437193903569.

[66] Xu H, Sagaut P. Optimal low-dispersion low-dissipation LBM schemes for computational aeroacoustics. Journal of Computational Physics. 2011;230(13):5353–5382.

[67] Augier A, Dubois F, Gouarin L, Graille B. Linear lattice Boltzmann schemes for acoustic: Parameter choices and isotropy properties. Computers and Mathematics with Applications. 2013;65(6):845–863. Available from: http://dx.doi.org/10.1016/j.camwa.2012.06.025.

[68] LBHydra Homepage;. Available from: http://lbhydra.umn.edu/LBHydra/Home.html.

[69] Davis MA, Walsh SDC, Saar MO. Statistically reconstructing continuous isotropic and anisotropic two-phase media while preserving macroscopic material properties. Physical Review E - Statistical, Nonlinear, and Soft Matter Physics. 2011;83(2):1–11.

[70] Chun B, Ladd AJC. Interpolated boundary condition for lattice Boltzmann simulations of flows in narrow gaps. Physical Review E - Statistical, Nonlinear, and Soft Matter Physics. 2007;75(6):1–12.

[71] Walsh SDC, Saar MO. Developing extensible lattice-Boltzmann simulationsfor general-purpose graphics-programming units. 2011.

[72] Palabos Home Page;. Available from: http://www.palabos.org/index.php.

[73] Tian M, Gu W, Pan J, Guo M. Performance analysis and optimization of PalaBos on petascale Sunway BlueLight MPP supercomputer. Procedia Engineering. 2013;61:241–245. Available from: http://dx.doi.org/10.1016/j.proeng.2013.08.010.

[74] Shan X, Yuan XF, Chen H. Kinetic theory representation of hydrodynamics: a way beyond the Navier–Stokes equation. Journal of Fluid Mechanics. 2006;550:413–441. Available from: http://journals.cambridge.org/article{_}S0022112005008153.

[75] Deardorff JW. A numerical study of three-dimensional turbulent channel flow at large Reynolds numbers. Journal of Fluid Mechanics. 1970;41(02):453–480. Available from: http://journals.cambridge.org/article{_}S0022112070000691.

[76] XFlow Homepage;. Available from: http://www.xflowcfd.com/.

[77] Coleman GN, Sandberg RD. A primer on direct numerical simulation of turbulence - methods, procedures and guidelines. 2010:1–21. Available from: http://eprints.soton.ac.uk/66182/.

[78] Germano M, Piomelli U, Moin P, Cabot WH. A Dynamic Subgrid-Scale Eddy Viscosity Model. Physics of Fluids A: Fluid Dynamics. 1991;3(7):1760.

[79] Villani C. A review of mathematical topics in collisional kinetic theory; 2002.

[80] Loeb LB. The Kinetic Theory of Gases. Courier Corporation; 2004.

[81] Frisch U, Hasslacher B, Pomeau Y. Lattice-gas automata for the Navier-Stokes equation; 1986.

[82] Thürey N, Rüde U, Körner C. Interactive Free Surface Fluids with the Lattice Boltzmann Method. Science. 2005;10. Available from: http://graphics.ethz.ch/{~}thuereyn/download/nthuerey{_}050607{_}tr1rtlbm.pdf.

[83] Krüger T, Kusumaatmaja H, Kuzmin A, Shardt O, Silva G, Viggen EM. The Lattice Boltzmann Method - Principles and Practice. Springer International Publishing; 2017.

[84] Frisch U, D'Humières D, Hasslacher B, Lallemand P, Pomeau Y, Rivet JP. Lattice Gas Hydrodynamics in Two and Three Dimensions. Complex Systems. 1987;1(4):649–707.

[85] He X, Luo LS. A priori derivation of the lattice boltzmann equation. Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics. 1997;55(6).

[86] Krüger T, Kusumaatmaja H, Kuzmin A, Shardt O, Silva G, Viggen EM. The Lattice Boltzmann Method: Principles and Practice. 1st ed. Springer; 2017.

[87] Bhatnagar PL, Gross EP, Krook M. A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems. Physical Review. 1954;94(3):511.

[88] Latt J. Choice of units in lattice Boltzmann simulations. Freely available online. 2008;(April):1–6. Available from: https://www.semanticscholar.org/paper/Choice-of-units-in-lattice-Boltzmann-simulations-Latt/69588e58522de7e51e1a546590b462e026746b07.

[89] Latt J, Malaspinas O, Kontaxakis D, Parmigiani A, Lagrava D, Brogi F, et al. Palabos: Parallel Lattice Boltzmann Solver. Computers and Mathematics with Applications. 2020. Available from: https://doi.org/10.1016/j.camwa.2020.03.022.

[90] Latt J, Chopard B. Lattice Boltzmann method with regularized pre-collision distribution functions. Mathematics and Computers in Simulation. 2006;72(2-6):165–168.

[91] Beny J, Latt J. Efficient LBM on GPUs for dense moving objects using immersed boundary condition. 2019;(1):1–14. Available from: http://arxiv.org/abs/1904.02108.

[92] Geller S, Krafczyk M, Tölke J, Turek S, Hron J. Benchmark computations based on lattice-Boltzmann, finite element and finite volume methods for laminar flows. Computers and Fluids. 2006;35(8-9):888–897.

[93] Al-Jahmany YY, Brenner G, Brunn PO. Comparative study of lattice-Boltzmann and finite volume methods for the simulation of laminar flow through a 4:1 planar contraction. International Journal for Numerical Methods in Fluids. 2004;46(9):903–920.

[94] Körner C, Thies M, Hofmann T, Thürey N, Rüde U. Lattice Boltzmann model for free surface flow for modeling foaming. Journal of Statistical Physics. 2005;121(1-2):179–196.

[95] De Rosis A, Ubertini S, Ubertini F. A Comparison Between the Interpolated Bounce-Back Scheme and the Immersed Boundary Method to Treat Solid Boundary Conditions for Laminar Flows in the Lattice Boltzmann Framework. Journal of Scientific Computing. 2014;61(3):477–489.

[96] SUZUKI K, OKADA I, YOSHINO M. Accuracy of the laminar boundary layer on a flat plate in an immersed boundary-lattice Boltzmann simulation. Journal of Fluid Science and Technology. 2016;11(3):JFST0017–JFST0017.

[97] Supercomputing Wales Portal; 2020. Available from: `portal.supercomputing.wales/`.

[98] Palabos Webpage;. Available from: `https://palabos.unige.ch/`.

[99] Schelkens AM. Numeca and FlowKit Announce a Strategic Partnership;. Available from: `https://www.numeca.com/readnews/article/314`.

[100] Palabos User Guide. Release 1. ed. Geneva: University of Geneva; 2019.

[101] Gallivan MA, Noble DR, Georgiadis JG, Buckius RO. An evaluation of the bounce-back boundary condition for lattice Boltzmann simulations. International Journal for Numerical Methods in Fluids. 1997;25(3):249–263.

[102] Kaneda M, Haruna T, Suga K. Ghost-fluid-based boundary treatment in lattice Boltzmann method and its extension to advancing boundary. Applied Thermal Engineering. 2014;72(1):126–134. Available from: `http://dx.doi.org/10.1016/j.applthermaleng.2013.12.024`.

[103] Zhu L, He G, Wang S, Miller L, Zhang X, You Q, et al. An immersed boundary method based on the lattice Boltzmann approach in three dimensions, with application. Computers and Mathematics with Applications. 2011;61(12):3506–3518. Available from: http://dx.doi.org/10.1016/j.camwa.2010.03.022.

[104] Roache PJ. Perspective: A Method for Uniform Reporting of Grid Refinement Studies. Journal of Fluid Engineering. 1994;116:405–413.

[105] Roache PJ. Quantification of Uncertainty in Computational Fluid Dynamics. Annual Review of Fluid Mechanics. 1997;29(1):123–160.

[106] Roache PJ. Verification and Validation in Computational Science and Engineering. Computing in Science Engineering. 1998:107–240. Available from: http://www.hermosa-pub.com/hermosa{\%}0Ahttp://scholar.google.com/scholar?hl=en{\&}btnG=Search{\&}q=intitle:Verification+and+validation+in+computational+science+and+engineering{\#}0.

[107] Celik IB, Ghia U, Roache PJ, Freitas CJ, Coleman H, Raad PE, et al. Procedure for estimation and reporting of uncertainty due to discretization in ${$CFD$}$ applications. 2008.

[108] Roache PJ, Celik IB, Ghia U, Roache PJ, Freitas CJ, Coleman HP, et al. Procedure for estimation and reporting of uncertainty due to discretization in CFD applications. Computing in Science Engineering. 2008;116(1):123–160. Available from: http://www.hermosa-pub.com/hermosa{\%}0Ahttp://scholar.google.com/scholar?hl=en{\&}btnG=Search{\&}q=intitle:Verification+and+validation+in+computational+science+and+engineering{\#}0.

[109] Xing T, Stern F. Factors of Safety for Richardson Extrapolation. Iowa City: IIHR - Hydroscience & Engineering, University of Iowa; 2009.

[110] Tóth P, Lohász MM. Anisotropic Grid Refinement study for LES. ERCOFTAC Series. 2008;12(January 2008):167–178.

[111] John D, Sc CB. Hydrodynamics of Surfboard Fins [Doctoral Thesis]. Swansea University; 2007.

[112] Storebro Homepage;. Available from: http://storebro.se/index.php?page=storebro-90-e.

[113] Uniform procedure for the calculation of frictional resistance and the expansion of model test data to full size. Bulletin No 1-2 of SNAME; 1948. .

[114] Savitsky D, Ross EW. Turbulence Stimulation in the Boundary Layer of Planing Surfaces-Part II-Preliminary Experimental Investigation. Stevens Institute of Technology Experimental Towing Tank Report No. 1952;44.

[115] Privinvest - DV15; 2020. Available from: https://www.privinvest.com/naval-vessels/interceptor-dv15-rws/.

[116] Butcher JC, Goodwin N. Numerical methods for ordinary differential equations. vol. 2. Wiley Online Library; 2008.

[117] Bastiani R. Maritimo Web Page;. Available from: https://www.maritimo.com.au/maritimo-racing-debut-new-r30-2019-uim-xcat-world-championship/.

[118] Advanced Aerodynamic Vessels (A2V) Project Webpage; 2017. Available from: https://www.aavessels.com/concept/.

[119] Naumann DS, Evans B, Walton S, Hassan O. A novel implementation of computational aerodynamic shape optimisation using Modified Cuckoo Search. Applied Mathematical Modelling. 2016;40(7-8):4543–4559.

[120] Naumann DS, Evans B, Walton S, Hassan O. Discrete boundary smoothing using control node parameterisation for aerodynamic shape optimisation. Applied Mathematical Modelling. 2017;48:113–133. Available from: http://dx.doi.org/10.1016/j.apm.2017.03.042.

[121] McKay MD, Beckman RJ, Conover WJ. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. Technometrics, American Statistical Association. 1979;21(2):239–245.

[122] Virtual Wind Tunnel webpage;. Available from: https://altairhyperworks.com/product/virtual-wind-tunnel.

[123] Delauney B. Sur la sphère vide. Otdelenie Matematicheskii i Estestvennyka Nauk. 1934;7:793–800.

[124] Walton S, Hassan O, Morgan K, Brown MR. Modified cuckoo search: A new gradient free optimisation algorithm. Chaos, Solitons and Fractals. 2011;44(9):710–718. Available from: http://dx.doi.org/10.1016/j.chaos.2011.06.004.

[125] Viswanathan GM, Raposo EP, da Luz MGE. Lévy flights and superdiffusion in the context of biological encounters and random searches. Physics of Life Reviews. 2008;5(3):133–150. Available from: http://dx.doi.org/10.1016/j.plrev.2008.03.002.

[126] Sørensen KA. A multigrid accelerated procedure for the solution of compressible fluid flows on unstructured hybrid meshes [Doctoral Thesis]. University of Wales, Swansea; 2001.

[127] Allmaras PRSSR. A one-equation turbulence model for aerodynamic flows. La Recherche Aerospatiale. 1994;1:5–21.

[128] Latt J, Malaspinas O, Kontaxakis D, Parmigiani A, Lagrava D, Brogi F, et al. Palabos: Parallel Lattice Boltzmann Solver. Computers and Mathematics with Applications. 2021;81:334–350. Available from: https://doi.org/10.1016/j.camwa.2020.03.022.

[129] Amazon Web Services marketplace: CFD Direct from the Cloud; 2021. Available from: https://aws.amazon.com/marketplace/pp/CFD-Direct-CFD-Direct-From-the-Cloud/B017AHYO16.