NUMERICAL SIMULATION OF A CONTINUOUS CASTER

by

Matthew T. Moore

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science in Mechanical Engineering

Department of Mechanical and Civil Engineering

Hammond, Indiana

December 2019

THE PURDUE UNIVERSITY GRADUATE SCHOOL

STATEMENT OF COMMITTEE APPROVAL

Dr. Chenn Q. Zhou, Chair
Department of Mechanical and Civil Engineering

Dr. Harvey Abramowitz
Department of Mechanical and Civil Engineering

Dr. Ran Zhou
Department of Mechanical and Civil Engineering

Approved by:
Dr. Chenn Q. Zhou

I dedicate this thesis in memory of my late grandfather,

May his kindness and passion for learning live on in those he inspired;

To my parents,

I will always be grateful for their seemingly endless patience and support;

And to the rest of my family and friends who've supported me on this journey,

Here's hoping that I won't be buried in my work the next time we meet.

Thank you.

ACKNOWLEDGMENTS

I would like to acknowledge the Steel Manufacturing Simulation and Visualization Consortium (SMSVC) members for supporting the research discussed in this study, and more specifically, the primary cooling project technical committee for providing the geometry, operating conditions, validation data, and technical expertise for this study.

I would also like to acknowledge the Center for Innovation through Visualization and Simulation (CIVS) for providing students with tools and resources to nurture their curiosity for the unknown, and to broaden their appetite for knowledge. Special thanks to Dr. Armin Silaen, (the soon-to-be Dr.) Haibo Ma, and the rest of the CIVS family for their support and guidance.

Finally, I would like to thank my thesis advisory committee members: Dr. Chenn Zhou, Dr. Harvey Abramowitz, and Dr. Ran Zhou for generously offering their time, support, guidance, and goodwill throughout the preparation and review of this document.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

3D	three-dimensional
AMR	adaptive mesh refinement
AMR-CT	adaptive mesh refinement criteria tests
BC	boundary conditions
BF	broad face
CA	control algorithm
CAR	cell aspect ratio
CC	continuous caster/continuous casting
CFD	computational fluid dynamics
DOF	degree-of-freedom
HF	heat flux
HFP	heat flux profile
HPC	high performance computing
HTC	heat transfer coefficient
HTR	heat transfer rate
IC	industrial collaborator
ipm	inches-per-minute
k-ε	k-epsilon
k-ω	k-omega
LTF	liquid test front
mps	meters-per-second
MZ	mushy zone
NF	narrow face
PC	primary cooling
POC	proof-of-concept
RANS	Reynolds-averaged Navier-Stokes
RFC	ratio of failing cells
RP	refinement profile
SC	secondary cooling

SEN	submerged entry nozzle
S-P	Savage-Pritchard
SST	shear-stress transport
STF	solid test front
UDF	user-defined field function
UTN	upper tundish nozzle
VOF	volume of fluid

NOMENCLATURE

a	Savage-Pritchard peak heat flux coefficient
A	cross-sectional area
$\mathbf{A}_f$	surface area vector
A_{inlet}	cross-sectional area of domain inlet
A_{outlet}	cross-sectional area of domain outlet
b	Savage-Pritchard heat flux depreciation coefficient
c	dendrite shape constant
c_s	dendrite shape factor
$d\mathbf{x}$	position vector from the face centroid to the centroid of the cell
E	total energy
$\mathbf{f}_b$	vector of body forces
$\mathbf{f}_P$	porous resistance force vector
F_K	mushy zone permeability switching function
$\mathbf{g}$	gravity vector
h_{ls}	sensible heat
h_{ls}^*	corrected enthalpy value
h_{fusion}	latent heat of fusion
H	total enthalpy
$\mathbf{I}$	identity matrix/unity tensor
k	turbulent kinetic energy
k_0	ambient value to counteract turbulence decay
K	cell permeability
m_{Dims}	number of dimensions the mesh exists within
n_{Faces}	number of faces composing the cell
p	pressure
q''	mold heat flux profile
$\mathbf{q}''$	heat flux vector
q''_{BF}	heat flux profile defined for the broad face boundary conditions

q''_{NF}	heat flux profile defined for the narrow face boundary condition
$\bar{q}''_{Msrd}$	average surface heat flux determined from measurement data
$\dot{q}_{Actual}$	heat transfer rate indicated from the mold surface measurement data
$\dot{q}_{corner}$	scaled heat transfer rate in the corner regions of the mold surface
$\dot{q}_{std}$	heat transfer rate from the standard heat flux profile
$\dot{q}_{Tot}$	total heat transfer rate through a mold surface
s	mushy zone permeability switching-function constant
t	time
T	temperature
T^*	normalized temperature
T_{liq}	liquidus temperature
T_{sol}	solidus temperature
u	velocity
$\mathbf{u}$	velocity vector
$\bar{\mathbf{u}}$	mean velocity vector
u_{cs}	casting speed
$\mathbf{u}_{d,i}$	diffusion velocity vector of phase *i*
u_{inlet}	inlet velocity
$\mathbf{u}_{mix}$	mixture velocity vector
V	volume
$\dot{V}_{steel}$	volumetric flow rate of steel
w	horizontal offset from the center of the mold surface
ΔW	working width of the mold surface
ΔW_{corner}	mold corner-offset distance for decreased heat flux
ΔW_{std}	width of the mold surface between the corner offsets
y	depth below the meniscus
ΔY	working height of the mold surface
α_i	volume fraction of phase *i*
α_s	volume fraction of solid phase
$\alpha_{s,cr}$	critical solid fraction for the Mushy Zone Permeability switching function

β	turbulence model coefficient
β^*	turbulence model coefficient
δ	power-regression predicted shell thickness profile
η	scaling factor for the heat flux profile in the mold corners
$\mathbf{\Theta}_{MZ}$	mushy zone porous resistance tensor
λ_1	primary dendrite arm-spacing
μ	dynamic viscosity of steel
μ_t	turbulent eddy viscosity
ξ_k	turbulent kinetic energy production term
ξ_ω	specific dissipation rate production term
ρ	density of steel
σ_k	turbulence model coefficient
σ_ω	turbulence model coefficient
τ	turbulence time scale
ϕ_β	vortex-stretching modification factor
ϕ_{β^*}	free-shear modification factor
$\mathbf{\Psi}$	stress tensor
ω	specific turbulence dissipation rate
ω_0	ambient value to counteract turbulence decay

ABSTRACT

Heat transfer and solidification models were developed for use in a numerical model of a continuous caster to provide a means of predicting how the developing shell would react under variable operating conditions. Measurement data of the operating conditions leading up to a breakout occurrence were provided by an industrial collaborator and were used to define the model boundary conditions. Steady-state and transient simulations were conducted, using boundary conditions defined from time-averaged measurement data. The predicted shell profiles demonstrated good agreement with thickness measurements of a breakout shell segment – recovered from the quarter-width location. Further examination of the results with measurement data suggests pseudo-steady assumption may be inadequate for modeling shell and flow field transition period following sudden changes in casting speed. An adaptive mesh refinement procedure was established to increase refinement in areas of predicted shell growth and to remove excess refinement from regions containing only liquid. A control algorithm was developed and employed to automate the refinement procedure in a proof-of-concept simulation. The use of adaptive mesh refinement was found to decrease the total simulation time by approximately 11% from the control simulation – using a static mesh.

1. INTRODUCTION

Continuous casting (CC) is a critical part of the modern steel industry – accounting for roughly 95% of the world's steel production [1]. CC is the process by which molten metals are cast in an open-bottomed mold, so as to produce a continuous strand of semi-finished products; such as slabs, billets, and blooms. The general process and key features are depicted in Figure 1.1 for reference.

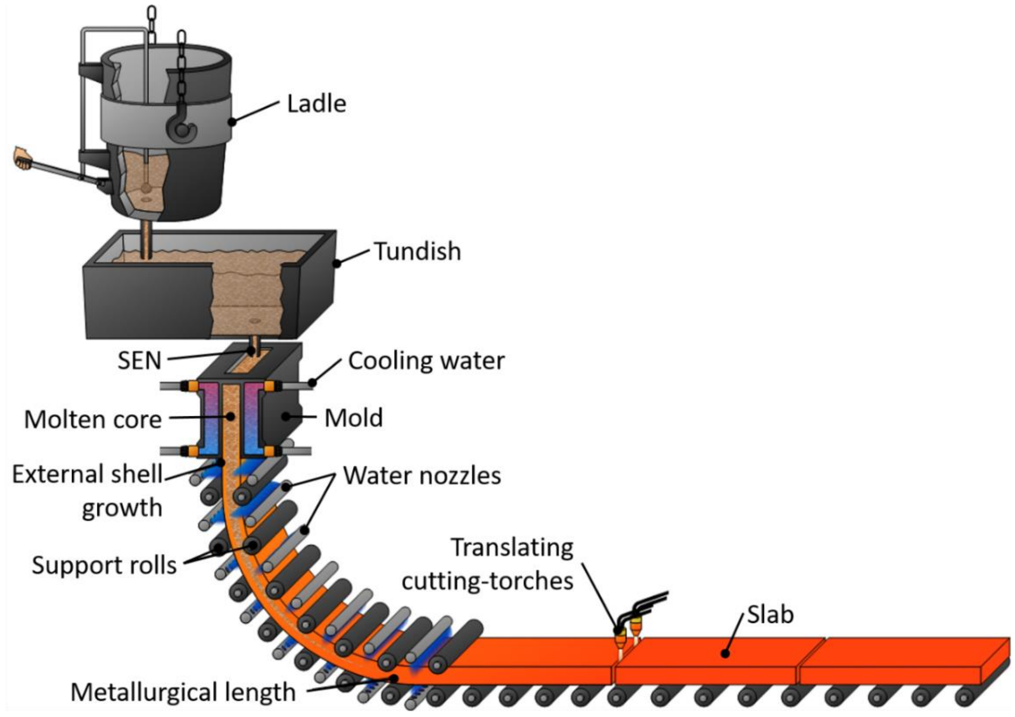


Figure 1.1. Diagram overview, and key features, of the CC process [2].

1.1 An Overview of Continuous Casting

As with other casting processes, CC begins after a supply of metal has been heated to a liquid state. The molten metal will then be transferred to a large container lined with an insulating refractory material, known as a *ladle*. The ladle provides a means of storing the material in a molten state during transport. Once the ladle has been transported to an operating caster – either by overhead crane or by railcar – it will then be moved into position to allow for its contents to drain into an intermediate reservoir, known as a tundish; which is located directly above one or more casters.

The tundish – which is also refractory-lined – can vary in shape, size, and complexity, depending on the casting system it is responsible for feeding, as well as the filtration and flow-control systems it implements. The primary function of a tundish is to provide a continuous and consistent flow of molten metal to the caster during operation. The liquid metal flows through the tundish and is drained through a relatively small opening located at the base of the tundish, called the *upper-tundish nozzle* (UTN). After a ladle has drained all of its contents into the tundish, a ladle exchange can be performed so that a new batch can continue the supply of the molten metal to the tundish. The interchange of ladles at the tundish, along with a flow regulating device at the UTN, allows for a continuous and consistent flow of the liquid metal into the mold.

As the liquid steel moves past the UTN, it continues through a shrouded channel-way that is formed of packed refractory, called the *submerged entry nozzle* (SEN). The SEN is commonly cylindrical in shape, and extends vertically downwards from the base of the tundish into the open-top of the mold. A set of port-openings at the base of the SEN eject the molten steel into the copper mold.

In continuous *slab* casters, the mold takes the form of a rectangular volume that extends vertically downwards, and is open at the top and bottom. The surfaces of the mold are referred to as the *narrow face* (NF) and *broad face* (BF). These copper surfaces of the mold are water-cooled, and rapidly extract heat from the melt, causing an encasing shell to solidify along the perimeter of the mold. One commonality of modern casters would be the ability to alter the positioning of the mold surfaces in a precise manner. This provides a means for controlling the dimensions of the encasing shell exiting the mold, as well as final cross-sectional dimensions of the fully solidified slab.

Slab casters frequently employ an SEN having two main side ports that redirect the flow horizontally-outwards, moving parallel with the BF surfaces. The flow streams exiting these ports act as liquid jets, which regularly impinge on the developing shell along the NF and causing it to re-melt. Due to the relatively shallow thickness of slab casters, the majority of the liquid impinging on the NF will be redirected to flow either down along the newly forming shell, or up towards the top of the mold, as shown in Figure 1.2. The stream flowing upwards will quickly change directions to flow back towards the SEN along the top surface of the liquid steel, referred to as the meniscus. As the flow stream approaches the SEN it will then change directions again, this time moving downwards, until it returns to the jet exiting the port, where it will then begin the cycle anew. As this flow pattern progressively builds up speed it will generate an area of relatively-low pressure at its center, causing it to retract inwards until it resembles a circle, known as a *roll pattern*. In somewhat of a similar manner, impingement flow that is instead directed downwards will continue on until buoyancy overcomes its momentum and causes the stream to circulate back upwards, at the center of the slab, where it will then meet back up with the jet. This *double-roll* pattern promotes a thorough distribution of the heat-energy within the mold, and mitigates the potential for stagnant conditions at the meniscus [3].

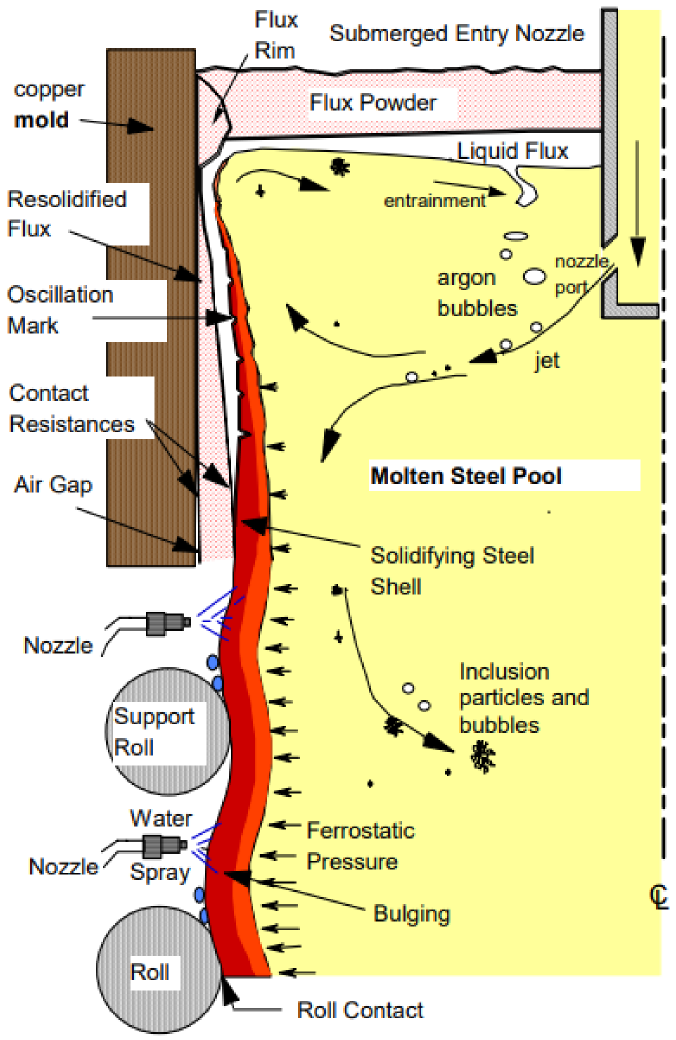


Figure 1.2. Schematic of physical phenomena in the mold region of a steel slab caster [4].

As the mold represents the initial stage in the cooling process of the caster, it is referred to as the *primary cooling* (PC) section of CC. After the slab exits through the base of the mold the shell is still in a malleable state, and would naturally deform under the ferrostatic pressure of the molten core, if left alone. To preserve the rectangular profile of the shell, a series of hydraulic rollers compress the shell back towards its original shape from all four sides. The solidification process progresses as the heat extraction is continued beyond the mold by means of convective exchange, via spray-cooling, in the *secondary cooling* (SC) section of the caster. The spray nozzles are interspersed between each pair of rollers, and provide a more gradual rate of heat extraction through the shell so that the remaining liquid core of the strand can solidify at an optimal rate.

In curved-type CC, the rolls compressing the strand through the SC section will gradually redirect the strand from a vertical to a horizontal orientation. For general curved-slab casters, this transition should complete prior to the metallurgical length of the strand – the point at which the core fully solidifies. After the strand has passed through the curved section of the SC region, it must then go through an unbending process, in which a series of the closely-spaced rollers, along the top and bottom-surfaces, are used to forcefully straighten the strand. Finally, while the strand is still relatively hot, a cutting-torch system will translate along with the strand as it separates the end segment of the strand with a vertical cut across its width. The detached section of the strand is referred to as a *slab*, and can then be carried away for storage or further treatment.

1.2 Motivations and Objectives

As technology advances, the design and operation of CC's are continuously being modernized so to optimize each process involved in order to increase product yield while reducing operational costs. However, the current industry still experiences significant losses due to product defects originating from the PC and SC sections of the CC. While some of these defects can be attributed to factors occurring between the process equipment and the external surface of the shell, a large percentage are believe to originate along the solidification front.

A better understanding of the complex phenomena occurring along the solidification front in a CC process is a fundamental necessity for reliably evaluating whether specific casting conditions inherently influence defect generation rates. A representation of various phenomena known to contribute to these defects are indicated in Figure 1.2. However, analyzing these parameters presents a difficult challenge. Due to the intrinsic nature of the casting process, engineers are restricted in their ability to obtain accurate measurements of the shell development during casting operations. Plant experiments are impractical, limited in high temperature environments, and expensive as they will negatively influence casting procedures. Additionally, full-scale physical models, while not as costly as plant trials, can still be excessive and do not always effectively integrate all parameters and design considerations.

With the advent of *computational fluid dynamics* (CFD), the feasibility of conducting large-scale numerical simulations for studying complex systems has significantly improved, as increasingly-advanced computer technology became readily available. CFD presents a practical method of analyzing various casting phenomena with a relatively-low cost. Since the earliest attempts to computationally simulate a CC, CFD models of the molten steel flow field, heat transfer between the shell and the mold, as well as solidification and shell development models have aided in attempts to better-understand how different operating conditions will influence the final quality of the product. However, it was necessary for a large portion of these models to be simplified due to the relatively high computational costs involved with the simulations, and the limited computation resources available.

1.2.1 Heat Transfer and Solidification Model

In CC, the solidification process involves a variety of different physical phenomena that must be regulated in order to manage the overall quality of the product. Two of the more prominent factors leading to decreased product quality would include particle inclusions – largely entailing slag, flux powder or argon gas – along the solidification front, and surface defects; such as oscillation marks and various surface cracks [5]–[7]. Although advances in technology have aided in reducing the impact of these factors during production, they remain a constant threat in CC. Aside from altering the material composition of the product or improving upon equipment or operating practices upstream of the caster, regulation of the heat transfer, as well as the introduction of impurities during the solidification process are the two principal contributors instigating these defects.

The primary cooling section of a continuous caster exists for the purpose of initiating solidification, and to generate an external shell casing that is structurally capable of retaining the molten core, while heat is continuously extracted at a more appropriate rate for managing the effects of thermal stresses. Due to the high temperature of the molten steel, it is necessary for the initial shell growth to develop rapidly in order to extend the service life of the mold by reducing the thermal loading conditions it experiences. This requires a relatively higher rate of heat extraction to be utilized during the initial shell development, which progressively decreases along the length of the mold; and further still, as the shell becomes structurally capable of retaining its molten core and enters the secondary cooling section of the caster.

To replicate the high rate of heat extraction through the mold surfaces in numerical models a *heat flux profile* (HFP) can be assigned to represent the thermal *boundary condition* (BC) along these surfaces in the simulated domain. One common practice for defining a HFP that is quantifiably similar to what would be expected to exist along the mold surfaces, involves employing the *Savage-Pritchard* (S-P) heat flux correlation [8]–[12]. Typically, this approach would entail using known casting conditions to solve for two coefficients from the relation, so that the HFP might be more representative of the observed casting conditions. This approach is widely viewed as a relatively simplistic method of approximating the heat extraction through the mold. As such, it generally assumes that the magnitude of the HFP will remain constant across the width of the surface.

As discussed by Cai and Zhu, the solidified shell of steel will retract inwards, away from the mold surfaces, as it progresses through the mold [13]. This effect occurs as a result of the relative decrease in density that results from continued cooling, and it is known to be more prominent near the corners of the mold. When the shell retracts from the mold surfaces in this manner, radiation becomes the dominant form of heat transfer, while conductive heat transfer decreases significantly. An example of the interfacial gap development at the mold corner can be seen in Figure 1.3, where a noticeable gap was shown to develop over a distance of 400 mm.

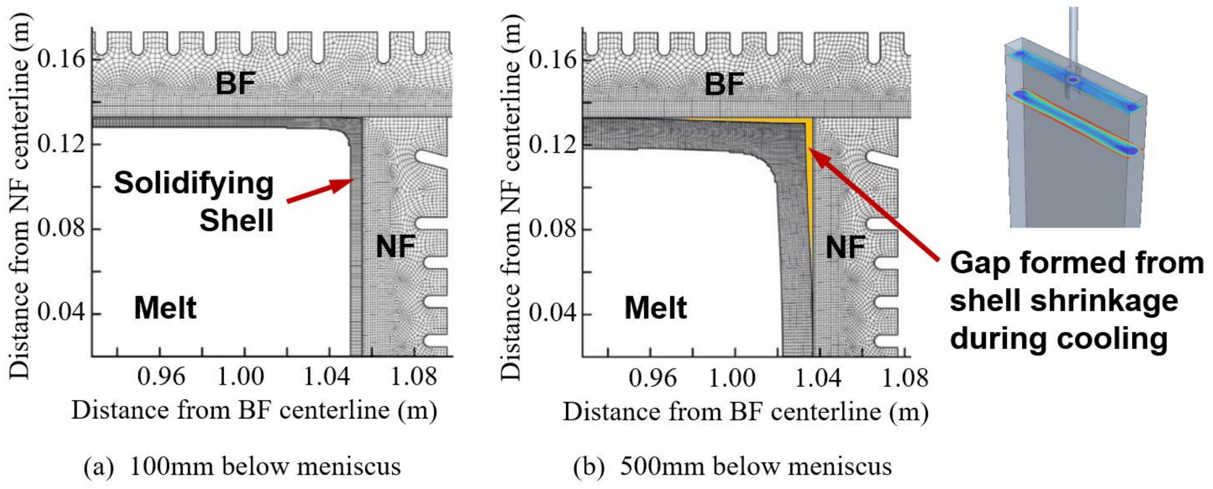

Distance from NF centerline (m)
0.16
0.12
0.08
0.04
BF
Solidifying Shell
NF
Melt
0.96
1.00
1.04
1.08
Distance from BF centerline (m)
(a) 100mm below meniscus
Gap formed from shell shrinkage during cooling
(b) 500mm below meniscus

Figure 1.3. Section views (from above) of the corner-gap development between the mold and shell surfaces, at depths of (a) 100 mm and (b) 500 mm below the meniscus [13].

The influence of this increased interfacial gap in the mold corners was previously identified by B.G. Thomas et al. [8], and was later reproduced by others [9], [12]. However, it was B.G. Thomas et al. that proposed a scaling factor that could be used to decrease the magnitude of the *heat flux* (HF) when within a specified offset-distance of 31 mm from the corner of each surface. This approach has been shown to mitigate the low surface temperatures that would otherwise result along the corners of the simulated shell.

Another area of interest involves the interface between the molten steel and the solidified shell formation, where the solidification process occurs. Commonly referred to as the *mushy zone* (MZ), this region plays an important role in the superheat extraction from the molten core [14]–[21]. Differing thermodynamic equilibrium solubility of the alloy elements, in conjunction with variations in the HF with time and relative solute concentration levels, can ultimately result in segregation. Segregation of the solute formation exists on both micro- and macroscopic scales, and can ultimately give rise to cracks and other problems during the cooling process [22].

Furthermore, the crystal growth is strongly influenced by the thermal and momentum exchange with the internal flow field of molten steel. The jets protruding from the SEN ports are a significant source of thermal and turbulent energy, and will overcome the local rate of heat extraction through the shell – causing the solidification process to be reversed at locations of impingement. Thus, the prediction of the conjugate heat transfer phenomena occurring within the MZ remains a complex and substantial undertaking due to variations in the flow field and HF at any given location.

Validating a Solidification Model

The process of validating a CFD model involves the arrangement of the various parameters – contributing to each of the physics models being applied – in a manner such that the behavior of the system being modeled can be calibrated in-fitting with known conditions. Upon its validation, a model can then be acknowledged as a reliable means of replicating, or possibly even predicting, certain physical phenomena that may occur within the bounds of the simulated domain, and the capabilities of the physics models employed. However, as it is customary for a CFD model to be calibrated against measurement data portraying distinctive characteristics of the flow field, it is not always possible for such measurements to be acquired. In this regard, the environmental conditions of an operating caster are prohibitive of equipment which might be used to gauge the solidifying shell throughout its development. Hence, the current methods of validating a CFD model developed to mimic the solidification process within a CC are not numerous.

As obtaining measurements of the shell growth during operation is rarely a sensible option, it can be reasoned that the next best option would be to measure the shell development when the CC is not operating. However, this approach is also restrictive in that it is only possible following the rare-occurrence of a shell breakout during operation. A breakout is a phenomenon that occurs as a result of improper cooling that ultimately leads to a section of the shell is unable to sufficiently develop to the extent that it is structurally capable of retaining the molten core prior to exiting the mold. In these situations, as the shell is permitted to bulge outwards near the base of the mold, the ferro static pressure causes the vulnerable portion of the shell to stretch beyond its structural limits. The weakened section continues to expand until a rupture forms, and the molten core begins to drain through the newly-formed opening.

The fluidity of the liquid steel exiting the side of the strand allows it to flow unrestrained through the structure of the caster. As the steel progresses, both the equipment mounted within the structure of the caster, as well as the structure itself, could potentially be damaged from exposure to the extreme temperature of the molten steel. Further, all of the steel that drains from the strand will eventually solidify – potentially encasing or clogging the surrounding equipment and structures. Consequently, it is common practice for casting operations to be halted immediately after a breakout is identified.

The fluidity of the liquid steel exiting the side of the strand allows it to flow unrestrained through the surrounding environment. Damage to the surrounding equipment and structure of the caster can result from exposure to the extreme temperature of the molten steel, or by becoming clogged or encased by the solidified metal. In addition to any losses resulting from equipment or structural damages, the time necessary to bring a caster back up to operation following a breakout is equivalent to lost production time. Therefore, it is standard practice for the casting operation to be halted as quickly as possible after a breakout has been identified, in order to minimize losses.

The flow of molten steel from the tundish is cutoff when the casting operation is stopped. With no additional steel being injected into the mold, remaining molten steel will drain through the breakout – leaving a hollow section of the shell extending from the base of the opening in the side of the strand up into the mold. Although continued cooling during the draining process would alter the profile of the shell growth in the unfinished segment of the strand, the final shape can still be considered to represent the relative thickness of the shell development that existed in the moments leading up to the breakout occurrence.

Should it be possible for the husk of the strand to be recovered from the mold successfully, then measurements taken of the shell can be used as the basis from which to compare the simulation results. By probing the simulation results of the predicted shell growth at the approximate locations within the mold where each respective measurement was taken, a comparison between the predicted and measured shell thickness will reveal whether the model is under or over predicting the shell growth. Adjustments can then be made to the appropriate physics model parameters in order to calibrate the model as required.

1.2.2 Adaptive Mesh Refinement

Through the end of the 20th century and up to the present, significant progress has been made in overcoming these limitations through the development of numerical models capable of accurately replicating the behavior of targeted phenomena that occur during the casting process. Continual revisions to these models have produced notable results, though these approaches have primarily implemented predefined mesh representations of a system for the simulation. The mesh utilized in any numerical simulation directly influences the results the model will produce. Construction of an inadequate mesh may yield insufficient refinement for capturing desired aspects of a system, and is likely to generate large areas of needless refinement. However, developing an appropriate mesh capable of replicating the complex behavior of a system requires, not only significant consideration of various factors involved in the construction of the mesh, but also of the physics being modeled, and the solvers to be utilized in evaluating the system. As a result, the process of developing a suitable mesh for a complex system may require days, weeks, or even months of testing to ensure that the solution will not be limited by the mesh.

The high rate of heat extraction occurring in the PC section of a CC, along with the conductive properties of solidified steel, generate a steep temperature gradient within the shell. This effect has been verified in numerous studies, and can be observed in the *three-dimensional* (3D) representation of the predicted temperature-profile shown in Figure 1.4. Therefore, it is necessary for an increased level of refinement to be applied along both the BF and NF surfaces of the domain in order to adequately resolve the temperature profile.

Another area of interest involves time-varying casting conditions. When conducting numerical studies of a CC, it is often convenient to assume the BC to be constant throughout a study. This a valid approach when attempting to replicate conditions and phenomena existing in the caster while the system is operating under relatively steady conditions. However, casting conditions do vary during operation. Depending on the CC and degree of change in the relative casting conditions, the numerical results may not be representative of the actual conditions. Some examples of the variations in the operating conditions that can be expected to occur during a cast would include: adjustments to the casting speed, the increased superheat after a ladle-change, modifications to the mold oscillation frequency, changing flux powders, as well as switching the alloy being cast. Each of these scenarios can be expected to influence either the flow field or the shell development, if not both, within the PC section.

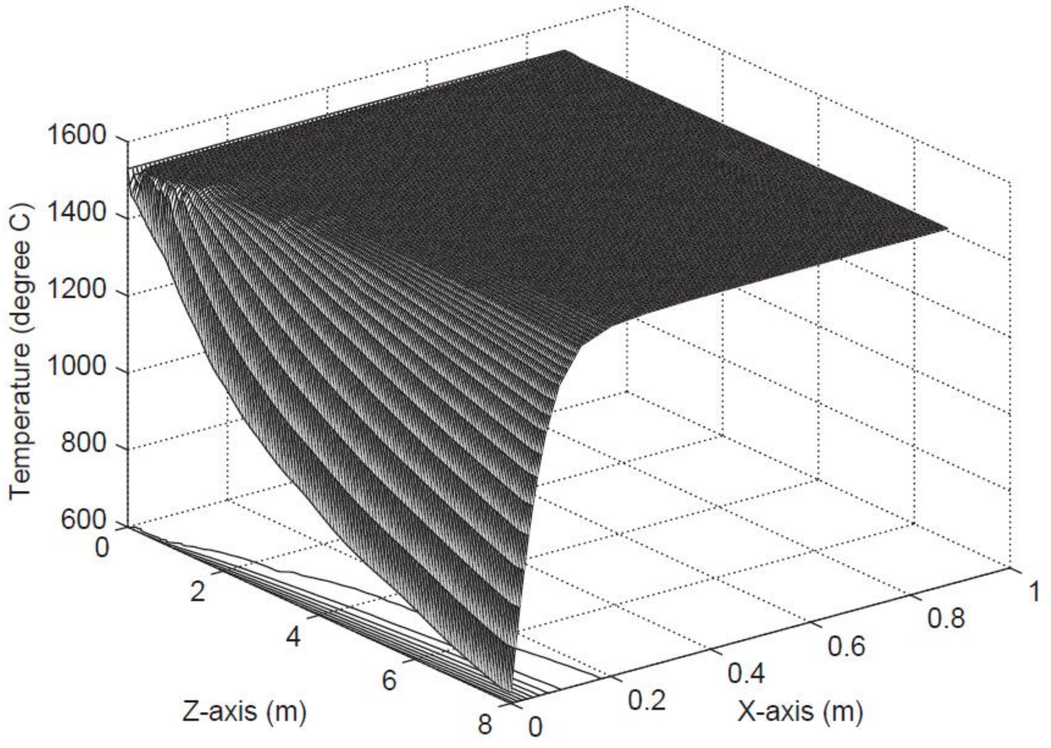


Figure 1.4. 3D-representation of the temperature profile numerically-determined to exist along a strand center plane [23].

Depending on the circumstances considered, it may be necessary for the numerical models to simulate the operating conditions through time in order to capture any phenomena that could be resulting. Under these circumstances, it would be possible for the simulated shell growth to vary in both shape and thickness, in accordance with the active conditions. To account for potential variations in the predicted shell thickness, the applied mesh refinement would need to be sufficient for all of the operating conditions. The use of common methods of applying mesh refinement to blocked-regions would potentially result in large quantities of unnecessary refinement throughout the simulation. Alternatively, more modern methods of applying refinement to regions enclosed within specified reference geometries, allow for the profile of the applied refinement to more-closely follow that of the simulated shell. However, this method poses a similar issue, as the profile of refinement would effectively be ‘fit’ to specific operating conditions; and therefore, may be inappropriate for resolving the shell profile under alternate conditions.

1.2.3 Research Objectives

One objective of the enclosed work involved the development of a numerical model using the Simcenter™ STAR-CCM+™ software, while implementing the Eulerian multiphase, *volume of fluid* (VOF), and Melting-Solidification models for both steady-state and transient simulations. The model was developed for the purpose of replicating the solidification of molten steel within a CC, and is intended for use in future research directed towards better-understanding casting phenomena relating to the shell development, and which are believed to contribute to defect-generation in steel slab production. Casting conditions and measurement data – provided by an *industrial collaborator* (IC) – were employed in defining the BC, as well as for the model validation.

Additional consideration was given towards the potential benefits that might be observed by incorporating an *adaptive mesh refinement* (AMR) procedure during the simulation of the aforementioned solidification model. A *control algorithm* (CA) was developed and incorporated into a simulation macro, using the JAVA software language, in order to manage the AMR operations during the execution of both steady-state and transient simulations of a continuous caster. Methodology and procedures for evaluating the active mesh against the current simulation results were established and defined in the CA. These AMR *criteria tests* (AMR-CT) were configured to provide increased refinement in areas containing shell growth while minimizing the overall cell count to reduce the total computational costs of the simulation. A *proof-of-concept* (POC) test case was developed using a simplified domain to verify the feasibility of utilizing AMR in a full-scale CC simulation.

2. MODELS AND METHODOLOGY

The work presented in this study was conducted in parallel with on-going research dedicated towards the development of a comprehensive numerical model of a CC. Due to the complex nature of the physical phenomena occurring in a CC system, the process of developing a new CC model was divided into a sequence of stages. This approach was intended to provide a means of incorporating and validating the various physics models individually to allow for issues relating to any one model to be isolated and resolved separately; rather than simultaneously.

2.1 Methodology

The heat transfer and solidification models discussed were developed to replicate the solidification of steel in a CC system. These models built on previous work, from which an isothermal flow model – for simulating the flow field of molten steel in a CC – were developed and validated against measurement data referenced from literature, using the STAR-CCM+ simulation software [24]. While the validated model pertained to a multiphase simulation of liquid steel and argon gas, it was believed that the effects of argon gas on the development of the solidifying shell in a CC would be negligible. Therefore, in order to reduce the computational costs of the simulations, argon gas was not considered during the initial validation of the heat transfer and solidification models outlined in this study.

2.1.1 Selection of Simulation Software

Although the previous isothermal-flow model had been developed using STAR-CCM+, consideration was also given towards utilizing the ANSYS® Fluent® software for simulating heat-transfer and solidification in a CC. A comparison of the solidification physics-models provided by each software package revealed that STAR-CCM+ presented greater flexibility in terms of the available parameters and additional models that could be utilized. In addition, the ANSYS solidification model was only found to permit the material solid-fraction to be interpreted as a linear relation with temperature.

Although the solid-fraction curve was treated to be linear in the discussed simulations, one of the future objectives of the encompassing project involves incorporating accurate temperature-dependent material properties – such as the solid-fraction curve – for the specific grade of steel being simulated. This objective is intended to permit future studies to consider how alternate alloys might behave under similar casting conditions. As such, the ability to define the solid-fraction curve as a non-linear function of temperature, using STAR-CCM+, was believed to align more with the overall project-scope.

Additionally, the licensing options available for STAR-CCM+ were found to be less restrictive in terms of the number of simulations that could be conducted at one time, as well as the number of processors that could be utilized to conduct a single simulation. Finally, the meshing algorithms employed in STAR-CCM+ were found to be more robust in terms of the types and capabilities of the meshing-schemes available. Although additional factors had also been considered in the comparison of these software packages, the examples discussed were deemed to hold more significance in the final selection.

2.1.2 Heat Transfer and Solidification Model

A computational model was developed to conduct steady-state and transient simulations of the solidification of steel within a half-mold domain. While the material properties of steel would naturally vary with temperature, and the local composition; for the purposes of the initial model development discussed in this study, the material properties of the simulated steel were assumed to be similar to the constant properties previously modeled by Pfeiler [25]. The behavior of the molten steel is modeled using the Eulerian Multiphase VOF, the Segregated Flow and the Segregated Multiphase Temperature models. The governing equations of the system for mass, momentum and energy conservation are given, respectively as;

$$\frac{\partial}{\partial t}\left(\int_V \rho \, dV\right) + \oint_A \rho \mathbf{u} \cdot d\mathbf{A}_f = 0 \qquad (2.1)$$

$$\frac{\partial}{\partial t}\left(\int_V \rho\mathbf{u}\,dV\right)+\oint_A \rho\mathbf{u}\otimes\mathbf{u}\cdot d\mathbf{A}_f =$$

$$-\oint_A p\mathbf{I}\cdot d\mathbf{A}_f+\oint_A \mathbf{\Psi}\cdot d\mathbf{A}_f+\int_V \rho\mathbf{g}\,dV+\int_V \mathbf{f}_b\,dV-\sum_i\int_A \alpha_i\rho_i\mathbf{u}_{d,i}\otimes\mathbf{u}_{d,i}\cdot d\mathbf{A}_f \quad (2.2)$$

$$\frac{\partial}{\partial t}\left(\int_V \rho E\,dV\right)+\oint_A \left[\rho H\mathbf{u}+p+\sum_i \alpha_i\rho_i H_i\mathbf{u}_{d,i}\right]\cdot d\mathbf{A}_f =$$

$$-\oint_A \mathbf{q}''\cdot d\mathbf{A}_f+\oint_A \mathbf{\Psi}\cdot\mathbf{u}\,d\mathbf{A}_f+\int_V \mathbf{f}_b\cdot\mathbf{u}\,dV \quad (2.3)$$

Here, *p* is the pressure, $\mathbf{I}$ is the unity tensor, $\mathbf{\Psi}$ is the stress tensor, vector $\mathbf{f}_b$ represents all body forces, $\mathbf{q}''$ is the HF vector, and *H* and *E* are the total enthalpy and energy, respectively. The volume fraction and diffusion velocity of each phase are depicted by α_i and $\mathbf{u}_{d,i}$, respectively. The diffusion velocity of the phase is defined as:

$$\mathbf{u}_{d,i}=\mathbf{u}_i-\mathbf{u}_{mix}$$

where $\mathbf{u}_i$ and $\mathbf{u}_{mix}$ designate the velocity vectors of the phase and the mixture, respectively, and $\mathbf{A}_f$ signifies the cell face area vector. The resultant relation is equivalent to the vector representation of the relative velocity of the considered phase with respect to the overall velocity of the mixture in each cell.

The turbulence was evaluated using the *k-omega* (k-ω) *shear stress transport* (SST) model. This model resembles the standard two-equation *k-epsilon* (k-ε) model for unsteady *Reynolds averaged Navier-Stokes* (RANS) simulations; however, it implements different near-wall treatment with blending functions, thereby providing better performance in predicting flows with adverse pressure gradients and separation [26]. The k-ω turbulence equations utilized are of the forms:

$$\frac{\partial}{\partial t}(\rho k)+\nabla\cdot(\rho k\bar{\mathbf{u}})=\nabla\cdot[(\mu+\sigma_k\mu_t)\nabla k]+\xi_k-\rho\beta^*\phi_{\beta^*}(\omega k-\omega_0 k_0) \quad (2.4)$$

$$\frac{\partial}{\partial t}(\rho\omega)+\nabla\cdot(\rho\omega\bar{\mathbf{u}})=\nabla\cdot[(\mu+\sigma_\omega\mu_t)\nabla\omega]+\xi_\omega-\rho\beta\phi_\beta(\omega^2-\omega_0{}^2) \quad (2.5)$$

and are evaluated simultaneously to obtain the values of the turbulent kinetic energy, k, and the specific dissipation rate, ω, in order to solve for the turbulent eddy viscosity as:

$$\mu_t = \rho k \tau \tag{2.6}$$

where τ represents the turbulent time scale.

To account for the additional latent heat being extracted during the solidification process, h_{fusion}, an additional set of calculations are performed in order to correct the enthalpy term, such that;

$$h_{ls}^{*} = h_{ls} + (1 - \alpha_s) h_{fusion} \tag{2.7}$$

Where h_{ls} is the calculated value for the sensible heat, and the solid fraction is evaluated as:

$$\alpha_s = \begin{cases} 1 & if \quad T^* < 0 \\ 1 - T^* & if \quad 0 < T^* < 1 \\ 0 & if \quad 1 < T^* \end{cases} \tag{2.8}$$

and T^* denotes the normalized temperature as:

$$T^* = \frac{T - T_{sol}}{T_{liq} - T_{sol}} \tag{2.9}$$

In order to simulate the solidification process, two momentum sources are applied to represent the increased resistance against relative motion within the MZ, as well as the solid-structure of the shell. As the temperature falls below the liquidus temperature, T_{liq}, the Carman-Kozeny Mushy Zone Permeability model is applied to emulate the flow resistance induced by dendritic growths, as the temperature approaches the solidus temperature, T_{sol}, by treating cells possessing a solid volume fraction, α_s, greater than zero as a porous medium using the Carman-Kozeny equation:

$$K \; [m^2] = \frac{(1 - \alpha_s)^3}{F_K \; c_s \; {\alpha_s}^2} \tag{2.10}$$

From this relation, K represents the permeability of the cell and is evaluated as a function of the relative solid fraction. The additional terms in the relation, c_s and F_K, denote the shape factor of the dendritic-growth and a non-dimensional switching function, respectively. The shape factor is evaluated as:

$$c_s = \frac{c}{{\lambda_1}^2} \tag{2.11}$$

where c represents a shape constant, and λ_1 is the primary dendrite arm-spacing.

The switching function is specified as:

$$F_K = 0.5 + \frac{tan^{-1}\left[s\left(\alpha_s - \alpha_{s,cr}\right)\right]}{\pi} \tag{2.12}$$

Where s is a constant used to regulate how aggressively the switching-function transition occurs, and $\alpha_{s,cr}$ is the critical solid fraction; depicting the point at which the solid grain growth begins to significantly influence the flow resistance.

The permeability contributes to the flow resistance through a porous resistance tensor in the form of Darcy's term, given by:

$$\mathbf{\Theta}_{MZ} = \frac{\mu}{K} \tag{2.13}$$

The resulting momentum source term, representing the porous resistance force acting against the flow velocity is then

$$\mathbf{f}_p = -\mathbf{\Theta}_{MZ} \cdot \mathbf{u} \tag{2.14}$$

The translation of the shell was also incorporated using the Melting-Solidification Flow Stop physics model, which applies a momentum source to restrict the velocity to that of the casting speed for cells containing the solid phase of steel.

Validation Approach

Permission was obtained from an IC to conduct on-site thickness measurements of a vertical shell segment that had been recovered after a breakout occurrence. The breakout was recorded to have occurred along the eastern BF, in proximity to the north NF. The IC had extracted the recovered segment from the BF section of the shell located in the southwest corner of the mold, at the quarter-width location, half-way between the SEN and the NF.

Shell thickness measurements were obtained from a segment of the recovered shell that was approximately 100 mm wide, and roughly 1 m in height. The measurements were conducted at 5 mm intervals along both vertical edges, extending perpendicularly from what was the external surface of the shell to the inner face of the shell. The resulting measurements are presented in Figure 2.1.

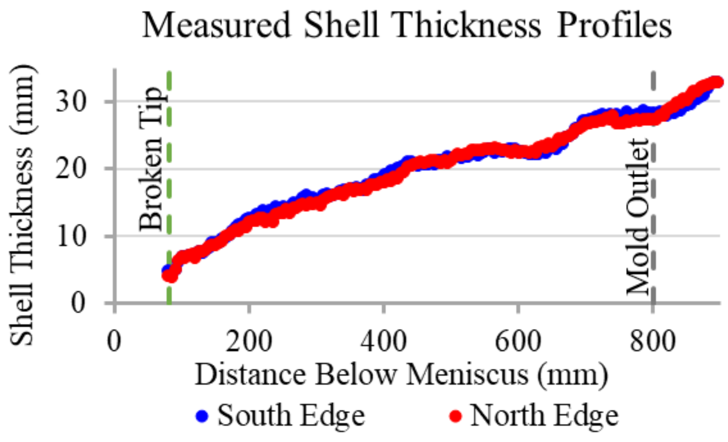


Figure 2.1. Measured shell thickness profiles along the north and south vertical edges of the recovered breakout shell segment.

In the process of performing the measurements, the IC noted that approximately the top 3 inches (76.2 mm) of the recovered shell segment had broken away during the recovery process. Furthermore, due to the additional cooling-time that the steel would have experienced during the draining process, it was suggested that only the upper section of the shell segment, which would have remained above the base of the mold, should be considered for the model validation. Validation of the solidification model was conducted through a comparative analysis of the simulated shell growth at the quarter-width location with the shell measurement data.

Domain-Length Sensitivity Study

The model development discussed in this study was built onto a previously validated isothermal flow model, the initial length of the domain – the distance extending from the meniscus to the domain outlet – had previously been determined without consideration for the influence of the increased flow resistance imparted by the mushy zone. As a result, the domain length used in the first solidification-model simulation conducted of the IC caster, extended nearly 2.2 m below the base of the mold. This measure of extension below the mold was believed to be unnecessary upon incorporating the solidification physics models, therefore, a domain-length sensitivity study was conducted in order to determine whether a shorter domain could be used without influencing the predicted shell profile.

Predicted Shell Thickness Profile Extraction

To compare the simulated shell profile against the shell thickness measurements, two vertical reference planes – being perpendicular with the BF surfaces – were created at the relative locations where the measured edges are believed to have been in the mold. These references planes were then utilized to extract the wall-displacement distances, from the BF surface, at the solidification front, and are displayed in Figure 2.2.

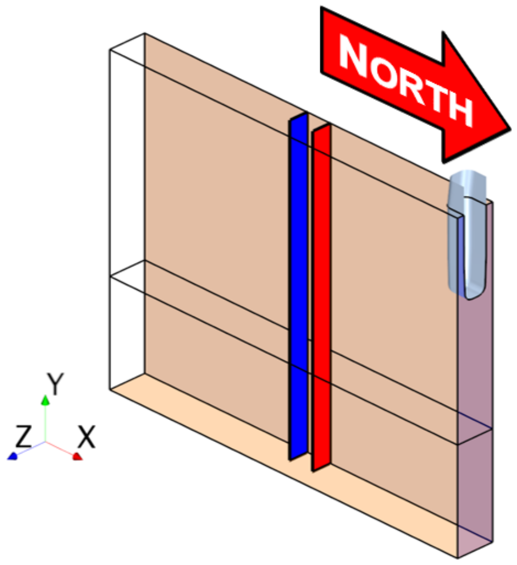


Figure 2.2. Locations of the data sampling planes for the North & South-edges of the recovered shell segment.

An isosurface was generated for a solid fraction of 0.9 and was assumed to represent the solidification front in the discussed simulations. The profile formed by the intersection of the isosurface with each reference plane was therefore considered to symbolize the predicted solidification front along the North and South-edges of the recovered shell segment. The profile-offset distance from the adjacent BF boundary was then extracted at the elevations believed to correspond with the locations where the shell measurements had been performed.

2.1.3 Adaptive Mesh Refinement

In consideration of the preliminary results obtained during the initial development of the heat transfer and solidification models, a CA for implementing the AMR procedures during a CC simulation was developed to mitigate the potential for mesh-dependency, while simultaneously reducing the total simulation time. The intention was for the algorithm to be capable of automating the entire simulation process by managing everything involved with the AMR procedures, as well as some of the more-general simulation operations.

Due to the inherent computational costs involved with the CC models discussed in this study, it was necessary for the simulations to be conducted using 300 processors on a *high-performance computing* (HPC) cluster. While this allowed for the simulations to be performed at greater speeds than alternative tower or desktop resources available, it imposed an additional restriction on the length of time that a simulation would be allowed to continuously run. As a result, each simulation was conducted through a series of sequential-job submission-files; with each submission-file allowing the simulation to run for maximum wall-time of 4 hours.

As it was necessary for the CC simulations to be performed through a series of sequential job-submissions using the HPC-cluster, the CA was also designed so that it would be able to operate regardless of the starting-state of the simulation at the time the CA is launched. This entails that the CA should be able to evaluate the state of a simulation, and then determine the appropriate operations that should be performed for the given conditions. To accomplish this objective, the CA evaluates the following series of logic-tests in order to determine the simulation-state, both during the *start-up operations*, as well as at the conclusion of each simulated time-step:

- the simulation-file is corrupt, or an unrecoverable error exists within the file

- the simulation has completed
- the most recent activity was the completion of a time-step
- the most recent activity was the completion of a meshing operation
- the existing mesh satisfies the AMR-CT for the existing solution

While it is essential for the above tests to be performed continuously, throughout a simulation, it is only necessary for the following to be evaluated during the start-up operations:

- the simulation has a previous solution history
- the simulation should be continued from its existing solution or if the solution history should be cleared and initialized
- a valid mesh exists for the domain

A general workflow diagram of the AMR procedures is presented in Figure 2.3, while a high-level overview of the AMR CA procedures are discussed below, and a condensed version of the full macro is provided in the APPENDIX.

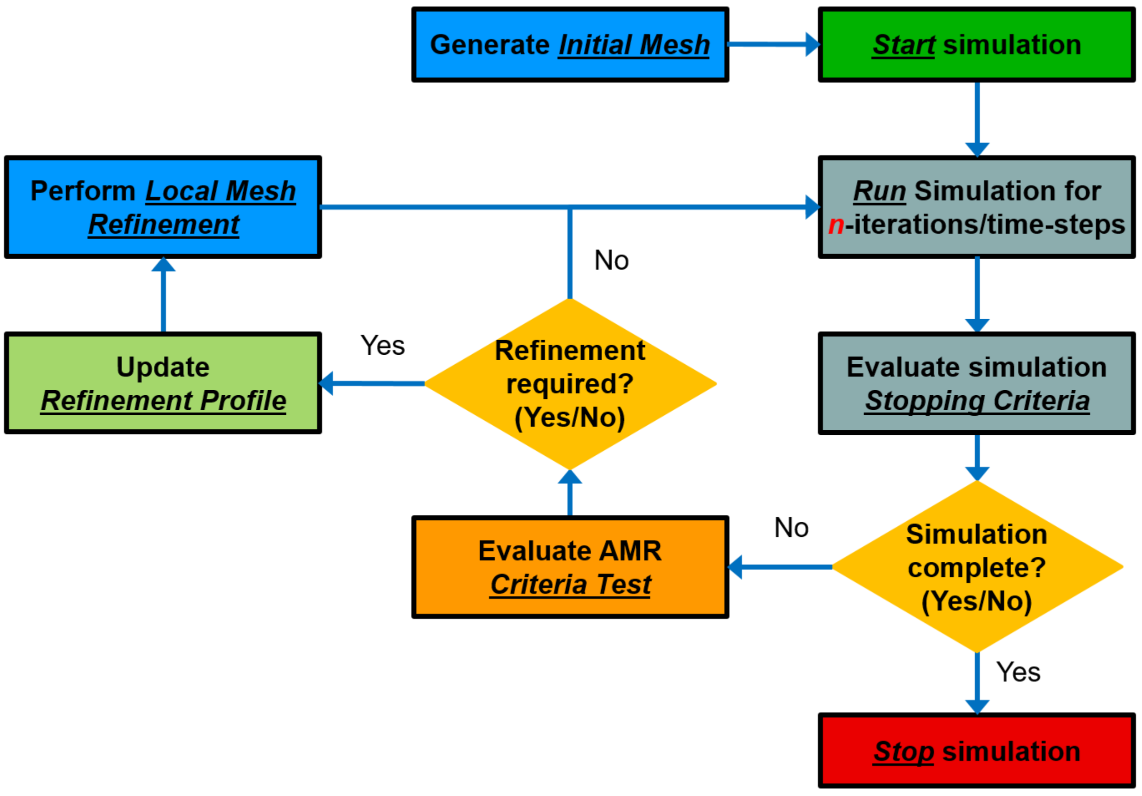

Generate Initial Mesh
Start simulation
Perform Local Mesh Refinement
Run Simulation for n-iterations/time-steps
No
Yes
Refinement required? (Yes/No)
Update Refinement Profile
Evaluate simulation Stopping Criteria
No
Evaluate AMR Criteria Test
Simulation complete? (Yes/No)
Yes
Stop simulation

Figure 2.3. General AMR procedural operation flow diagram.

Control Algorithm – Procedure Overview

After the CA is initially launched, it will perform its start-up operations, during which it will evaluate the current state of the simulation. During the start-up operations, the CA will test for the existence of any of the accepted indicators signifying that the simulation should first have its solution initialized. If the CA detects one of the initialization-indicators, it will then test for the existence of a valid-mesh within the domain. Should no mesh be identified, the CA will then execute the meshing operations following the specified settings for the initial-mesh.

If the CA determines that the simulation does not need to be initialized, or upon initializing the solution, the stopping criteria will then be updated in accordance with the determined simulation-state. The simulation will then be run out for a single time-step. After which, the CA will re-evaluate the simulation-state using the stopping criteria that were identified to have triggered during the recent time-step. If the CA determines that the simulation has completed, then it will begin performing its shut-down procedures, and no further iterations will be performed. Otherwise, the CA will perform the AMR-CT to verify whether the mesh remains valid for the current solution or if an AMR procedure should be carried out prior to continuing the simulation further.

If the mesh fails the AMR-CT, then the CA would then extract new reference geometries from the solution and assign them to the appropriate refinement controls before executing the sequence of geometric- and meshing-operations involved with the AMR procedure – the AMR mesh pipeline. Upon the completion of the mesh pipeline, or if the mesh had been found to have passed the AMR-CT, the stopping criteria would then be updated to the newly discovered simulation-state, before continuing the simulation through an additional time-step. This procedural-loop is continued until either the simulation is completed, or it is forcefully terminated early as a result of the simulation exceeding the available wall-time on the HPC-cluster.

Defining Areas for Refinement Controls

STAR-CCM+ permits for reference geometries to be utilized to define local areas where refinement of the mesh should be performed. The implications of this being that the external surfaces of the reference geometry effectively represent the interface between the refined cells that it encloses, and those external to its volume where refinement is not applied. This concept of *localized refinement* allows for the profile of the applied refinement – the *refinement profile* (RP) – to take complex geometric forms that closely-fit around targeted areas of interest.

As a result, there exists a potential for the resolution of the simulation results to be increased, while simultaneously decreasing the time required for those results to be obtained. However, the potential mentioned above is limited by the *degree-of-freedom* (DOF) of the RP. Should the DOF of the RP be small, then the profile would likely be unable to match closely with the desired region under varied conditions. Therefore, a greater DOF in the reference geometries can be expected to provide a more efficient RP for AMR.

While investigating various methods of employing reference geometries, consideration was initially given towards manipulating the geometric profiles of pre-constructed parts by individually adjusting a collection of dimensioned-parameters. Although this method promises at least one additional degree of freedom in the profile shapes that can be created for each dimensioned-parameter employed, a significant number of parameters would be necessary for the profile of each shape to resemble that of the RP. It was also recognized that each parameter would require the employ of an individual solution probe and that each probe would slightly increase the computation cost for each iteration. Although the incurred simulation time would be negligible for individual probes, this method was ultimately determined to be impractical for use in these simulations, as the total number of solution probes would noticeably subtract from the time-savings the procedure was intended to generate.

The AMR procedure described herein implemented a newly-incorporated feature of STAR-CCM+, which allows for geometric profiles to be generated from the current simulation results. By extracting profiles tuned to a specific value, or range of values, for a select transport property, it is then possible to extract a geometric volume possessing the shape of all the mesh cells meeting the desired criteria. This approach allows for a much larger degree of flexibility in RP shapes that could be obtained without the incurred computational costs associated with the previous method.

As AMR is intended to minimize unnecessary refinement in order to reduce the overall simulation time, it is also possible for an unrestrained CA to negatively influence the results through under-refinement of the domain. One example of this, pertaining to the CC simulation and the AMR CA discussed in this study, would involve a scenario in which the simulation is initialized using the default initial-conditions, rather than a mapped-solution. In this situation, the casting temperature would be assigned to each cell in the domain – resulting in no shell-formation existing at the beginning of the simulation. The CA is intended to implement AMR to increase the level of refinement in regions of shell growth, and to remove additional refinement from regions having only liquid present. In this scenario, it can be reasoned that an unrestrained CA would attempt to remove all of the refinement along the BF and NF surfaces, upon completing the first AMR-CT. This would significantly influence the results, as the steep temperature gradient known to exist along these boundaries cannot be sufficiently resolved without additional refinement.

In order to mitigate the potential of the CA removing refinement that is necessary for resolving the BF and NF boundary layer profiles, additional measures were taken to ensure that a minimum level of refinement shall always be retained. The minimum level of refinement is defined using a specific set of reference geometry, which are separate from the AMR procedures, and do not vary in size or location throughout an individual simulation. As these reference geometries are intended to ensure that the minimum refinement thickness is not exceeded, it is necessary for the minimum thickness to be specified for the particular casting conditions being simulated.

Refinement Criteria – Tracking the Refinement Profile

In order to reduce the computational complexity of the CA, it was determined that tracking the geometric profile in 3D-coordinates would overcomplicate the computations required to track the RP during a simulation. Therefore, an alternative method for assessing whether the RP deviates from its original form – at the time when the active mesh was constructed – was identified in order to minimize the additional memory and time necessary to perform the evaluation.

By recognizing that the RP, by definition, should be located between the refined and core cell types immediately after a meshing procedure has been conducted, it becomes apparent that the active mesh itself could be used to reference the geometry of the original RP. This eliminates the obligation for retaining a virtual copy of the 3D geometry from when the active mesh was constructed; thereby, reducing the overall memory requirements, as well as the additional read/write times that would be required to test for deviations in the RP during each criteria test. Additionally, as the RP should be bordered by select cell types on either side, it should also be possible to identify deviations of the RP by conducting a simple logic test of the cells types directly adjacent to the RP. In essence; should the RP shift towards the core mesh to an extent that a core cell is then located on the opposite side of the RP, then by testing the cell types adjacent to the RP would reveal that a core cell would then be located on the 'refined' side of the RP – thus, indicating that the RP has deviated from its original position. A visual representation of this reasoning is presented in Figure 2.4.

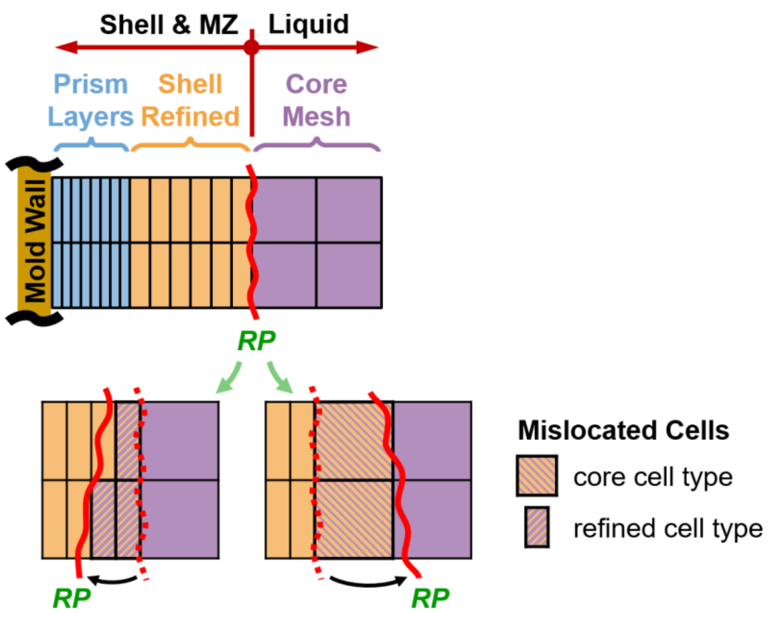


Figure 2.4. Concept for evaluating RP deviations from adjacent cell types.

From the diagrams in Figure 2.4, it can be seen that the RP will need to transverse a greater distance to overtake the first row of cells on the 'core' side than it would on the 'refined' side, due to the larger size of the core cells. As a result, this method of testing will naturally demonstrate a greater sensitivity to fluctuations of the RP towards the refined cells. To account for this increased level of sensitivity, the cells tested on the 'core' side of the RP – the *liquid test front* (LTF) – should be offset to a greater distance than the cells of the *solid test front* (STF).

Refinement Criteria – Defining the Refinement Profile

In the process of determining how the RP should be defined, the underlying objective of incorporating AMR into these simulations was emphasized to be that additional refinement was to be applied in areas where the shell existed. Therefore, various transport properties relating to the shell and MZ were evaluated as potential sources for defining the RP. Some of the more-prominent properties evaluated include both the scalar and gradient values of the following quantities: solid fraction, temperature, velocity magnitude, effective viscosity, average MZ permeability, and the average MZ viscous resistance. These options were then narrowed down to the temperature or the velocity, as the others could either be derived or evaluated from these two properties.

With regard to temperature, it was known that the refinement should be applied to all areas have shell growth. It could therefore be reasoned that refinement should at least be applied to all locations having a temperature equal, or less than, the solidus temperature. Additionally, as the underlying intention of utilizing AMR was to increase the resolution of the predicted shell growth. Additionally, as the formation of the crystal dendrites within the MZ would invoke an increase in the local thermal conduction of the material, the temperature gradient could be expected to begin demonstrating sharp changes at temperatures slightly below the liquidus temperature. A representation of the rate of change in the temperature with respect to horizontal distance from the mold surface is demonstrated in Figure 2.5. Therefore, in an effort to ensure that refinement would exist at all locations where the temperature was below the solidus temperature, and that additional refinement would be applied in areas of the MZ, the RP was designated to be an isosurface generated at the liquidus temperature of the simulated steel for the AMR simulations discussed in this study.

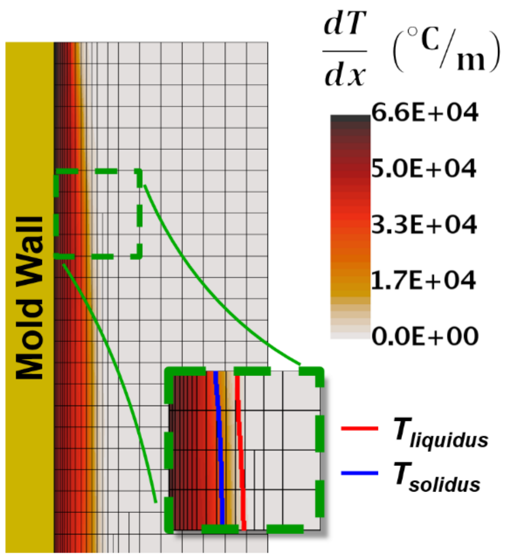


Figure 2.5. Observations of the temperature gradient in relation to the solid- and liquid-fronts in the mold.

Refinement Criteria – Creating the Test-Fronts

In order for the CA to conduct the AMR-CT within the cells adjacent to the RP, a pair of reference geometry needed to be created from the cells forming the LTF and the STF. Therefore, to begin defining the LTF and STF reference geometries, an isosurface was generated on either side of the RP. These isosurfaces were intended to represent the LTF and STF profiles. As such, they were defined so as to be slightly offset from the RP. Additionally, to account for the increased sensitivity of the STF, the isosurface for the LTF was designated such that it would be offset slightly further from the RP than the STF.

Numerous parameters and scalar quantities – having known relations to either the liquidus temperature or the phase of the material – were evaluated for potential use in defining the isosurfaces for the LTF and the STF. From the evaluation, it was believed that the most efficient method of defining the test-front isosurfaces was to select a relative temperature for the LTF and designate a solid fraction-value for the STF. Therefore, the isosurface for the LTF was specified to be 5°C greater than the liquidus temperature, while the STF was defined to be a solid fraction of 0.15. By defining these isosurfaces in such a way, this allows for the reference geometry to adapt with alternate material properties – should the AMR be used to simulate different casting systems.

Upon creating the isosurfaces, a second reference geometry was then created for each test-front. These geometries were created in STAR-CCM+ as *Cell Surfaces*, which are geometries formed from all of the cells in direct-contact with a specified surface. As such, each of the test-front isosurfaces would be fully encased by the resulting geometry, and the cells forming the geometries were then considered to be the corresponding test-front cells.

Refinement Criteria – Delineating the Test Criteria

In order for the CA to determine whether it would be necessary for the AMR mesh pipeline to be executed, a set of logical test-criteria needed to be defined. In this regard, the AMR-CT were established to outline distinct limitations on the conditions that would be deemed acceptable by the CA. As the active mesh is to be evaluated against the most-current simulation results at the end of each time-step, a minimum of two parameters must be assessed in each of the test-front cells. While it would be possible for more than two parameters to be considered in the AMR-CT, only two were utilized in the study discussed. These parameters were implemented in either an *implicative* or *conditional* manner, within the described AMR-CT. Implicative parameters were used in a way to implicate the state of the individual cell being tested. Conditional parameters were compared against a listing of values that were recognized to be acceptable for the indicated state-condition of the cell.

In determining the parameters to be evaluated in the AMR-CT, consideration was given towards quantities previously incorporated into the procedure. As was earlier noted, it would be possible to track the progression of the RP by examining the adjacent cell types. To that end, the type of cell being tested should be considered as one of the parameters. In addition, it was recognized that a temperature-value could be interpolated from the solid fraction-value that had been specified to define the STF. Therefore, by understanding that the cell-temperature had, essentially, already been employed, it was selected as the implicative-parameter.

With the STF and LTF effectively defining the temperature of the cells they contain, the criteria tests were comprised of evaluating whether the cell-type was appropriate for the respective test-front. The cells were classified as being either a *prism-layer*, *refined*, or *core* cell-type, as depicted in Figure 2.4. From the known range of dimensions for each cell-type were known, no overlap was found to exist between the corresponding value-ranges of the *cell aspect ratio* (CAR), which can be defined as:

$$CAR = \frac{m_{Dims}\, n_{Faces}\, V}{\left(\sum_f \left|\mathbf{A}_f\right|\right) \cdot \left(\sum_f \left|\frac{\mathbf{A}_f \cdot d\mathbf{x}}{\left|\mathbf{A}_f\right|}\right|\right)} \tag{2.15}$$

where m_{Dims} signifies the number of dimensions that the mesh exists within – having a value of three for a 3D mesh – n_{Faces} represents the number of faces that comprise the cell, while $d\mathbf{x}$ denotes the position vector from the face centroid to the cell centroid.

By designating a listing of the potential CAR-value ranges within the AMR-CT, it was then possible to distinguish the different cell-types of both the STF and the LTF, using the CAR. The AMR-CT was then set up so to evaluate the CAR of each cell belonging to one of the test-fronts, against the range of values corresponding to the accepted cell-type for the respective test-front. Should a cell be found to have a CAR indicative of the cell-type from the opposing test-front, then that cell would be noted to have failed the AMR-CT. The *ratio of failing cells* (RFC) from each test-front is evaluated against a predefined value signifying the maximum failure ratio for the respective test-front. Finally, if the RFC from either test-front is found to exceed their maximum threshold, then the AMR will be triggered, and the mesh pipeline will be executed before the simulation can be resumed.

Proof-of-Concept Testing

In order to verify that the CA would be capable of tracking the RP and updating the mesh appropriately during a simulation, a POC test case was developed to ensure conditions that would invoke noticeable displacement of the RP. This was accomplished by replicating conditions in which the casting speed is altered during operation. As the casting speed effectively governs the duration of time the steel exposed to the mold-cooling, it can be expected to possess an inversely proportional relation to the shell growth rate within the mold. Therefore, the tests were devised to emulate conditions in which the local shell growth is first increased, and then decreased, by respectively decreasing, and then increasing the relative casting speed.

2.2 Computational Domain and Mesh

2.2.1 Heat Transfer and Solidification Model

A 3D simulation geometry was constructed based on the dimensions and parameters provided by an industrial collaborator. The caster utilizes a stopper-rod for flow control, however, as the simulations conducted for this caster were primarily focused towards the development of the solidification model, the inlet geometry was simplified by neglecting the influences of the stopper-rod and UTN on the flow field and assuming a uniform velocity field at the top of the SEN. The domain was further simplified by the assumption that the influences of cross-flow and energy transfer between the two ends of the mold could be neglected, and that the flow field to either side of the SEN would be symmetric; thereby, allowing for the simulation to be performed using a half-mold domain. The general casting parameters are provided in Table 2.1, and depictions of the geometries used are shown in Figure 2.6.

Table 2.1. General caster geometry dimensions.

Casting Parameter	Dimension (mm)
SEN submergence depth	180
Mold (working) length	800
Mold thickness	152
Mold width	3086

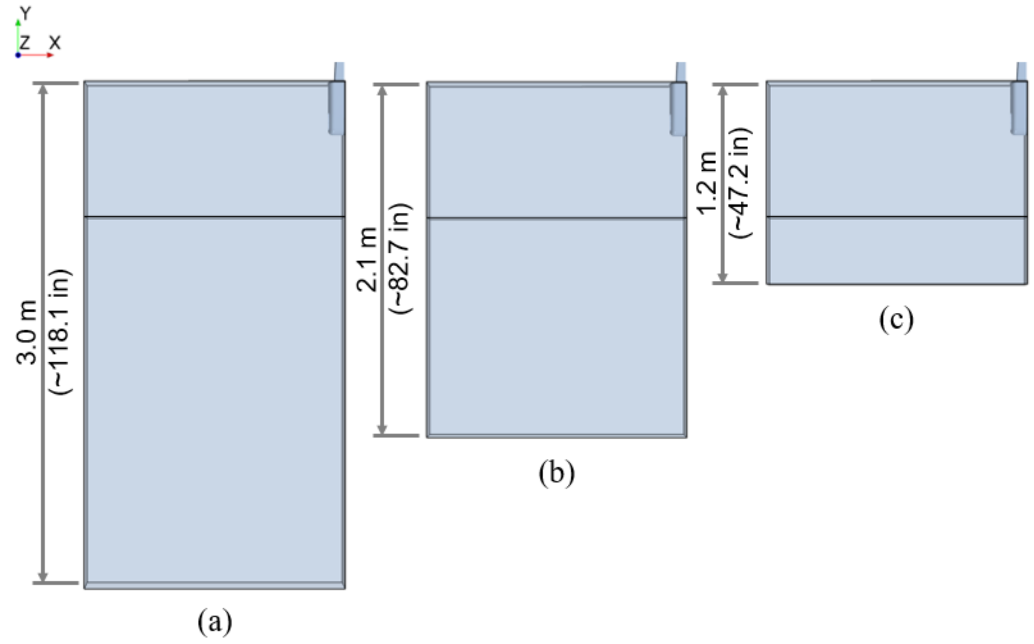


Figure 2.6. Half-mold caster geometries, having strand lengths of (a) 3.0 m, (b) 2.1 m, and (c) 1.2 m, used for domain-length sensitivity study and simulating the heat transfer and solidification model validation.

The three geometries in Figure 2.6 are identical to one another aside from the length of the domain extension below the mold. These geometries were utilized to conduct a domain-length sensitivity study for the solidification model.

The shell measurement data was referenced when determining the thickness of the reference geometry used to specify the regions of local refinement. However, the measurement data only extended to a depth of roughly 1 m below the meniscus, and therefore, would be insufficient for defining the thickness of the local refinement near the domain outlet. In light of this, it was found that the shell thickness profiles seemingly followed the general trend of a power-regression of the form:

$$\delta = 0.3035y^{0.6841} \tag{2.16}$$

and demonstrating an R^2-value of approximately 0.97. In this relation, δ represents the predicted thickness of the shell, y is the depth below the meniscus, and both y and δ are in units of millimeters.

It should be noted that the profile generated by the identified power relation was not intended to perfectly represent the measurement data, but rather that it could be utilized to define an augmented profile that the simulated shell profile could be expected to fall within. For this reason, the relation was adjusted to increase the calculated value of the relative profile-thickness by 5 mm. Thus, the relation utilized to determine the thickness of the local refinement region was defined as:

$$\delta = 5 + 0.3035y^{0.6841} \tag{2.17}$$

Section views of the mesh constructed for the 3.0 m domain length are presented in Figure 2.7. The mesh configuration remained the same for each of the three geometries created of this mold. Therefore, each configuration demonstrated similar characteristics between the meniscus and the elevation at their respective outlet, as depicted in Figure 2.7.

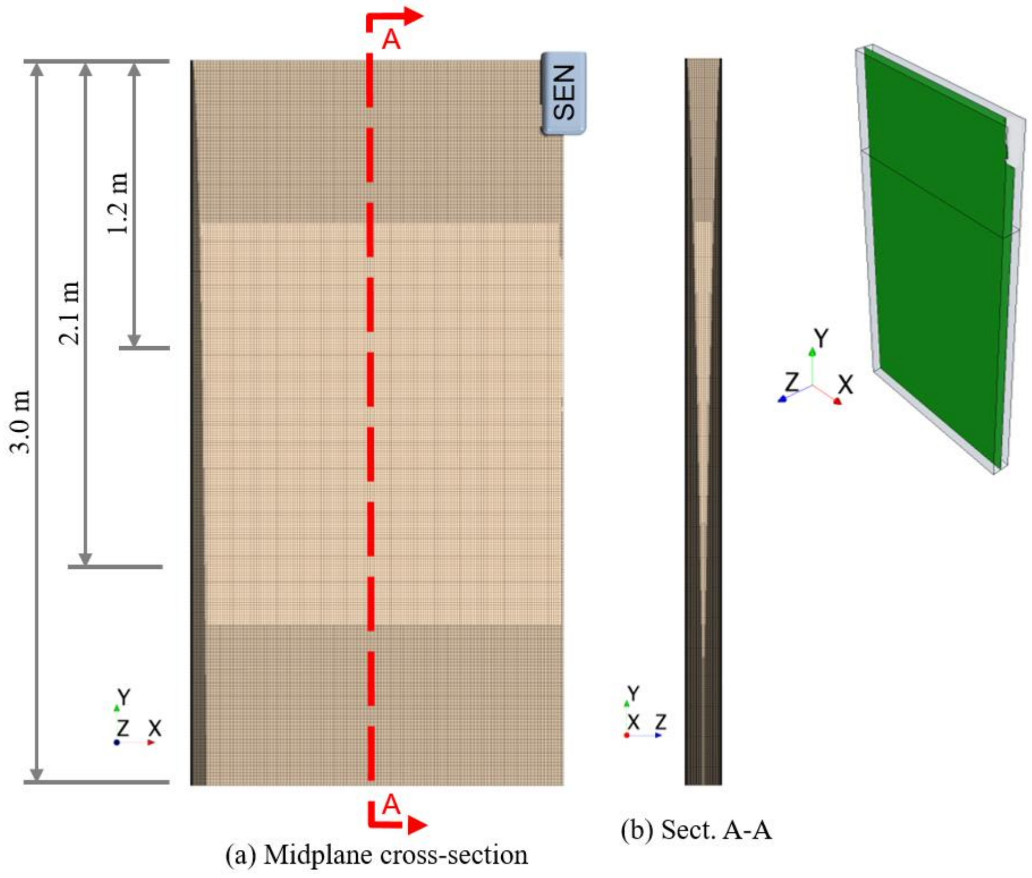


Figure 2.7. (a) Midplane cross-section and (b) Section A-A views of the mesh constructed for the 3.0 m domain length.

From the cross-section view of the mesh along the XY-midplane, as shown in Figure 2.7a, there is a notable increase in the level of refinement in the core-mesh cells near the domain outlet. This can be attributed to the close proximity of the two regions of local refinement along the BF surfaces, near the domain outlet. A depiction of the relative thickness of the two BF refinement regions is depicted in the section view of Figure 2.7b. As these two local refinement regions continue to grow inwards with depth, the maximum growth rate permitted for the core mesh begins to inhibit the cells from reaching their intended size. The total cell counts for each of the three domain lengths used are presented in Table 2.2.

Table 2.2. Resulting cell count for each domain length.

Domain Height Below Meniscus (m)	Total Cell Count (million)
1.2	20.8
2.1	13.4
3.0	6.7

2.2.2 Adaptive Mesh Refinement

The AMR simulations were intended to emulate the solidification of steel along a single surface of the mold. Therefore, a simplified volume was generated to be roughly 2% of the volume in an actual mold. The geometry and mesh defined for each simulation are presented in Figure 2.8. The Mesh arrangement shown in Figure 2.8a, was utilized for the entirety of the control simulation. As this mesh would not be adjusted during the simulation it was necessary that the defined RP be sufficient for encompassing all of the predicted shell growth, for all intended operating conditions. Therefore, the RP was defined for the shell growth that was expected to develop at the slowest of the casting speeds being simulated.

The POC simulations were initialized using the default values for the initial conditions, and therefore, no shell would exist within the domain. As no shell would exist within the domain at the beginning of the simulation, the initial-mesh generated for the AMR simulation – shown in Figure 2.8b – portrays the *minimum-refinement* that should be retained along the boundary surface.

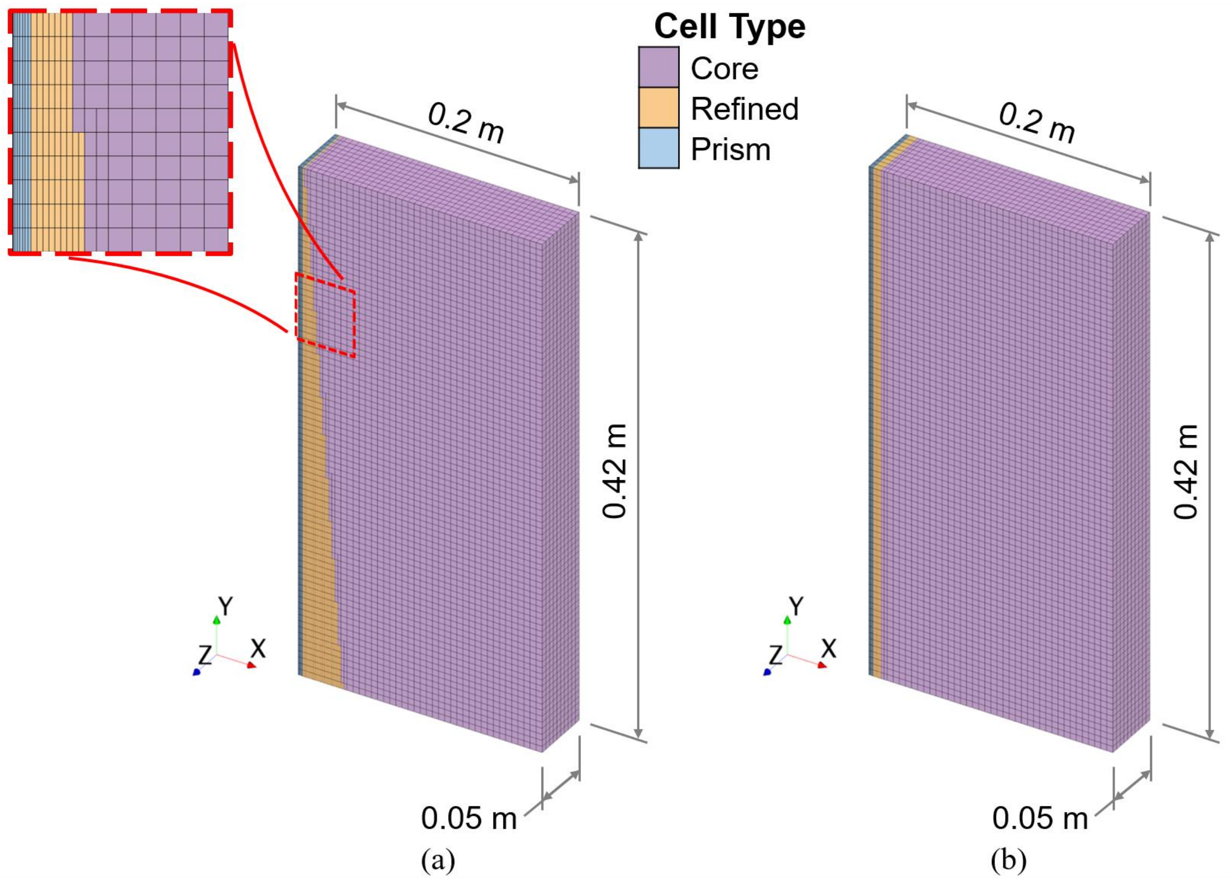


Figure 2.8. Side-by-side comparison of the domains, and the initial mesh cell-type distributions used at the beginning of (a) the control simulation and (b) the AMR simulation.

2.3 Boundary Conditions

2.3.1 Heat Transfer and Solidification Model

The solidification model simulations were performed using BC defined to closely resemble the operating conditions existing in the lead-up to the breakout occurrence. The boundary-types assigned to each of the surfaces are presented in Figure 2.9. The effects of cross-flow and heat transfer across the YZ-center plane were assumed to be negligible. Therefore, the YZ-plane surfaces of the SEN and the strand were each treated as a symmetry plane.

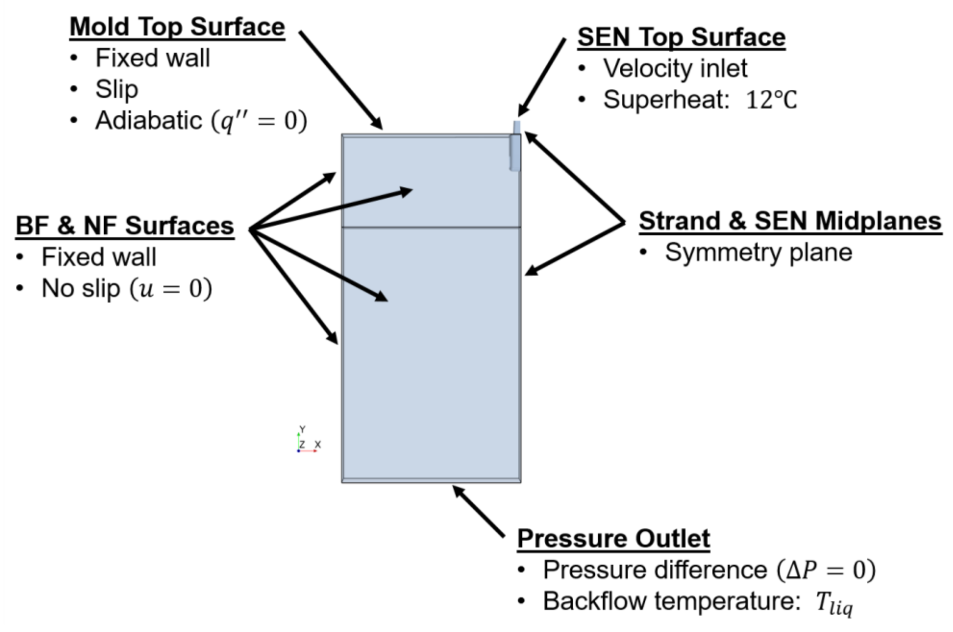


Figure 2.9. Boundary-types assigned in the solidfication model simulations.

The top surface of the mold represents the interface between the molten steel and the insulating flux powder layer – the *meniscus*. Due to the lower melting temperature of the flux, a layer of liquid flux will exist between the molten steel and the powder flux. Therefore, it was assumed that boundary-shear effects at the meniscus could be neglected, and the top surface of the mold was treated as a slip wall. Although the SEN and the meniscus would actually be sources of heat-loss in an operating caster, these were considered to be negligible, relative to the heat extraction rates existing along the BF and NF surfaces, and were defined as adiabatic surfaces.

The top surface of the SEN and the bottom surface of the strand were defined as a velocity inlet and pressure outlet, respectively. A negligible pressure difference was assumed to exist at the domain outlet. A backflow temperature at the outlet could not be quantified, however, it was believed that any backflow at the outlet would only involve the liquid-phase of the steel. Therefore, it was reasoned that the backflow would be, at minimum, the liquidus temperature.

A collection of recorded data depicting the active casting conditions leading up to the breakout occurrence were provided by the IC, and were utilized to derive the BC used for the inlet velocity, inlet temperature, casting speed, and the mold surface HFP’s. Prior to the breakout occurrence, a sticker alarm had resulted in the casting speed being decreased from approximately 40 ipm to 20 ipm to allow for additional heat extraction within the mold to sufficiently increase the shell thickness around the sticker location. Conservation of mass and the assumption of constant density allowed for the inlet velocity to be approximated from the relation:

$$u_{inlet} = A_{inlet} \cdot \dot{V}_{steel} \tag{2.18}$$

where $\dot{V}_{steel}$ represents the volumetric flow rate of steel through the domain, and is assumed as:

$$\dot{V}_{steel} \approx A_{outlet} \cdot u_{cs} \tag{2.19}$$

In these relations, u_{inlet} is the velocity of the molten steel entering the domain through the SEN, u_{cs} is the casting speed, while A_{inlet} and A_{outlet} represents the cross-sectional area of the flow field at the inlet and outlet, respectively.

Heat extraction through each mold surface was modeled to be a HF BC for the simulations discussed in this study. Recorded thermocouple and cooling water data – provided by the IC – were utilized to determine the average HF for each of the respective BF and NF surfaces of the mold. However, as the HFP along a mold surface is known to decrease with increased distance below the meniscus, it was decided that applying an average HF for the BC along the mold surfaces of the domain would not be sufficient for replicating the shell growth. As previously discussed, a commonly used method for approximating the HFP is to implement the S-P HF correlation [9]–[12]; and represented as:

$$q''(y) \approx a - b\sqrt{\frac{|y|}{u_{cs}}} \tag{2.20}$$

where a represents the peak HF value existing at the meniscus, b is the HF depreciation coefficient, y denotes the vertical distance below the meniscus, and the casting speed is evaluated in units of meters-per-minute. The HFP produced from this relation exhibits a peak value at the meniscus and resembles a parabolic curve, which decreases with depth. The general form of the S-P HF correlation is a function of the casting speed and the depth below the meniscus, and it is commonly considered an adequate means of approximating the HFP from known operating conditions, by solving for the corresponding a and b coefficients. It should be noted that the general form of the S-P HF correlation only varies with position along the vertical axis. Without consideration for the lateral position, the general S-P relation must assume the HFP to be constant along the entire width of the surface.

As previously mentioned, the increased rate of thermal shrinkage known to occur in the mold corners results in a notable reduction of the local heat transfer between the shell and the mold surfaces [13]. Gonzalez et al. recognized that the general S-P correlation could not provide a good approximation of the heat transfer in areas where notable interfacial gaps exist [9]. Likewise, other studies have acknowledged that unrealistic surface temperatures result along the corners of the simulated shell if the decreased heat transfer is not accounted for in the BC's [8]–[12]. To account for the thermal influence of the corner gaps, Thomas et al. demonstrated that a scaling factor could be applied to the HFP such that it would be decreased to 67% of its standard value when within an offset distance of 31 mm from the mold corners [8]. This approach was adopted for use in this study.

To aid in assessing how the scaling factor would be distributed along each surface, the working surfaces of the BF and NF were considered to be comprised of three regions. Two of these were defined as being the corner-offset regions – existing within 31 mm of the corners, while the third region was composed of the surface area remaining between the two corner-offsets. As such, a relation for the working width of the mold was established as:

$$\Delta W = \Delta W_{std} + 2\Delta W_{corner} \tag{2.21}$$

where ΔW is the width of the working area for the considered surface, ΔW_{corner} symbolizes the corner-offset distance of 31 mm, and ΔW_{std} represents the width of the remaining surface existing the two corner-offsets; where the standard HFP is applied.

By incorporating the scaling factor into the S-P HF correlation, the relation could then be defined as:

$$q''(y) \approx \eta \left[a - b \sqrt{\frac{|y|}{u_{cs}}} \right] \tag{2.22}$$

where η is the HFP scaling-factor and was defined as a conditional relation, such that:

$$\eta(w) = \begin{cases} 1\,, & |w| < \left(\frac{\Delta W}{2} - \Delta W_{corner}\right) \\ \frac{2}{3}, & |w| \geq \left(\frac{\Delta W}{2} - \Delta W_{corner}\right) \end{cases} \tag{2.23}$$

where w represents the horizontal displacement from the surface center.

As was previously noted, two known conditions were necessary to solve for the corresponding values of the S-P a and b coefficients. Since the measurement data provided by the IC represented the actual conditions of the casting system, the average HF of each surface was selected to be one of the known conditions. While the S-P HF relation could not be examined against the average HF of each surface directly, a comparison could be made between the measured and calculated values for the total *heat transfer rate* (HTR) of each surface. The *actual* HTR was assumed to be the product of the working surface area and the corresponding average HF, and is expressed as:

$$\dot{q}_{actual} = A \cdot \bar{q}''_{Msrd} = (\Delta Y\ \Delta W) \cdot \bar{q}''_{Msrd} \tag{2.24}$$

Here, $\dot{q}_{Actual}$ symbolizes the actual HTR and $\bar{q}''_{Msrd}$ is the average HF recorded for the considered surface, while ΔY, represents the corresponding height of the working surface area, A, of the considered BF or NF. The total HTR resulting from the general S-P relation, the HFP was integrated over the working area of each considered surface, to produce the following relation:

$$\dot{q}_{Tot} \approx \left(a \cdot \Delta Y - \frac{2}{3} b \sqrt{\frac{|\Delta Y|^3}{u_{cs}}} \right) \cdot \Delta W \tag{2.25}$$

Previous work by Brian Thomas demonstrated that the HF value, at a location 25 mm below the meniscus, could be approximated to be 70% greater than the average HF for the mold surface [27]. This approach allowed for a value of the HF to be approximated at a single elevation as:

$$q''(0.025\,m) \approx 1.7\bar{q}''_{Msrd} \approx a - b\sqrt{\frac{(0.025\,[\mathrm{m}])}{u_{cs}}} \tag{2.26}$$

By evaluating the total HTR from the measured HF of each surface, and by approximating the HF value at a distance of 25 mm below the meniscus, it was then possible to solve for values of the S-P coefficients. However, in doing so, the S-P HFP would be assumed to remain constant along the entire width of each mold surface, which is known to result in unrealistic shell-surface temperatures. As such, consideration was then given towards determining whether the scaling factor should be included when evaluating the a and b coefficients, through comparison of the calculated and actual HTR for each of the mold surfaces.

From the comparison, the calculated total HTR's for the BF surfaces were found to demonstrate a negligible difference from the measured conditions, with the percent error being around 0.6%. Conversely, the a and b coefficients for the NF HFP were found to underpredict the total HTR for each surface by nearly 13.6%. The percent error difference between the BF and NF surfaces was attributed to the high-to-low aspect ratios of the BF and NF surfaces, respectively. As the total surface area contained within the corner-offset regions account for approximately 2% of each BF, whereas, the same offset-regions make up roughly 41% of each NF. Therefore, it was deemed necessary for the scaled-HF to be considered when determining the a and b coefficients for the respective S-P HFP's.

The relation for the total HTR was then redefined to account for the heat transfer through the corner-offset regions by integrating Eqn. (2.22) over a considered mold surface. A simplified form of this relation can be shown as:

$$\dot{q}_{Tot} = \dot{q}_{std} + 2\dot{q}_{corner} \tag{2.27}$$

Here, $\dot{q}_{Tot}$ is the total HTR of the surface, and is defined to be the sum of the *standard* HTR, $\dot{q}_{std}$, and the combined HTR from the two corner-offset regions – the *corner* HTR, $\dot{q}_{corner}$. With respect to the considered mold surface, $\dot{q}_{std}$ signifies the HTR accruing from the *standard* S-P HFP along the surface-segment existing between the two corner-offset regions. The relations for the standard and corner HTR's were evaluated similarly to that of Eqn. (2.25), however, they employed the width corresponding to their respective sections of the surface, with the scaling factor also included for the corner HTR, and can be represented as:

$$\dot{q}_{std} = \left[\Delta Y\, a - \frac{2}{3} b \sqrt{\frac{|\Delta Y|^3}{u_{cs}}}\right] \cdot \Delta W_{std} \tag{2.28}$$

$$\dot{q}_{corner} = \frac{2}{3}\left[\Delta Y\, a - \frac{2}{3} b \sqrt{\frac{|\Delta Y|^3}{u_{cs}}}\right] \cdot \Delta W_{corner} \tag{2.29}$$

where the height and width-quantities were specified in units of meters, and the casting speed is in units of meters-per-minute.

By setting Eqn. (2.27) equal to Eqn. (2.24), and by assuming the standard HFP to be equivalent to Eqn. (2.26), the system was then rearranged and solved for the S-P coefficients. The resulting relations for the a and b coefficients could then be simplified to the forms:

$$a = \frac{\dot{q}_{std}}{|\Delta Y|\, \Delta W_{std}} + \frac{2}{3} b \sqrt{\frac{|\Delta Y|}{u_{cs}}} \tag{2.30}$$

$$b = \frac{0.7 \dot{q}_{std} \sqrt{u_{cs}}}{|\Delta Y|\, \Delta W_{std} \left[\frac{2}{3}\sqrt{|\Delta Y|} - \sqrt{(0.025\ [\mathrm{m}])}\right]} \tag{2.31}$$

These relations were then evaluated simultaneously to determine the coefficients for the standard HFP of each BF and NF surface. Upon determining the coefficients, the total calculated HTR was then evaluated against the actual HTR and the resulting percent error was found to be negligible for each surface.

As time-averaged measurement data were utilized in determining the standard HFP coefficients, it was assumed that a single HFP could be used to define the thermal BC for both BF surfaces. Therefore, the average of these two coefficients were determined, and the subsequent HF relation was assigned to both BF surfaces. Thus, the resulting relations for the BF and NF HFP could roughly be defined as:

$$q''_{BF}(y) = 2.1 - 1.2\sqrt{\frac{|y|}{0.51}} \tag{2.32}$$

and

$$q''_{NF}(y) = 2.6 - 1.5\sqrt{\frac{|y|}{0.51}} \tag{2.33}$$

where q''_{BF} and q''_{NF} are depicted in units of megawatts-per-meter squared, and y is shown in units of meters. The relations for the BF and NF HFP's were each assigned to their respective BC using a *user-defined function* (UDF), within STAR-CCM+. The resulting HFP's are presented in Figure 2.10, along with the corresponding HF values that were assumed for a depth of 25 mm when solving for the a and b coefficients.

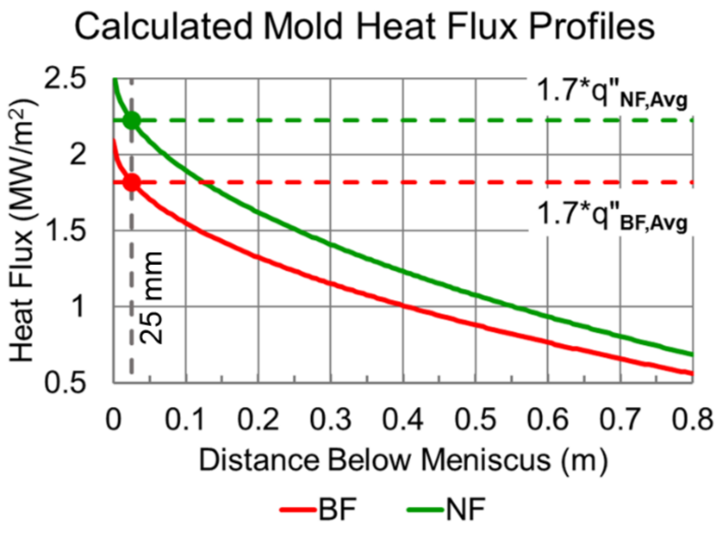


Figure 2.10. Derived heat-flux profiles defined for the mold BC in the solidification model simulations.

From previous simulation results, and per the suggestion of the IC, a simple correlation was utilized to define the SC BC using *heat transfer coefficient* (HTC) values, rather than as a constant HF. The cooling data for the secondary cooling section of the caster was considered to be proprietary information, and therefore was not incorporated into this model. Previous work by Meng and Thomas demonstrated a simple correlation for evaluating an effective HTC for each spray zone and roll pairing by summing heat transfer, by means of convection, conduction and radiation, over the respective surface area of the slab from which they would occur [28]. A profile for the HTC-values was derived from the secondary cooling parameters and the simulated surface temperature data provided in the report. The separate zones, the surface temperature profile, and locations of each nozzle and roll used in the evaluation of the HTC-values, along with the derived HTC profile are displayed in Figure 2.11

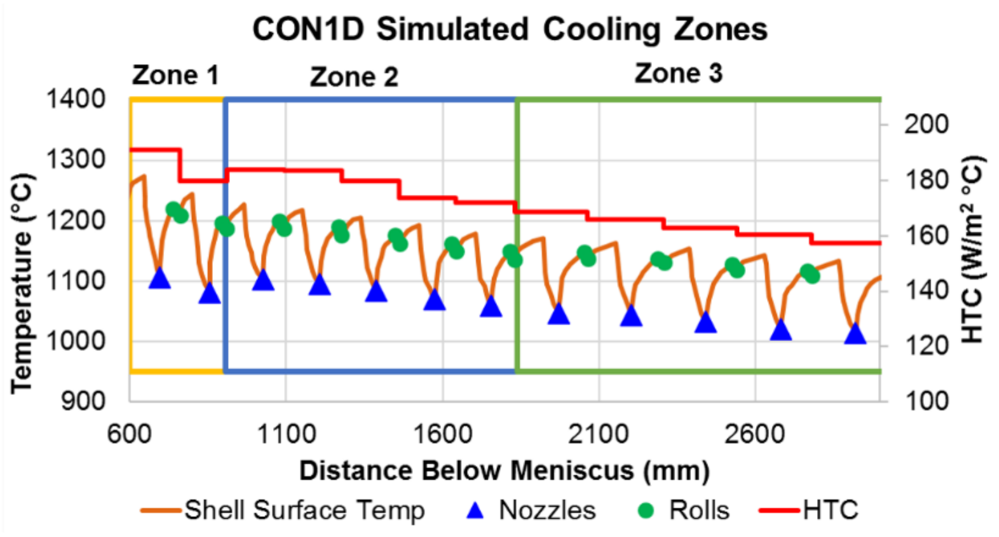
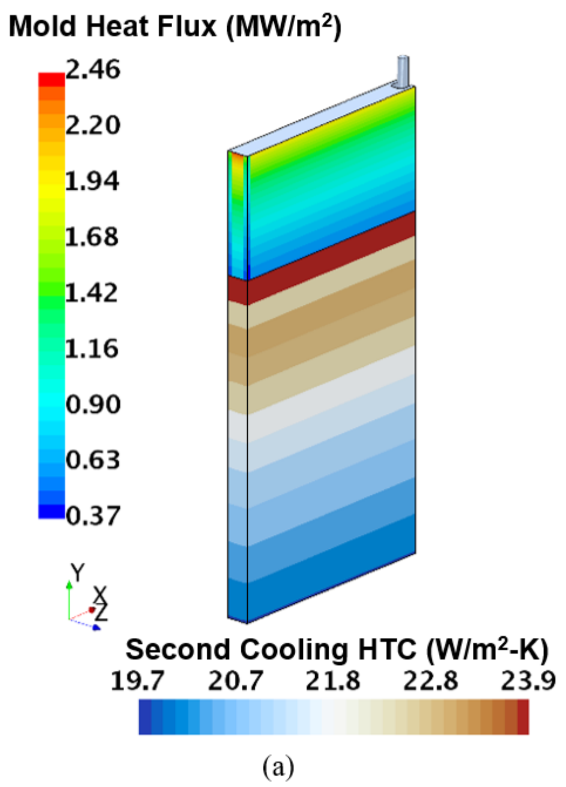


Figure 2.11. HTC-values derived from the simulated surface temperature profile and nozzle-roll parameters from literature [28].

The HTC-values were assigned for the BC of the secondary cooling section in the IC caster simulation, and are interpolated as a function of the vertical position in the domain. Color contours of the defined BC and the resulting surface HF are shown in Figure 2.12.



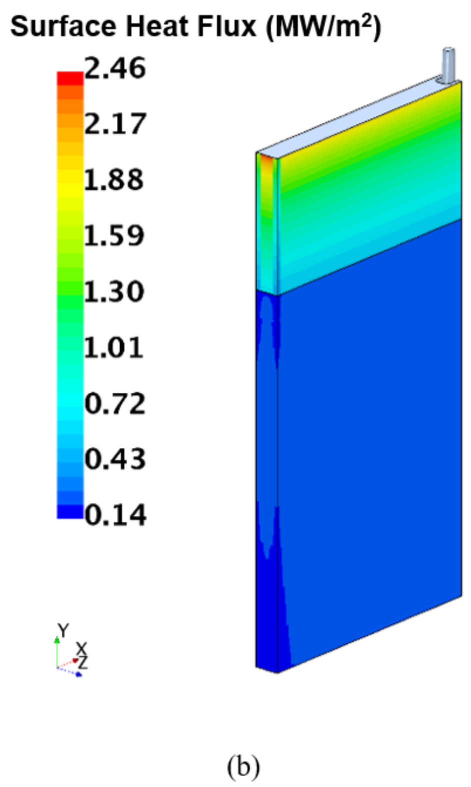


Figure 2.12. BF and NF surface contours depicting (a) the applied BC and (b) the resulting surface HF for the 3.0 m length domain.

2.3.2 Adaptive Mesh Refinement

The BC for the AMR POC simulations were defined so to create a flow field that would run parallel to the cooled-surface. This was intended to provide conditions which would promote consistent shell growth rates for the respective inlet velocity. Therefore, the top and bottom surfaces were defined as being a velocity inlet and pressure outlet, respectively. A HFP was applied to the wall boundary of the cooled surface. The profile of the applied HF is shown in the plot that is offset-left of the cooled surface in Figure 2.13.

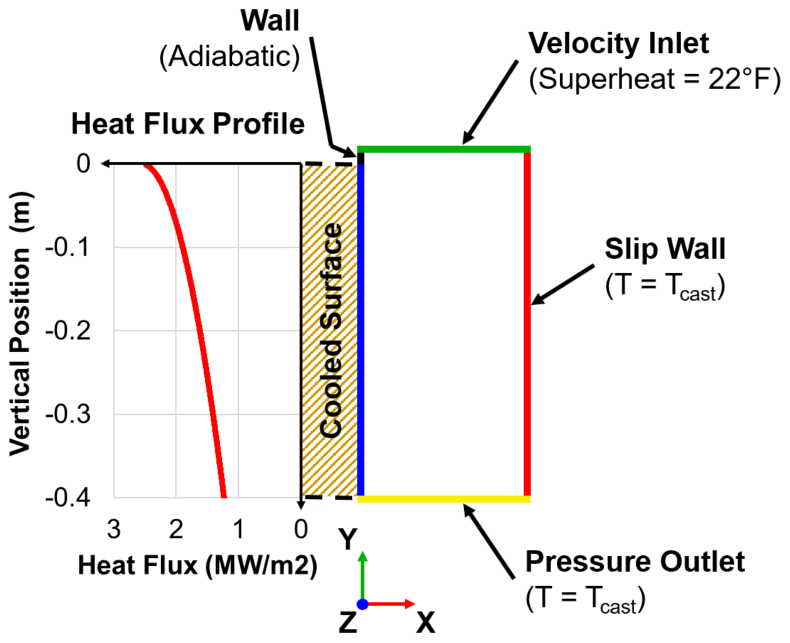


Figure 2.13. The boundary-types and HFP assigned in the AMR POC simulations.

A section of vertical surface – measuring 0.02 m in height and being located between the inlet and the cooled surface – was treated as a no-slip wall, and was defined to be adiabatic. This was done so to prohibit any shell growth from forming directly below the inlet surface. The surface opposite to the cooled surface was defined to be a slip wall having a temperature equal to the inlet temperature. Finally, the remaining front and back surfaces of the domain were treated as symmetry planes.

The inlet velocity magnitude was treated as a function of the simulated time, and was defined as:

$$u(t)\ [\text{ipm}] = \begin{cases} 80, & t \leq 20\ [\text{s}] \\ 80 - 15 \cdot (t - 20), & 20\ [\text{s}] < t < 24\ [\text{s}] \\ 20, & 24\ [\text{s}] < t \leq 120\ [\text{s}] \\ 20 + 5 \cdot (t - 120), & 120\ [\text{s}] < t \leq 124\ [\text{s}] \\ 40, & 124\ [\text{s}] < t \leq 200\ [\text{s}] \end{cases} \qquad (2.34)$$

3. RESULTS AND DISCUSSION

3.1 Heat Transfer and Solidification Model

3.1.1 Domain-Length Sensitivity Study

Three steady-state simulations were conducted of the IC caster, using three different domain lengths, to determine whether a shortened domain would influence the predicted shell profile. The findings of this study showed no discernable difference between the predicted shell profiles within the elevation range where valid shell measurements had been obtained. From these findings, the three domain-lengths tested were not believed to influence the simulated shell growth. Therefore, as the larger domains tested incurred greater computational costs – required longer periods of time for each simulation to complete – the shortest domain was selected for use in conducting the described transient simulation.

3.1.2 Solidification Model Validation

The predicted shell thickness profiles were extracted from both the steady-state and transient simulations, at the locations believed to represent the original positions of the North and South-edges of the recovered shell segment within the mold. A comparison of the predicted and measured shell thickness profiles for both edges is presented in Figure 3.1, where the predicted profiles can be seen to exhibit similar trends with the shell measurement data.

From Figure 3.1, a distinct decrease in the shell growth rate was noted in the profile of the predicted North-edge at around 0.5 m below the meniscus. This decrease in growth rate can be seen to match closely with the profile trends of the measurement data. However, this change was not found to be present in either of the predicted profiles for the South-edge. As such, the average percent error of the predicted South-edge profiles was found to be greater than those for the North-edge. The average percent error for each edge is presented in Table 3.1, along with the overall for each simulation.

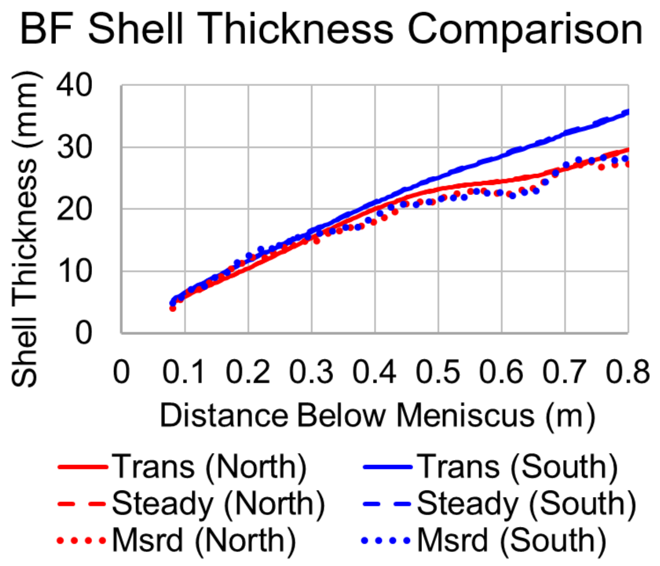


Figure 3.1. Comparison plot of the predicted shell growth from the steady-state and transient simulations with the shell thickness measurements.

Table 3.1. Average percent error of the simulated shell thickness at the North and South-edges of the recovered shell segment

Simulation Type	Average Percent Error (%)		
	South Edge	North Edge	Overall
Steady-state	16.0	7.0	11.5
Transient	14.4	6.6	10.5

Upon collectively overviewing the content of both Figure 3.1 and Table 3.1, the difference between the steady-state and transient results, for the respective North and South-edges, can be said to be negligible. Further review of the velocity and temperature fields of the steady-state and transient results revealed similar circumstances. Moreover, by recalling that constant BC's had been defined for the transient simulation, the results produced from this simulation can be assumed to be representative of a pseudo-steady casting condition. Therefore, the results from the steady-state and the transient simulations were considered to be equivalent to one another. With this understanding, the transient results were also assumed to be indicative of the steady-state results.

While Table 3.1 only provides an outlook of the overall error found along each edge, it is apparent from Figure 3.1 that the predicted profiles matched more-closely with the measurement data in the upper half of the mold. To better-discern the error-dispersion along the North and South-edges, a percent error distribution chart is provided in Figure 3.2. In addition, a parity plot depicting the predicted-to-measured shell thickness is presented in Figure 3.3.

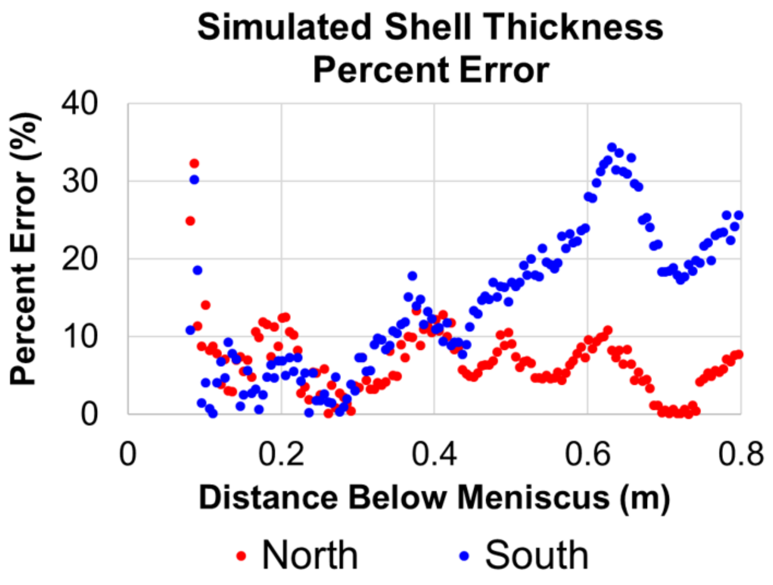


Figure 3.2. Percent error distribution with distance below the meniscus for the predicted shell growth from the transient simulation.

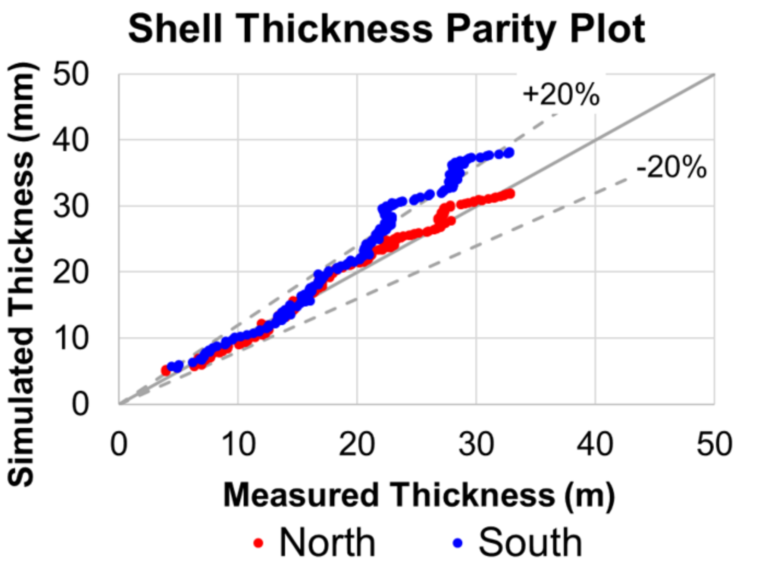


Figure 3.3. Parity plot of the predicted and measured thicknesses for the North and South-edges of the recovered shell segment.

From Figure 3.2, a sudden spike in the percent error can be identified at the top of the shell profile for both edges. In consideration of the relative uncertainty for the vertical position of the recovered shell segment within the mold, along with the magnitude of the shell thickness at these locations, it is believed that a slight discrepancy might exist with the estimated vertical-offset distance – being 3.0 inches – that was applied to the measured values. With this in mind, should the top four measurements be removed from consideration, then the error in the predicted shell profile for the North-edge would largely fall below 10%. However, as previously discussed, this would not hold for the predicted profile of the South-edge as well, as the percent error tends to increase in the lower half of the mold.

As previously discussed in regards to Figure 3.1, the profile for the predicted shell thickness of the South-edge fails to demonstrate a decrease in the growth rate at approximately 0.5 m from the meniscus. However, from the depicted profiles for the South-edge in Figure 3.2 and Figure 3.3, the simulated shell growth begins to deviate more-notably from the measured values at a depth of around 0.4 m, with the peak error location being slightly below 0.6 m. Through additional consideration for the percent-error distribution depicted in Figure 3.2, it can be observed that the North and South-edges demonstrate similar profile trends for the ranges falling between the meniscus and around 0.5 m, as well as from approximately 0.6 m to the mold outlet. This inferring that a discernible difference between the North and South percent error-trends can be recognized between 0.5 m and 0.6 m below the meniscus, which echoes previous observations made from Figure 3.1.

Additionally, for the ranges having similar trends in the percent error between the North and South profiles, as found in Figure 3.2, the trend of the percent error for the South profile can be seen to demonstrate greater rates of change than those of the North profile. From these observations, it can be reasoned that the increased error in the predicted shell growth for the South-edge, likely results from a difference in the local heat transfer conditions. By recalling the defined HFP that was assigned as the BC for both BF surfaces, the rate of heat extraction from the domain can be assumed to be identical for the two locations where the predicted shell thickness is being sampled. As such, it can be reasoned that the applied BC does not represent a significant contribution towards the discrepancy between the predicted profiles of the North and South-edges.

Consideration was then given towards the heat transfer of the internal flow field. Further review of the temperature-field revealed an unequal distribution of the superheat along the North and South-edges of the recovered shell segment. Reference lines – depicting the relative positions of the North and South-edges of the shell segment, as well as for depths of 0.4 m and 0.5 m below the meniscus – are shown overlaid atop a color-contour plot of the superheat delivery, in Figure 3.4, along the center plane of the domain. Direction vectors, depicting the tangential flow along the center plane, are also shown to clarify how the superheat was distributed.

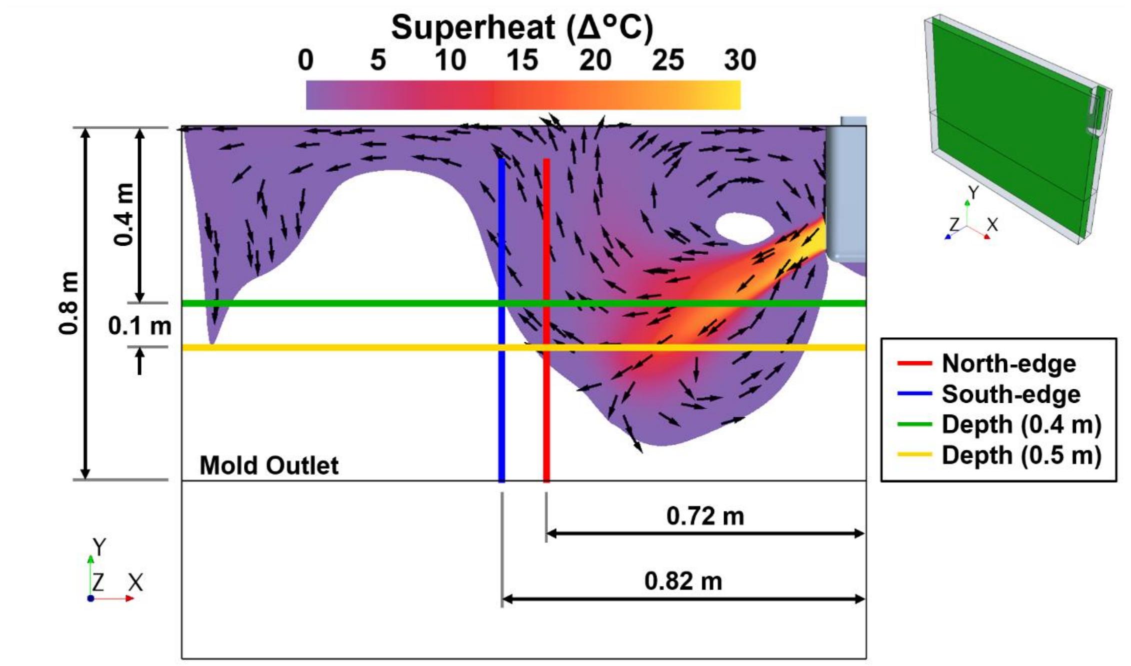


Figure 3.4. Superheat distribution with respect to the North and South edges of the recovered shell segment, and depth in the mold.

From Figure 3.4, the downward-momentum of the jet can be seen to diminish, and then be redirected upwards – towards the meniscus – prior to reaching the North-edge of the recovered shell segment. As a result, the superheat distribution extends to a greater depth along the North-edge, as opposed to the South-edge. This difference can be observed in Figure 3.4, where the perimeter of the superheat only extends to a depth of around 0.4 m below the meniscus along the South-edge, whereas, it exceeds 0.5 m along the North-edge. Further, the difference in superheat delivery that was shown to extend from 0.4 m, to slightly below 0.5 m, coincides with the section of decreased growth rate found in Figure 3.1, and similarly, with the difference between the percent error trends previously noted in Figure 3.2. Therefore, it was reasoned that the simulated superheat distribution, relative to the location of the referenced sample-planes, was responsible for the increased error identified along the South edge.

The simulated shell growth, shown in Figure 3.1, can be said to overpredict the measured profiles for both the North and South-edges. By comparison with the superheat delivery presented in Figure 3.4, it can be argued that the horizontal positioning of the sampling planes may differ from the actual location where the segment had been recovered from within the mold. This reasoning stands as the IC had only been able to affirm that the segment had been recovered from the quarter-width location; however, a dimensioned offset could not be established. Due to this uncertainty, the location of the sampling-planes may not aptly represent the relative position of the recovered segment within the mold. With this understanding, it can be argued that by decreasing the offset of the reference planes from the SEN, then simulated shell growth for both the North and South-edges would demonstrate greater agreement with the measurement data.

Additional insight into the source of increased error along the South-edge can be gained from understanding that under normal operating conditions – with a casting speed of around 40 ipm – the jet is known to impinge upon the NF, regularly. This factor would imply that, during regular operation, the downward-momentum of the jet would continue past the South-edge; thereby, resulting in a similar reduction in the shell growth rate, as was beheld of the North-edge for depths below 0.4 m. Moreover, the jet-impingement on the NF can be expected to form an upper-roll that would roughly span the width of the BF between the SEN and the NF. Should such an upper-roll remain relatively stable, it would result in a more uniform distribution of the superheat throughout the mold. Thus, the shell growth could likewise be expected to be more uniform along the width of the BF. While the solidification model was found to overpredict the shell growth along the South-edge, the more uniform superheat distribution – expected from the normal casting speed – would likely produce a profile more analogous to the shell measurement data.

In reviewing the events leading up to the breakout occurrence, a sticker-alarm had resulted in the casting speed being reduced from the normal-operating rate of 40 ipm to a reduced speed of 20 ipm. The casting speed had been decreased over a time interval of around 6 seconds, and then operation had continued at the reduced speed for approximately 93 seconds before the breakout occurrence. The reduction of the casting speed would dictate that the velocity of the steel entering the SEN would be halved from more than1.8 mps, down to just over 0.9 mps. This reduction of the flow rate through the SEN would decrease the momentum of the exiting-jets, which ultimately drive the formation of the roll-patterns.

Subsequently, as the decreased momentum of the jet would no longer be able to sustain the normal state of the flow field, it can be reasoned that the flow field would undergo a period of transition as it conforms to the new conditions. Just as the upper-roll formation could be presumed to stretch from the SEN to the NF under normal casting conditions, the upper-roll resulting from the reduced casting speed can be expected to be relatively smaller in size. A general profile of this upper-roll formation can be recognized in Figure 3.4, falling between the North-edge of the recovered shell segment and the SEN, and being located directly above the exiting-jet. Should the upper-roll remain intact during this transition period, it would progressively retract towards the SEN as angular momentum is sapped from the roll, until it was able to attain a profile comparable to that shown in Figure 3.4. In such a scenario, it would be necessary for the angular momentum of the roll to decline at a rate sufficient to prevent an abrupt dispersal of the roll.

Unfortunately, while the reduction of the casting speed had occurred in around 6 seconds, it is unclear what duration of time would be necessary for the flow field to attain a new pseudo-steady state. Assuming that the rolls would be retained, this time interval would be solely governed by the rate of momentum decay from the roll formations. As such, under optimal conditions, it can be reasoned that the transition period might extend for a prolonged duration of time beyond the completion of the casting speed reduction. By considering the above factors, it can be argued that the flow field, and therefore, the shell growth, may yet still have been in the process of transitioning to the new flow conditions. Should this be the case with the system discussed in this study, then the flow field could not be assumed to have achieved a pseudo-steady state.

Consequently, if the steady-state assumption is found to be inadequate, then it would be necessary for a transient simulation of the system to be conducted so as to replicate the system more-accurately. In order to conduct such a transient simulation, appropriate BC's would need to be defined so that an adequate rendition of the flow field transition, resulting from the casting speed reduction, might be produced. To do so would likely involve initializing the simulation having a fully-developed flow field while operating under the conditions which existed prior to the sticker-alarm. In defining the BC's for the transient simulation, the simulation should attempt to replicate three primary periods of operation:

1. the *normal operation period* that existed prior to the sticker alarm
2. the transition period from which the casting speed was decreased
3. the reduced operation period where the caster continuously operated at the reduced casting speed

Finally, the simulation results obtained using the described solidification model employed three assumptions of the material properties that should be addressed in future work. The first being that the material properties utilized in the simulation were those referenced from literature, and may not adequately represent those of the actual composition that was being cast [25]. The second assumption involved treating the solid fraction curve to be a linear relation with temperature, while the third concerns the assumption that the material properties could be defined as constants when they are known to be temperature-dependent. As the solid fraction is expected to influence the distribution of the momentum, energy, and turbulence transport properties within the MZ, designating a linear relation for the solid fraction could be expected to impact the overall distribution of the simulated MZ, and thereby, the shell development. Similarly, the resulting values of density, thermal conductivity, specific heat, and viscosity that would be evaluated in the transport relations would differ from the actual conditions by treating them as constants. Therefore, it is advised that the temperature-dependent material properties for the considered composition should be incorporated into the model for further studies.

3.2 Adaptive Mesh Refinement

Upon studying the results from the AMR POC simulation it was determined that the CA was successfully able to manage the AMR procedure throughout the simulation. It should be noted that while the simulation is being conducted, the order-of-execution of the AMR-CT procedures is largely governed by the STAR-CCM+ software. As a result, any UDF’s employed within the procedure will be evaluated for every cell within the domain, prior to the start of the AMR-CT, rather than only for the test-front cells. However, this does not impede the operation of the AMR-CT, as the values determined from these UDF's are temporarily recorded within the simulation file, and can therefore, be accessed as needed during the AMR-CT. By order-of-execution within the simulation, the first step in this procedure entails distinguishing the different cell types within the domain, as shown in Figure 3.5.

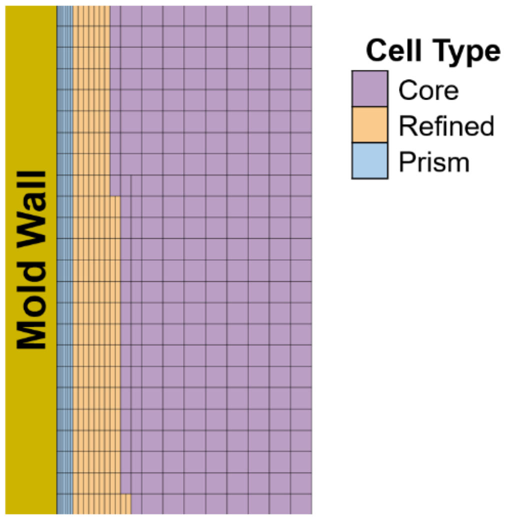


Figure 3.5. AMR procedure step 1: distinguish the different cell types within the domain.

Once the cell types have been identified, the next step in the sequence involves excluding cells – that will never require additional refinement – from further consideration. This predominantly includes the prism layer cells along the BF and NF boundaries, as well as the initial two-to-three rows of refined cells directly adjacent to the prism layers. These are excluded to ensure sufficient discretization of the domain for the temperature profile to be sufficiently evaluated. An example of this exclusion step can be visualized in Figure 3.6.

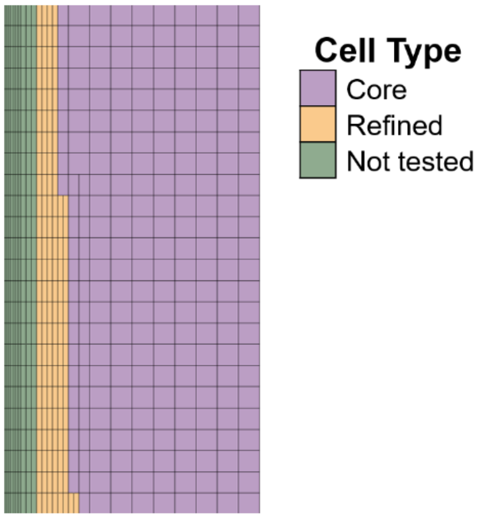


Figure 3.6. AMR procedure step 2: define the range limits for the cells considered for criteria testing.

Once the unnecessary cells have been removed from consideration, the following step involves identifying the test front cells. This is performed by generating two isothermal surfaces adjacent to the RP. To account for the increased sensitivity on the refined side of the RP, the LTF isosurface should be offset slightly further from the RP than the STF isosurface. With the test front isosurfaces created, all of the cells contacting the LTF isosurface or the STF isosurface should be identified – these represent the LTF and the STF cells for the refinement criteria tests to be conducted. The processes involved in this step of the refinement procedure are depicted in Figure 3.7.

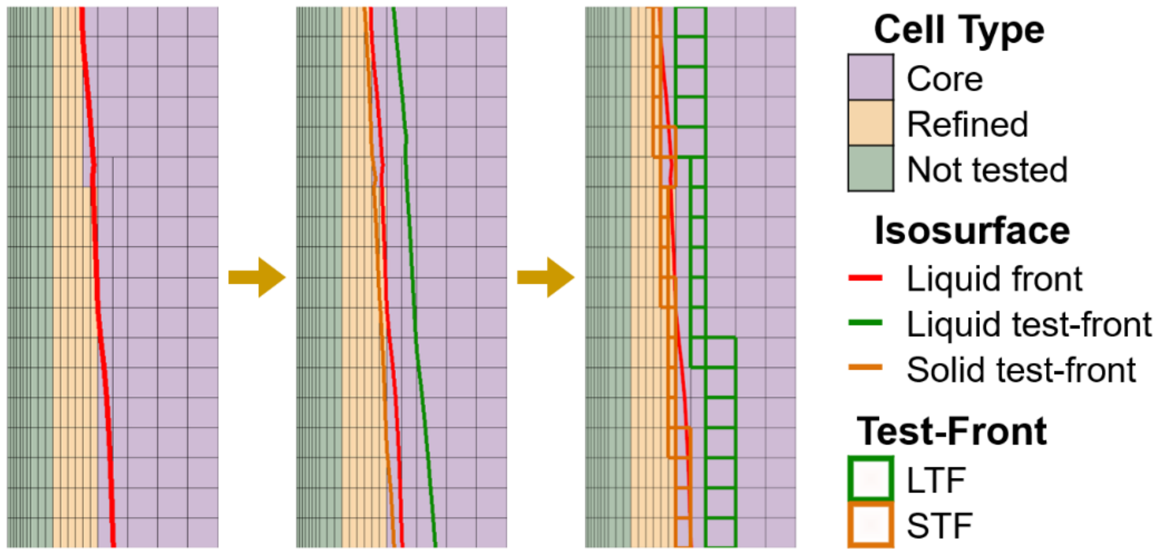


Figure 3.7. AMR procedure step 3: identifying the test-front cells.

Upon identifying the test-front cells, the respective criteria test can be conducted for each in their corresponding test-fronts. The cells identified to have failed the AMR-CT for their respective test-front, are highlighted in red in Figure 3.8 for visual reference.

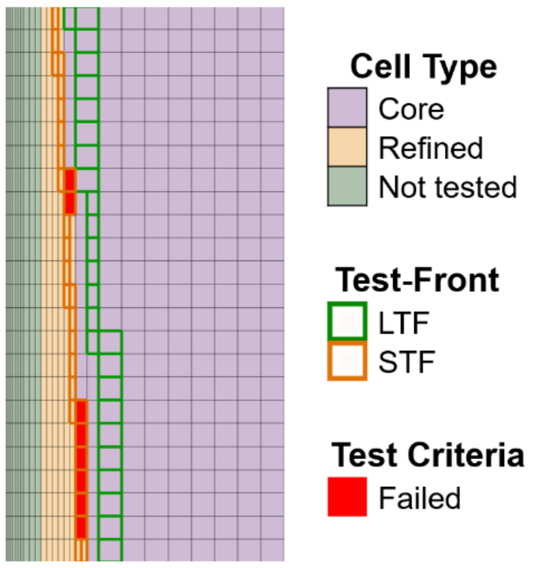


Figure 3.8. AMR procedure step 4: evaluate the corresponding test criteria for the cells in each test-front.

After the criteria tests have been performed, the final step in the AMR criteria testing procedure is to evaluate the ratio of failed-tests from each test-front against the total number of cells in each test-front, against a specified upper-threshold value. Should either of the test-front RFC exceed their designated threshold then the AMR procedure will be triggered. Upon the completion of the AMR mesh pipeline, the resulting mesh should be such that none of the test-front cells fail the AMR-CT for either test-front, as shown in Figure 3.9.

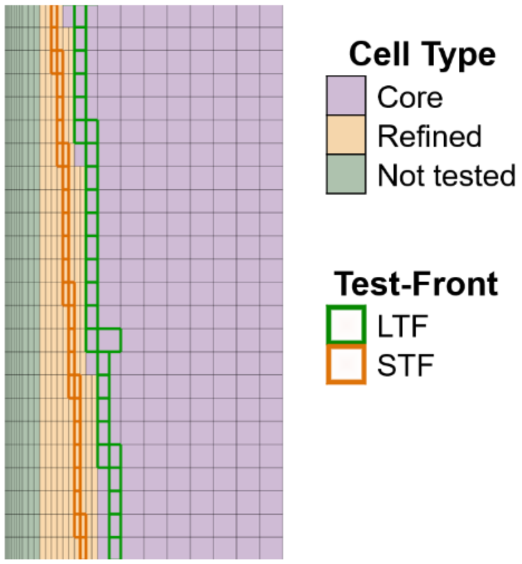


Figure 3.9. AMR procedure step 5: execute the local remeshing procedures.

The CA was developed to manage the entire refinement procedures during transient simulations. This permits the simulation to execute continuously – without the need for user input to conduct the AMR-CT or to execute the AMR mesh pipeline. The final results obtained from the simulations conducted both with, and without AMR demonstrate similar trends in the predicted shell development throughout the simulation. A comparison of the results are presented in Figure 3.10.

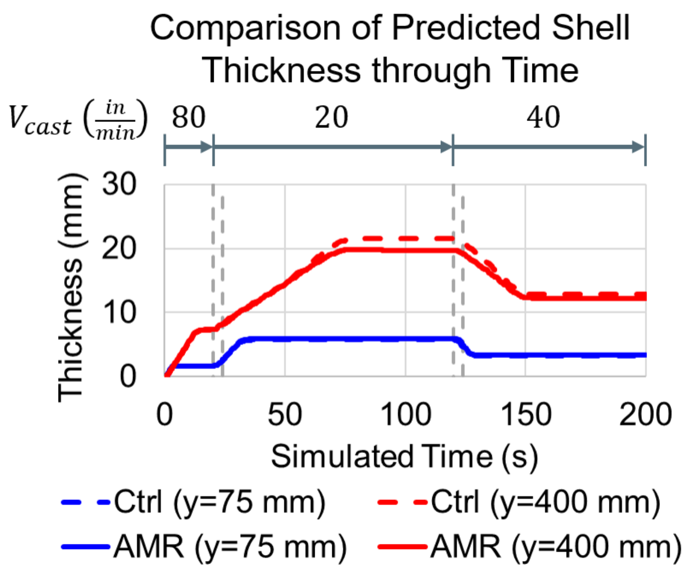


Figure 3.10. Results comparison for simulated shell thickness at 75 mm and 400 mm distances along the cooled surface.

Finally, the average percent difference between the two profiles observed at a distance of 400 mm was found to be approximately 6%, while the AMR simulation demonstrated roughly an 11% decrease in the overall simulation time.

4. CONCLUSIONS

4.1 Heat Transfer and Solidification Model

A 3D CFD model was generated to replicate the internal flow conditions that existed within a CC, prior to a breakout occurrence, to allow for heat transfer and solidification models to be developed and validated against physical measurement data obtained from a segment of the recovered shell. Three steady-state simulations were conducted in a performance and sensitivity study of the domain-length, using a similar meshing scheme for each of the domains. Each simulation was performed using 300 processors and required between 9.5 and 30-hours to complete. Deviations between the predicted shell growths of these simulations were found to be negligible for the relative elevation-range of the recorded shell thickness measurements. Therefore, the shortest domain was selected for use in further studies to reduce the total computational requirements necessary to conduct a simulation.

The caster being modeled was assumed to have reached a pseudo-steady state while operating at the reduced speed. Casting parameters and thermocouple measurement data were obtained from the IC and were utilized to generate time-averaged BC's representative of the assumed pseudo-steady operating conditions. The BC's were applied for both a steady-state and transient simulation using the shortened domain length, and a negligible difference was found between the results of each simulation. The predicted shell growth was found to match closely with the obtained shell thickness measurements at the quarter-width location believed to represent the recovered shell segment, with an average percent error of around 11%, and approximately a 1% difference between the error produced by the steady-state and transient simulations. From this, it was assumed that the use of either to simulate a pseudo-steady casting system would yield similar results.

Uncertainty of the position where the shell segment had been recovered from within the mold is believed to have attributed to increased error witnessed near the meniscus, as well as along the lower South-edge. From conceptual deliberation of the state of the system, including further consideration of the thermocouple data, it could be argued that the system had been unable to reach a pseudo-steady state during the time the caster had operated at a reduced rate – leading up to the breakout occurrence. Should this be the case, then it may be necessary for an additional transient simulation to be conducted – utilizing the measurement data to drive time-varying BC rather than time-averaged values. Such a simulation would be intended to determine whether the solidification model would be capable of replicating the same results if the shell profile were initially at a greater thickness while following more closely with the time-varying heat extraction rates recorded. Hence, one additional source of error may stem from the use of the steady-state assumption. Therefore, it was suggested that further consideration be given towards conducting a transient simulation, using time-varying BC's, to account for the transition in the flow pattern development after the casting speed was reduced.

Finally, the simulation results obtained using the described solidification model employed three assumptions of the material properties that should be addressed in future work. The first being that the material properties utilized in the simulation were those referenced from literature, and may not be representative of the actual property values existing during the considered operation. The second assumption pertains to the treatment of the material properties as constants when they are known to be temperature-dependent, and could potentially influence the predicted shell development. Third, the material properties were referenced from a literature source, and may not adequately represent those of the actual composition that was being cast [25]. As the solid fraction is expected to influence the distribution of the momentum, energy, and turbulence transport properties within the MZ, the solid fraction was assumed to have a linear relation with temperature. Therefore, it is recommended that temperature-dependent material properties for the considered composition should be incorporated into the model for further studies.

4.2 Adaptive Mesh Refinement

A set of procedures were established for utilizing the current solution data to locally apply additional mesh refinement in areas of predicted shell growth, while simultaneously reducing refinement in regions of the domain where it is no longer necessary. AMR-CT’s were then developed to periodically evaluate the active mesh against the current solution to determine the validity of the mesh. The full workflow involved with the AMR-CT’s and meshing procedures was then developed into a CA, and was incorporated into a Java-macro in order to automate the entire process during an individual simulation. The CA was tested through a simple POC-test to verify whether the AMR-CT’s would be capable of tracking the RP under transient conditions resulting in both increased and diminished growth of the simulated shell profile, while still being capable of decreasing the overall time required for the simulation to complete.

The results of the POC-tests revealed that the CA was capable of tracking the displacement of the RP during a simulation. It was also found that the total simulation time could be reduced by approximately 11% through the use of AMR. As the time-savings can be expected to increase with larger domains and higher-cell-counts, it is believed that incorporating AMR into a full-scale simulation of an actual CC would produce greater time-savings than those witnessed in this study.

Comparison of the simulated shell growth produced by the control and AMR-simulations showed a difference in the predicted shell thickness with an average value of around 6% at the domain outlet. While the profile produced by the AMR simulation was expected to match more-closely with the control-results, it can be argued that the profile obtained from the control simulation may not be representative of the actual shell growth that would develop for the simulated conditions. Therefore, it is recommended that AMR be incorporated into a full-scale CC-model for further evaluation of the CA performance to allow for the simulated shell profiles to be compared against shell-measurement data.

APPENDIX

Adaptive Mesh Refinement Control Algorithm

The following code entries comprise of a single java class that can be run as a macro in the STAR-CCM+ simulation software.

```
// STAR-CCM+ macro: adaptiveMesh_v00.java
package adaptiveMesh;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.time.LocalDateTime;
import java.util.*;

import star.common.*;
import star.base.neo.*;
import star.base.report.ElementCountReport;
import star.base.report.ExpressionReport;
import star.base.report.ReportManager;
import star.base.report.ReportMonitor;
import star.base.report.SumReport;
import star.cadmodeler.SolidModelPart;
import star.mapping.ProximityInterpolationModel;
import star.mapping.SolutionInterpolationMethod;
import star.meshing.*;
import star.surfacewrapper.*;
import star.vis.*;
import star.vis.Scene;
import star.vis.SceneUpdate;

/**
 * This java class is intended to be used as a simulation-macro in STAR-CCM+ to govern the
 * execution of steady-state & transient simulations of a continuous caster while implementing
 * AMR to provide increased resolution in areas of shell growth while minimizing the overall
 * cell count in order to reduce the total computation costs of the simulation.
 *
 * AMR-CT are performed at controlled intervals throughout the simulation and vary depending
 * on the relative state of the simulation and the recent procedures performed.
 *
 * ALL user-variables necessary for managing the performance of the macro execution are
 * located at the beginning of the class under the heading: 'USER-INPUT VARIABLES'.
 *
 * @author Matthew T. Moore
 * @version 00
 */

public class adaptiveMesh_v00 extends StarMacro {

//*--------------------------------------------------------------------------------------------------------------*//
//*-----------------------------------* USER-INPUT VARIABLES *-----------------------------------*//
//*--------------------------------------------------------------------------------------------------------------*//
// STEADY-STATE SETTINGS
   private final int maxIters = 4000;    // Max number of iterations for steady-state simulations
   private int fixedIters_Initial = 400;    // Fixed number of iterations for initial RUN
   private final int fixedIters_Meshed = 100;    // Min number of iterations between AM
            refinements
   private final boolean use_StopCrit_SteadyShellGrowth = false;    // Stop simulation when shell
            growth has reached a steady-state (( Steady-state only ))
// TRANSIENT SETTINGS
   private final double maxTime = 200.0;    // Max simulation time (seconds)
   private final double timeStep = 0.05;    // Simulated time-step (seconds)
   private int innerIters_Initial = 400;    // Min inner iterations during first time-step
   private final int innerIters_Meshed = 50;    // Min inner iterations AFTER mesh refinement
   private final int innerIters_Normal = 50;    // Max inner iterations without refinement
            procedure

// AM PARAMETERS
   private final double solidFraction_SolidTestFront = 0.15;    // Solid fraction used to designate
            the 'Test-Front' used for evaluating the AM_Shell Refine Criteria
   private final boolean use_ParallelMesher = false;    // Boolean: parallel meshing should be
            used (( only active when mesh cell-count > 100,000 cells ))
   // AM ACTIVATION METHODS
   // Only one activation method can be used at a time - if multiple are set to 'true' at RUN then
            the Shell Thickness Activation will be used
   private boolean use_ActivateAM_ShellThick = true;    // Boolean: 'Shell-Thickness' AM
            Activation method should be used
   private boolean use_ActivateAM_Temp = false;    // Boolean: 'Temperature-Range' AM
            Activation method should be used

// AUTO-SAVE SETTINGS
   private final int autoSave_nDigits = 7;    // Number of characters in the iteration/sim-time
            extension attached to the end of the Auto-Saved filename
   private final int autoSave_MaxFiles = 3;    // Maximum number of auto-saved files
   private String autoSave_Type = "TIMESTEP";    // Auto-save trigger type (Options:
            "ITERATION", "TIMESTEP", "DELTATIME", "EVENT")
```

```
    private final int autoSave_Frequency_Iters = 1000;    // Number of iterations between auto-
            saves ("ITERATION" trigger type)
    private final int autoSave_Frequency_TimeSteps = 20;    // Number of time-steps between
            auto-saves ("TIMESTEP" trigger type)
    private final double autoSave_Frequency_DeltaTime = 0.5;    // Simulated time-interval
            between auto-saves ("DELTATIME" trigger type)


//*----------------------------------------------------------------------------------------------------------*//
//*-----------------------* INSTANTIATE SIMULATION VARIABLES *------------------------*//
//*----------------------------------------------------------------------------------------------------------*//
    private Simulation simulation_0;
    private AutoSave autoSave_0;
    private StarLog starLog_0;
    private Units units_s;
    private Solution solution_0;
    private SimulationIterator iterator_0;

    private MonitorIterationStoppingCriterion stopCrit_ActivateAM_ShellThick;    // Stopping
            Criterion: indicates if shell thickness will support AM Refinement
    private MonitorIterationStoppingCriterion stopCrit_ActivateAM_Temp;    // Stopping
            Criterion: indicates if temperature profile will support AM Refinement (0: OFF, 1:
            ON)
    private MonitorIterationStoppingCriterion stopCrit_AMCoreRemesh;    // Stopping Criterion:
            ratio-threshold of liquid-test-front cells meeting the core-remesh criteria
    private MonitorIterationStoppingCriterion stopCrit_AMShellRefine;    // Stopping Criterion:
            ratio-threshold of solid-test-front cells meeting the shell-refine criteria
    private MonitorIterationStoppingCriterion stopCrit_Continuity;    // Stopping Criterion:
            Continuity
    private MonitorIterationStoppingCriterion stopCrit_Energy;    // Stopping Criterion: Energy
    private FixedStepsStoppingCriterion stopCrit_FixedSteps;    // Stopping Criterion: fixed steps
    private FixedPhysicalTimeStoppingCriterion stopCrit_FixedTime;    // Stopping Criterion:
            fixed time
    private InnerIterationStoppingCriterion stopCrit_MaxInnerIter;    // Stopping Criterion:
            maximum inner iterations
    private StepStoppingCriterion stopCrit_MaxIters;    // Stopping Criterion: maximum number
            of iterations
    private PhysicalTimeStoppingCriterion stopCrit_MaxTime;    // Stopping Criterion: maximum
            physical time
    private MinimumInnerIterationStoppingCriterion stopCrit_MinInnerIter;    // Stopping
            Criterion: minimum inner iterations
    private MonitorIterationStoppingCriterion stopCrit_Sdr;    // Stopping Criterion: Specific
            Dissipation Rate
    private MonitorIterationStoppingCriterion stopCrit_SteadyShellGrowth;    // Stopping
            Criterion: indicates that the shell development has reached a steady-state
```

```
private MonitorIterationStoppingCriterion stopCrit_Tke;    // Stopping Criterion: Turbulent
        Kinetic Energy
private MonitorIterationStoppingCriterion stopCrit_Xmomentum;    // Stopping Criterion: X-
        momentum
private MonitorIterationStoppingCriterion stopCrit_Ymomentum;    // Stopping Criterion: Y-
        momentum
private MonitorIterationStoppingCriterion stopCrit_Zmomentum;    // Stopping Criterion: Z-
        momentum

private ReportManager manager_Report;    // Simulation Report Manager
private ExpressionReport report_ActivateAM_ShellThick;    // Report: indicates if shell
        thickness will support AM Refinement
private ExpressionReport report_ActivateAM_Temp;    // Report: indicates if temperature
        profile will support AM Refinement (0: OFF, 1: ON)
private PhysicsContinuumIterationReport report_Iteration;    // Report: simulated iterations
private ExpressionReport report_LTF_CellRatio;    // Report: ratio of the liquid-test-front cells
        meeting the core-remesh criteria
private ExpressionReport report_STF_CellRatio;    // Report: ratio of the solid-test-front cells
        meeting the shell-refine criteria

private int value_CurrentIteration;    // Report value: current iteration

private ScalarGlobalParameter GP_AMCoreRemesh_StopCritLimit;    // Global Parameter:
        upper limit for the Core Remesh Stopping Criterion
private ScalarGlobalParameter GP_AMShellRefine_StopCritLimit;    // Global Parameter:
        upper limit for the Shell Refine Stopping Criterion
private ScalarGlobalParameter GP_AMSolidTestFront_SolidFraction;    // Global Parameter:
        solid fraction used to define the 'Solid Test-Front'

private double value_AMCoreRemesh_StopCritLimit;    // Global Parameter: value of the
        upper limit for the Core Remesh Stopping Criterion
private double value_AMShellRefine_StopCritLimit;    // Global Parameter: value of the upper
        limit for the Shell Refine Stopping Criterion

private String dirName;    // Active directory location of sim file
private String simName;    // Active sim filename
private String simBasename;    // Base filename of active sim
private String simNameDescriptor;    // Filename descriptor attached in filename (follows
        basename)
private String opDescriptor;    // Operation descriptor attached to end of filename
private final String sep = System.getProperty("file.separator");    // system file separator

private String name_TimeModel;    // Identified type of Time-Model active in sim ("steady" or
        "implicit")
private String stopCrit_UpdateType;    // String indicating the Stop. Crit. settings to be used
        ("initial", "meshed", or "normal")
```

```
private boolean statusRUN = true;    // Boolean: macro execution should continue
private boolean statusActiveAM = false;    // Boolean: sufficient iterations have been
        conducted for the AM Procedure to be performed
private boolean statusInitialize = true;    // Boolean: sim should be Initialized before 'initial'
        RUN
private boolean statusInitialRUN = true;    // Boolean: 'Initial-Run' Stopping Criteria should be
        applied before RUN (( used for very first steady/transient RUN ))
private boolean statusERROR = false;    // Boolean: unrecoverable error has occurred and been
        recorded in the simulation log
private boolean statusNewMeshRUN = false;    // Boolean: 'New-Mesh' Stopping Criteria
        should be applied before RUN (( used for first RUN following an AM Remesh
        procedure ))
private int statusRemesh = 0;    // Integer-test: domain should be remeshed instead of RUN
        during the next WHILE-iteration (false=0,true=1:inf+)

private boolean test_Report_ActivateAM_ShellThick = false;    // (Report) Boolean: shell
        thickness will support AM Refinement
private boolean test_Report_ActivateAM_Temp = false;    // (Report) Boolean: temperature
        profile will support AM Refinement
private boolean test_Report_AMRefine = false;    // (Report) Boolean: AM_Refinement
        Criteria Reports are satisfied
private boolean test_StopCrit_ActivateAM_ShellThick = false;    // (StopCrit) Boolean: shell
        thickness will support AM Refinement
private boolean test_StopCrit_ActivateAM_Temp = false;    // (StopCrit) Boolean:
        temperature profile will support AM Refinement
private boolean test_StopCrit_AMRefine = false;    // (StopCrit) Boolean: AM_Refinement
        Criteria Stopping Criteria are satisfied
private boolean test_StopCrit_FixedSteps = false;    // (StopCrit) Boolean: Fixed-Steps
        Stopping Criteria is satisfied
private boolean test_StopCrit_FixedTime = false;    // (StopCrit) Boolean: Fixed-Physical-
        Time Stopping Criteria is satisfied
private boolean test_StopCrit_MaxIters = false;    // (StopCrit) Boolean: Maximum-Iterations
        Stopping Criteria is satisfied
private boolean test_StopCrit_MaxTime = false;    // (StopCrit) Boolean: Maximum-Physical-
        Time Stopping Criteria is satisfied
private boolean test_StopCrit_RESIDS = false;    // (StopCrit) Boolean: Stopping Criteria for
        the Residuals is satisfied
private boolean test_StopCrit_SteadyShellGrowth = false;    // (StopCrit) Boolean: shell
        development has reached a steady-state

private int n_Mesh = 0;    // Number of mesh constructions performed during macro procedure
```

```
//*-------------------------------------------------------------------------------------------------------------*//
//*---------------------------------------* AMR PROCEDURE *-----------------------------------------*//
//*-------------------------------------------------------------------------------------------------------------*//
   @Override
   public void execute() {
      simulation_0 = getActiveSimulation();

      simVars();    // Initialize simulation variables
      startUpOperations();

      if (statusRUN) {
         autoSave();

         if (statusInitialize) {
            if (opDescriptor.contains("@mesh")) {
               simulation_0.println("    '@mesh' descriptor identified at end of simulation
                        filename...\n    --> Mesh Pipeline will be executed with 'initial mesh' settings
                        prior to initializing solution.");
               meshPipeline();
            }

            initializeSim();
         }

         stopCritTests();
         fileOps();
      }

   // LOOP OPERATIONS
      while (statusRUN) {
         switch (statusRemesh) {
            case 0:
               switch (stopCrit_UpdateType) {
                  case "initial":
                     initialRUN();
                     break;
                  case "amInactive":
                     amInactiveRUN();
                     break;
                  case "meshed":
                     meshedRUN();
                     break;
                  case "normal":
                     normalRUN();
                     break;
               }
```

```
                break;
            default:
                meshPipeline();
                break;
        }

        stopCritTests();
        fileOps();
    }

    simulation_0.println("\n\nMacro execution complete!!!");
}


//*------------------------------------------------------------------------------------------------------------------*//
//*--------------------------* INITIALIZE SIMULATION VARIABLES *---------------------------*//
//*------------------------------------------------------------------------------------------------------------------*//
  private void simVars() {
    simulation_0.println("System Type:    " + System.getProperty("os.name"));
    simulation_0.println("\nInitializing simulation-variables...");
    solution_0 = simulation_0.getSolution();
    iterator_0 = simulation_0.getSimulationIterator();

// SIMULATION TIME-MODEL
    name_TimeModel =
            iterator_0.getRunnableSolver().getMenuPresentationName().toLowerCase()
            .split(" ",0)[0];
    simulation_0.println("       Solver Time-Model (Solver):    '" + name_TimeModel + "'");

// UNITS
    units_s = simulation_0.getUnitsManager().getObject("s");

// AUTO-SAVE
    autoSave_0 = iterator_0.getAutoSave();
    autoSave_Type = (( name_TimeModel.equals("steady") &&
            ( autoSave_Type.toUpperCase().equals("TIMESTEP") ||
            autoSave_Type.toUpperCase().equals("DELTATIME") ) ) ? "ITERATION" :
            autoSave_Type.toUpperCase());

// SIMULATION LOG
    starLog_0 = simulation_0.getStarLog();

// STOPPING CRITERIA
    stopCrit_ActivateAM_ShellThick = ((MonitorIterationStoppingCriterion)
            simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion(
            "AM_Activation_Shell Thickness Criterion"));
```

```
stopCrit_ActivateAM_Temp = ((MonitorIterationStoppingCriterion)
        simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion(
        "AM_Activation_Temperature Criterion"));
stopCrit_AMCoreRemesh = ((MonitorIterationStoppingCriterion)
        simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion(
        "AM_Core Remesh Criterion"));
stopCrit_AMShellRefine = ((MonitorIterationStoppingCriterion)
        simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion(
        "AM_Shell Refine Criterion"));
stopCrit_Continuity = ((MonitorIterationStoppingCriterion)
        simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion(
        "Continuity Criterion"));
stopCrit_Energy = ((MonitorIterationStoppingCriterion)
        simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion(
        "Energy Criterion"));
stopCrit_FixedSteps = ((FixedStepsStoppingCriterion)
        simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion(
        "Fixed Steps"));
stopCrit_FixedTime = ((FixedPhysicalTimeStoppingCriterion)
        simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion(
        "Fixed Physical Time"));
stopCrit_MaxInnerIter = ((InnerIterationStoppingCriterion)
        simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion(
        "Maximum Inner Iterations"));
stopCrit_MaxIters = ((StepStoppingCriterion)
        simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion(
        "Maximum Steps"));
stopCrit_MaxTime = ((PhysicalTimeStoppingCriterion)
        simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion(
        "Maximum Physical Time"));
stopCrit_MinInnerIter = ((MinimumInnerIterationStoppingCriterion)
        simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion(
        "Minimum Inner Iterations"));
stopCrit_Sdr = ((MonitorIterationStoppingCriterion)
        simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion(
        "Sdr Criterion"));
stopCrit_SteadyShellGrowth = ((MonitorIterationStoppingCriterion)
        simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion(
        "AM_Steady-State Shell Growth Criterion"));
stopCrit_Tke = ((MonitorIterationStoppingCriterion)
        simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion(
        "Tke Criterion"));
stopCrit_Xmomentum = ((MonitorIterationStoppingCriterion)
        simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion(
        "X-momentum Criterion"));
        stopCrit_Ymomentum = ((MonitorIterationStoppingCriterion)
                simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion(
                "Y-momentum Criterion"));
        stopCrit_Zmomentum = ((MonitorIterationStoppingCriterion)
                simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion(
                "Z-momentum Criterion"));

    // REPORTS
        manager_Report = simulation_0.getReportManager();
      // ITERATION
        int count_iterReports =
                manager_Report.getObjectsOf(PhysicsContinuumIterationReport.class).size();
        if (count_iterReports < 1) {
          simulation_0.println("\nNo existing Iteration Reports identified...\nGenerating new "
                  + "Iteration Report...");
          report_Iteration =
                  manager_Report.createReport(PhysicsContinuumIterationReport.class);
        } else {
          report_Iteration =
                  manager_Report.getObjectsOf(PhysicsContinuumIterationReport.class).get(0);
        }
        value_CurrentIteration = report_Iteration.getValue();
      // AM ACTIVATION: SHELL THICKNESS
        report_ActivateAM_ShellThick = ((ExpressionReport)
                manager_Report.getReport("Expression - Max Shell Thickness_NF1"));
      // AM ACTIVATION: TEMPERATURE
        report_ActivateAM_Temp = ((ExpressionReport) manager_Report.getReport("Expression"
                + " - AM_Activation_Temperature"));
      // CELL-COUNT RATIOS
        report_LTF_CellRatio =    ((ExpressionReport) manager_Report.getReport("Expression - "
                + "AM_Core Remesh Fraction"));
        report_STF_CellRatio = ((ExpressionReport) manager_Report.getReport("Expression - "
                + "AM_Shell Refine Fraction"));

    // GLOBAL PARAMETERS
        GP_AMCoreRemesh_StopCritLimit = ((ScalarGlobalParameter)
                simulation_0.get(GlobalParameterManager.class).getObject("AM_CoreRemesh_"
                + "StopCritLimit"));
        GP_AMShellRefine_StopCritLimit = ((ScalarGlobalParameter)
                simulation_0.get(GlobalParameterManager.class).getObject("AM_ShellRefine_"
                + "StopCritLimit"));
        GP_AMSolidTestFront_SolidFraction = ((ScalarGlobalParameter)
                simulation_0.get(GlobalParameterManager.class).getObject("AM_SolidTestFront"
                + "_SolidFraction"));
        GP_AMSolidTestFront_SolidFraction.getQuantity().setValue(
                solidFraction_SolidTestFront);
        // Global Parameters: Values
        value_AMCoreRemesh_StopCritLimit =
                    Double.valueOf(GP_AMCoreRemesh_StopCritLimit.getExpression().toString());
                    // AM_CORE REMESH STOPPING CRITERION LIMIT
        value_AMShellRefine_StopCritLimit =
                    Double.valueOf(GP_AMShellRefine_StopCritLimit.getExpression().toString());
                    // AM_SHELL REFINE STOPPING CRITERION LIMIT

// AM ACTIVATION
        if (( ((use_ActivateAM_ShellThick) ? 1 : 0) + ((use_ActivateAM_Temp) ? 1 : 0)  ) > 1) {
          use_ActivateAM_ShellThick = true;
          use_ActivateAM_Temp = false;
        }

// FILE/DIRECTORY INFORMATION
        dirName = simulation_0.getSessionDir();
        simName = simulation_0.getPresentationName();
        simBasename = simName.split("@",0)[0];
        simNameDescriptor = ((simName.split("@",0).length > 1) ?
                    simName.substring(simName.indexOf("@")) : "");
        opDescriptor = ((simName.split("@",0).length > 1) ?
                    simName.substring(simName.lastIndexOf("@")) : "");

        simulation_0.println("    FILE/DIRECTORY INFORMATION");
        simulation_0.println("       dirName:                    " + dirName);
        simulation_0.println("       simName:                    " + simName);
        simulation_0.println("       simBasename:                " + simBasename);
        simulation_0.println("       simNameDescriptor:          " + simNameDescriptor);
    }


//*-------------------------------------------------------------------------------------------------------*//
//*-----------------------------* MACRO START-UP OPERATIONS *------------------------------*//
//*-------------------------------------------------------------------------------------------------------*//
    private void startUpOperations() {
        simulation_0.println("\nPerforming macro start-up operations...");

        test_StopCrit_MaxIters = ( name_TimeModel.equals("steady") &&
                    (iterator_0.getCurrentIteration() >= maxIters) );
        test_StopCrit_MaxTime = ( name_TimeModel.equals("implicit") &&
                    ((iterator_0.getCurrentTimeLevel()*timeStep) >= maxTime) );

// STATUS TEST: EXISTING ERROR
        simulation_0.println("    STATUS TEST:    EXISTING ERROR");
        if ( starLog_0.hasNonRecoverableError() || starLog_0.isSavedAfterError() ||
                    simName.contains("@ERROR") ) {
```

```
        simulation_0.println("\n\n    !!!!! WARNING !!!!!\n\n    An 'Error' occurrence was "
                + "detected to have occurred during a previous execution of this simulation file!"
                + "\n    --> Simulation will be terminated without attempting to RUN.");
        statusERROR = true;
        statusRUN = false;
        statusInitialize = false;

      } else {
// STATUS TEST: INITIALIZE SIM
        simulation_0.println("    STATUS TEST:    INITIALIZE-SIM");
        if (simName.matches(".+@initial.*")) {
          value_CurrentIteration = 0;
          test_StopCrit_MaxIters = false;
          test_StopCrit_MaxTime = false;
          simulation_0.println("    '@initialize' descriptor identified in simulation filename...\n"
                  + "--> Solution history will be cleared and simulation will be initialized before "
                  + "RUN.\n    --> '@initialize' Descriptor to be removed from filename.");

        } else {
          statusInitialize = false;
          fixedIters_Initial = fixedIters_Meshed;
          innerIters_Initial = innerIters_Normal;
          simulation_0.println("    '@initialize' descriptor not found in the simulation filename."
                  + "\n--> Simulation will be continued from existing solution.");

          if (test_StopCrit_MaxIters || test_StopCrit_MaxTime) {
            statusRUN = false;
            simulation_0.println("\n\n    !!!!! ALERT !!!!!\n\n    Current Iterations/Physical-"
                    + "Time exceeds upper-limit value specified by user for the Stopping "
                    + "Criteria."+ "\n    --> Simulation will not be able to continue with current"
                    + " settings.\n    --> Aborting simulation...");
          } else if ( simNameDescriptor.contains("@MaxIters") ||
                  simNameDescriptor.contains("@MaxTime") ) {
            simulation_0.println("\n\n    !!!!! ALERT !!!!!\n\n    The current filename contains"
                    + " either the '@MaxIters' or '@MaxTime' descriptor - indicating that the "
                    + "simulation met or exceeded the corresponding stopping criteria during"
                    + "\n     a previous RUN, however, that Stopping Criteria is no longer "
                    + "satisfied under the new settings.\n\n    --> The 'flagged' descriptor will "
                    + "be removed from the filename and the simulation will be RUN.");

          // SAVE SIMULATION WITH UPDATED FILENAME
            saveProgress();

          } else {
        // STATUS TEST: MESHED SIM
            simulation_0.println("\n    STATUS TEST:    MESHED-SIM");
```

```
                if (simName.endsWith("@meshed")) {
                    statusNewMeshRUN = true;
                    simulation_0.println("        'Meshed' descriptor identified in simulation filename"
                            + "\n        --> Simulation was last saved after mesh construction.");
                } else {
                    simulation_0.println("        'Meshed' descriptor not found in simulation "
                            + "filename.");
                }

                simulation_0.println("        statusExistingMesh:    " + statusNewMeshRUN);
            }
        }
    }
}


//*------------------------------------------------------------------------------------------------------*//
//*-----------------------------------* AUTOSAVE SETTINGS *--------------------------------------*//
//*------------------------------------------------------------------------------------------------------*//
    private void autoSave() {
        simulation_0.println("\nApplying Auto-Save settings...");

        StarUpdate starUpdate_0 = autoSave_0.getStarUpdate();
        autoSave_0.setMaxAutosavedFiles(autoSave_MaxFiles);
        autoSave_0.setAutoSaveBatch(true);
        autoSave_0.setAutoSaveMesh(true);
        autoSave_0.setCheckpoint(false);
        autoSave_0.setCheckpointFile("CHECKPOINT");
        autoSave_0.setSeparator("@");
        autoSave_0.setFormatWidth(autoSave_nDigits);
        starUpdate_0.setEnabled(true);

        switch (autoSave_Type) {
            case "ITERATION":
                starUpdate_0.getUpdateModeOption().setSelected(StarUpdateModeOption
                        .Type.ITERATION);
                IterationUpdateFrequency iterationUpdateFrequency_0 =
                        starUpdate_0.getIterationUpdateFrequency();
                iterationUpdateFrequency_0.setIterations(autoSave_Frequency_Iters);
                iterationUpdateFrequency_0.setStart(0);
                break;
            case "TIMESTEP":
                starUpdate_0.getUpdateModeOption().setSelected(StarUpdateModeOption
                        .Type.TIMESTEP);
                TimeStepUpdateFrequency timeStepUpdateFrequency_0 =
                        starUpdate_0.getTimeStepUpdateFrequency();
                timeStepUpdateFrequency_0.setTimeSteps(autoSave_Frequency_TimeSteps);
                timeStepUpdateFrequency_0.setStart(0);
                break;
              case "DELTATIME":
                autoSave_0.setFormatWidth(autoSave_nDigits - 1);
                starUpdate_0.getUpdateModeOption().setSelected(StarUpdateModeOption
                        .Type.DELTATIME);
                DeltaTimeUpdateFrequency deltaTimeUpdateFrequency_0 =
                        starUpdate_0.getDeltaTimeUpdateFrequency();
                deltaTimeUpdateFrequency_0.setDeltaTime(String.valueOf(
                        autoSave_Frequency_DeltaTime), units_s);
                deltaTimeUpdateFrequency_0.setStopTime(0.0, units_s);
                deltaTimeUpdateFrequency_0.setStartTime(0.0, units_s);
                break;
        }
    }


//*------------------------------------------------------------------------------------------------------------*//
//*-----------------------------------* INITIALIZE SOLUTION *--------------------------------------*//
//*------------------------------------------------------------------------------------------------------------*//
    private void initializeSim() {
        simulation_0.println("\n\nInitializing simulation...");

// CLEAR/INITIALIZE SOLUTION
        solution_0.clearSolution();
        solution_0.initializeSolution();

// RESET VALUES
        value_CurrentIteration = 0;
        test_StopCrit_MaxIters = false;
        test_StopCrit_MaxTime = false;
        starLog_0.setLog("");

// SAVE 'RUN-STATE' FILE-VERSION
        simulation_0.println("\n    Updating simulation filename - all descriptors will be "
                + "removed.");
        simulation_0.saveState(dirName + sep + simBasename + ".sim");

    // UPDATE SIMNAME
        simName = simulation_0.getPresentationName();
        simNameDescriptor = ((simName.split("@",0).length > 1) ?
                simName.substring(simName.indexOf("@")) : "");
        opDescriptor = ((simName.split("@",0).length > 1) ?
                simName.substring(simName.lastIndexOf("@")) : "");
        statusInitialize = false;
        statusInitialRUN = true;
    }


//*---------------------------------------------------------------------------------------------------------*//
//*---------------------------------* STOPPING-CRITERIA TESTS *---------------------------------*//
//*---------------------------------------------------------------------------------------------------------*//
        private void stopCritTests() {
                simulation_0.println("\n\nConducting tests of current Stopping Criteria...");

// UPDATE STATISTICS OUTPUT-VALUES
                value_CurrentIteration = report_Iteration.getValue();

// REPORT 'ENABLED'/'SATISFIED' TESTS
                test_Report_ActivateAM_ShellThick = ( use_ActivateAM_ShellThick &&
                        (report_ActivateAM_ShellThick.getReportMonitorValue() > 0.0025) );
                test_Report_ActivateAM_Temp = ( use_ActivateAM_Temp &&
                        (report_ActivateAM_Temp.getReportMonitorValue() > 0.5) );
                test_Report_AMRefine = ( (report_LTF_CellRatio.getReportMonitorValue() >
                        value_AMCoreRemesh_StopCritLimit) ||
                        (report_STF_CellRatio.getReportMonitorValue() >
                        value_AMShellRefine_StopCritLimit) );

// STOPPING CRITERIA 'ENABLED'/'SATISFIED' TESTS
                Collection <SolverStoppingCriterion> stopCrit_Collection =
                        simulation_0.getSolverStoppingCriterionManager().getObjects();

                if (stopCrit_Collection.size() > 0) {
                        simulation_0.println("    IDENTIFIED 'TRIGGERED' STOPPING CRITERIA");
                        String[] stopCrit_SatisfiedNames = new String[stopCrit_Collection.size()];
                        int nSatisfied = -1;
                        for (SolverStoppingCriterion sc: stopCrit_Collection) {
                                if ( (sc.getBoolean("IsUsed")) && (sc.getBoolean("IsSatisfied")) ) {
                                        nSatisfied++;
                                        stopCrit_SatisfiedNames[nSatisfied] = sc.getMenuPresentationName();
                                        simulation_0.println("    " + sc.getMenuPresentationName());
                                }
                        }

        // 'TEST'-VARIABLES
                        test_StopCrit_ActivateAM_ShellThick = ( use_ActivateAM_ShellThick&&
                                Arrays.stream(stopCrit_SatisfiedNames).anyMatch(
                                "AM_Activation_Shell Thickness Criterion"::equals) );
                        test_StopCrit_ActivateAM_Temp = ( use_ActivateAM_Temp &&
                                Arrays.stream(stopCrit_SatisfiedNames).anyMatch(
                                "AM_Activation_Temperature Criterion"::equals) );
                        test_StopCrit_AMRefine = ( Arrays.stream(stopCrit_SatisfiedNames).anyMatch(
                                "AM_Core Remesh Criterion"::equals) ||
                                Arrays.stream(stopCrit_SatisfiedNames).anyMatch(
                                "AM_Shell Refine Criterion"::equals) );
                        test_StopCrit_FixedSteps = Arrays.stream(stopCrit_SatisfiedNames).anyMatch(
                                "Fixed Steps"::equals);
                        test_StopCrit_FixedTime = Arrays.stream(stopCrit_SatisfiedNames).anyMatch(
                                "Fixed Physical Time"::equals);
                        test_StopCrit_MaxIters = Arrays.stream(stopCrit_SatisfiedNames).anyMatch(
                                "Maximum Steps"::equals);
                        test_StopCrit_MaxTime = Arrays.stream(stopCrit_SatisfiedNames).anyMatch(
                                "Maximum Physical Time"::equals);
                        test_StopCrit_RESIDS = ( Arrays.stream(stopCrit_SatisfiedNames).anyMatch(
                                "Continuity Criterion"::equals)
                                && Arrays.stream(stopCrit_SatisfiedNames).anyMatch(
                                "X-momentum Criterion"::equals)
                                && Arrays.stream(stopCrit_SatisfiedNames).anyMatch(
                                "Y-momentum Criterion"::equals)
                                && Arrays.stream(stopCrit_SatisfiedNames).anyMatch(
                                "Z-momentum Criterion"::equals)
                                && Arrays.stream(stopCrit_SatisfiedNames).anyMatch(
                                "Tke Criterion"::equals)
                                && Arrays.stream(stopCrit_SatisfiedNames).anyMatch(
                                "Sdr Criterion"::equals)
                                && Arrays.stream(stopCrit_SatisfiedNames).anyMatch(
                                "Energy Criterion"::equals) );
                        test_StopCrit_SteadyShellGrowth = ( use_StopCrit_SteadyShellGrowth &&
                                Arrays.stream(stopCrit_SatisfiedNames).anyMatch(
                                "AM_Steady-State Shell Growth Criterion"::equals) );
                }

// OPERATION STATUS-CHECKS
                simulation_0.println("\n    Status-Update Checks");
                if ( test_StopCrit_MaxIters || test_StopCrit_MaxTime || ( test_StopCrit_RESIDS &&
                        name_TimeModel.equals("steady") ) || test_StopCrit_SteadyShellGrowth ) {
                        statusRUN = false;
                } else {
                        if ( !statusActiveAM ) {
                                statusInitialRUN = (!test_StopCrit_RESIDS && ( value_CurrentIteration <
                                        (( name_TimeModel.equals("steady") ) ? fixedIters_Initial
                                        : innerIters_Initial) ));

                                statusActiveAM = !statusInitialRUN && ( (use_ActivateAM_ShellThick)
                                        ? (test_StopCrit_ActivateAM_ShellThick
                                        || test_Report_ActivateAM_ShellThick) : ((use_ActivateAM_Temp)
                                        ? (test_StopCrit_ActivateAM_Temp
                                        || test_Report_ActivateAM_Temp) : true) );
                        }

        // REMESH STATUS-CHECK
                        if ( statusActiveAM && !statusNewMeshRUN && (test_StopCrit_AMRefine
                                || test_Report_AMRefine) ) {
                            statusRemesh++;
                        }

        // STOPPING CRITERIA UPDATE-TYPE
                        if ( statusInitialRUN ) {
                            stopCrit_UpdateType = "initial";
                        } else if (!statusActiveAM) {
                            stopCrit_UpdateType = "amInactive";
                        } else if (statusNewMeshRUN) {
                            stopCrit_UpdateType = "meshed";
                        } else {
                            stopCrit_UpdateType = "normal";
                        }

                        simulation_0.println("     statusRUN:                    " + statusRUN);
                        simulation_0.println("     statusInitialRUN:             " + statusInitialRUN);
                        simulation_0.println("     statusActiveAM:               " + statusInitialRUN);
                        simulation_0.println("     statusRemesh:                 " + statusRemesh);
                        simulation_0.println("     statusNewMeshRUN:             "
                                + statusNewMeshRUN);
                        simulation_0.println("     stopCrit_UpdateType:          " + stopCrit_UpdateType);
                }
        }


//*------------------------------------------------------------------------------------------------------------*//
//*-----------------------------------* FILE-TEST OPERATIONS *------------------------------------*//
//*------------------------------------------------------------------------------------------------------------*//
    private void fileOps() {    // To be performed immediately after 'stopCritTests()'
        simulation_0.println("\n\nEvaluating simulation file-version.");

        if ( statusERROR || starLog_0.hasNonRecoverableError() || starLog_0.isSavedAfterError()
                || simName.contains("@ERROR") ) {
            statusERROR = true;
            statusRUN = false;
            statusInitialize = false;
        } else if ((name_TimeModel.equals("steady"))
                ? simNameDescriptor.contains("@MaxIters")
                : simNameDescriptor.contains("@MaxTime")) {
```

```
            statusRUN = false;
        } else if (statusActiveAM) {
            if ( opDescriptor.matches("^@[0-9]+((\\.[0-9]+)?[e][-,+][0-9]{2})?$")
                        && !statusNewMeshRUN && (statusRemesh > 0) ) {
                saveFileVersion("@AMRefineTriggered");
            }
            switch (opDescriptor) {
                case "@AMRefineTriggered":
                    saveProgress();
                    break;
            }
        }

// UPDATE SIMULATION NAME
        simName = simulation_0.getPresentationName();
        simNameDescriptor = ((simName.split("@",0).length > 1)
                    ? simName.substring(simName.indexOf("@")) : "");
        opDescriptor = ((simName.split("@",0).length > 1)
                    ? simName.substring(simName.lastIndexOf("@")) : "");
    }


//*-----------------------------------------------------------------------------------------------------------*//
//*---------------------------------* INITIAL RUN PROCEDURE *-----------------------------------*//
//*-----------------------------------------------------------------------------------------------------------*//
    private void initialRUN() {
        simulation_0.println("\n\nINITIATING OPERATION:    'initialRUN()'");

// UPDATE STOPPING CRITERIA
        stopCrit_Continuity.setIsUsed(true);
        stopCrit_Energy.setIsUsed(true);
        stopCrit_Sdr.setIsUsed(true);
        stopCrit_SteadyShellGrowth.setIsUsed(false);
        stopCrit_Tke.setIsUsed(true);
        stopCrit_Xmomentum.setIsUsed(true);
        stopCrit_Ymomentum.setIsUsed(true);
        stopCrit_Zmomentum.setIsUsed(true);
        simulation_0.println("\nStopping Criteria Activation:\n    stopCrit_AMShellRefine:    true"
                    + "\n    stopCrit_Energy:            true\n    stopCrit_Sdr:                true"
                    + "\n    stopCrit_Xmomentum:        true\n    stopCrit_Ymomentum:        true"
                    + "\n    stopCrit_Zmomentum:        true");

    // AM CRITERION
        stopCrit_ActivateAM_ShellThick.setIsUsed(false);
        stopCrit_ActivateAM_Temp.setIsUsed(false);
        stopCrit_AMCoreRemesh.setIsUsed(false);
        stopCrit_AMShellRefine.setIsUsed(false);
        simulation_0.println("    stopCrit_ActivateAM_ShellThick:    "
                    + stopCrit_ActivateAM_ShellThick.getIsUsed() + "\n    "
                    + "stopCrit_ActivateAM_Temp:    " + stopCrit_ActivateAM_Temp.getIsUsed()
                    + "\n    stopCrit_AMCoreRemesh:    " + stopCrit_AMCoreRemesh.getIsUsed()
                    + "\n    stopCrit_AMShellRefine:    " + stopCrit_AMShellRefine.getIsUsed());

    // TIME-MODEL CRITERION
        //Steady-state
        stopCrit_FixedSteps.setIsUsed(false);
        stopCrit_MaxIters.setIsUsed(false);
        stopCrit_MaxIters.setMaximumNumberSteps(maxIters);
        simulation_0.println("    stopCrit_FixedSteps:    " + stopCrit_FixedSteps.getIsUsed()
              + "\n    stopCrit_MaxIters:    " + stopCrit_MaxIters.getIsUsed());
        if (stopCrit_MaxIters.getIsUsed()) {
           simulation_0.println("    --> Value set to:    "
                      + stopCrit_MaxIters.getMaximumNumberSteps());
        }

        // Transient
        stopCrit_FixedTime.setIsUsed(true);
        stopCrit_FixedTime.getFixedPhysicalTime().setValue(timeStep);
        stopCrit_MaxInnerIter.setIsUsed(true);
        stopCrit_MaxInnerIter.setMaximumNumberInnerIterations(innerIters_Initial);
        stopCrit_MaxTime.setIsUsed(true);
        stopCrit_MaxTime.getMaximumTime().setValue(maxTime);
        stopCrit_MinInnerIter.setIsUsed(false);
        stopCrit_MinInnerIter.setMinimumNumberInnerIterations(10);
        simulation_0.println("    stopCrit_FixedTime:    " + stopCrit_FixedTime.getIsUsed());
        if (stopCrit_FixedTime.getIsUsed()) {
           simulation_0.println("    --> Value set to:    "
                      + stopCrit_FixedTime.getFixedPhysicalTime().getSIValue());
        }
        simulation_0.println("    stopCrit_MaxInnerIter:    " + stopCrit_MaxInnerIter.getIsUsed());
        if (stopCrit_MaxInnerIter.getIsUsed()) {
           simulation_0.println("    --> Value set to:    "
                      + stopCrit_MaxInnerIter.getMaximumNumberInnerIterations());
        }
        simulation_0.println("    stopCrit_MaxTime:    " + stopCrit_MaxTime.getIsUsed());
        if (stopCrit_MaxTime.getIsUsed()) {
           simulation_0.println("    --> Value set to:    "
                      + stopCrit_MaxTime.getMaximumTime().getSIValue());
        }

// 'INITIAL' RUN
        iterator_0.run();
```

```
// SAVE PROGRESS
        saveProgress();
    }


//*-----------------------------------------------------------------------------------------------------------*//
//*---------------------------* INACTIVE-AMR RUN PROCEDURE *----------------------------*//
//*-----------------------------------------------------------------------------------------------------------*//
    private void amInactiveRUN() {
        simulation_0.println("\n\nINITIATING OPERATION:    'amInactiveRUN()'");

// UPDATE STOPPING CRITERIA
    // RESID CRITERION
        stopCrit_Continuity.setIsUsed(true);
        stopCrit_Energy.setIsUsed(true);
        stopCrit_Sdr.setIsUsed(true);
        stopCrit_SteadyShellGrowth.setIsUsed(false);
        stopCrit_Tke.setIsUsed(true);
        stopCrit_Xmomentum.setIsUsed(true);
        stopCrit_Ymomentum.setIsUsed(true);
        stopCrit_Zmomentum.setIsUsed(true);
        simulation_0.println("\nStopping Criteria Activation:\n    stopCrit_AMShellRefine:    true"
                + "\n    stopCrit_Energy:            true\n    stopCrit_Sdr:               true\n    "
                + "stopCrit_SteadyShellGrowth:        " + stopCrit_SteadyShellGrowth.getIsUsed()
                + "\n    stopCrit_Tke:               true\n    stopCrit_Xmomentum:       true\n    "
                + "stopCrit_Ymomentum:        true\n    stopCrit_Zmomentum:        true");

    // AM CRITERION
        stopCrit_ActivateAM_ShellThick.setIsUsed(true && use_ActivateAM_ShellThick);
        stopCrit_ActivateAM_Temp.setIsUsed(true && use_ActivateAM_Temp);
        stopCrit_AMCoreRemesh.setIsUsed(false);
        stopCrit_AMShellRefine.setIsUsed(false);
        simulation_0.println("    stopCrit_ActivateAM_ShellThick:    "
                + stopCrit_ActivateAM_ShellThick.getIsUsed()
                + "\n    stopCrit_ActivateAM_Temp:    "
                + stopCrit_ActivateAM_Temp.getIsUsed() + "\n    stopCrit_AMCoreRemesh:    "
                + stopCrit_AMCoreRemesh.getIsUsed() + "\n    stopCrit_AMShellRefine:    "
                + stopCrit_AMShellRefine.getIsUsed());

    // TIME-MODEL CRITERION
        //Steady-state
        stopCrit_FixedSteps.setIsUsed(false);
        stopCrit_MaxIters.setIsUsed(false);
        stopCrit_MaxIters.setMaximumNumberSteps(maxIters);
        simulation_0.println("    stopCrit_FixedSteps:    " + stopCrit_FixedSteps.getIsUsed()
                + "\n    stopCrit_MaxIters:    " + stopCrit_MaxIters.getIsUsed());
        if (stopCrit_MaxIters.getIsUsed()) {
           simulation_0.println("    --> Value set to:    "
                    + stopCrit_MaxIters.getMaximumNumberSteps());
        }

        // Transient
        stopCrit_FixedTime.setIsUsed(false);
        stopCrit_FixedTime.getFixedPhysicalTime().setValue(timeStep);
        stopCrit_MaxInnerIter.setIsUsed(true);
        stopCrit_MaxInnerIter.setMaximumNumberInnerIterations(innerIters_Normal);
        stopCrit_MaxTime.setIsUsed(true);
        stopCrit_MaxTime.getMaximumTime().setValue(maxTime);
        stopCrit_MinInnerIter.setIsUsed(false);
        stopCrit_MinInnerIter.setMinimumNumberInnerIterations(10);
        simulation_0.println("    stopCrit_FixedTime:    " + stopCrit_FixedTime.getIsUsed());
        if (stopCrit_FixedTime.getIsUsed()) {
           simulation_0.println("    --> Value set to:    "
                    + stopCrit_FixedTime.getFixedPhysicalTime().getSIValue());
        }
        simulation_0.println("    stopCrit_MaxInnerIter:    " + stopCrit_MaxInnerIter.getIsUsed());
        if (stopCrit_MaxInnerIter.getIsUsed()) {
           simulation_0.println("    --> Value set to:    "
                    + stopCrit_MaxInnerIter.getMaximumNumberInnerIterations());
        }
        simulation_0.println("    stopCrit_MaxTime:    " + stopCrit_MaxTime.getIsUsed());
        if (stopCrit_MaxTime.getIsUsed()) {
           simulation_0.println("    --> Value set to:    "
                    + stopCrit_MaxTime.getMaximumTime().getSIValue());
        }

// 'AM INACTIVE' RUN
        iterator_0.run();

// SAVE PROGRESS
        saveProgress();
    }


//*-----------------------------------------------------------------------------------------------------------*//
//*--------------------------------* NORMAL RUN PROCEDURE *-----------------------------------*//
//*-----------------------------------------------------------------------------------------------------------*//
    private void normalRUN() {
        simulation_0.println("\n\nINITIATING OPERATION:    'normalRUN()'");

// UPDATE STOPPING CRITERIA
    // RESID CRITERION
        stopCrit_Continuity.setIsUsed(true);
        stopCrit_Energy.setIsUsed(true);
        stopCrit_Sdr.setIsUsed(true);
        stopCrit_SteadyShellGrowth.setIsUsed(false);
        stopCrit_Tke.setIsUsed(true);
        stopCrit_Xmomentum.setIsUsed(true);
        stopCrit_Ymomentum.setIsUsed(true);
        stopCrit_Zmomentum.setIsUsed(true);
        simulation_0.println("\nStopping Criteria Activation:\n    stopCrit_AMShellRefine:    true"
                + "\n    stopCrit_Energy:            true\n    stopCrit_Sdr:               true\n    "
                + "stopCrit_SteadyShellGrowth:        " + stopCrit_SteadyShellGrowth.getIsUsed()
                + "\n    stopCrit_Tke:               true\n    stopCrit_Xmomentum:       true\n    "
                + "stopCrit_Ymomentum:       true\n    stopCrit_Zmomentum:       true");

    // AM CRITERION
        stopCrit_ActivateAM_ShellThick.setIsUsed(false);
        stopCrit_ActivateAM_Temp.setIsUsed(false);
        stopCrit_AMCoreRemesh.setIsUsed(true);
        stopCrit_AMShellRefine.setIsUsed(true);
        simulation_0.println("    stopCrit_ActivateAM_ShellThick:    "
                + stopCrit_ActivateAM_ShellThick.getIsUsed()
                + "\n    stopCrit_ActivateAM_Temp:    "
                + stopCrit_ActivateAM_Temp.getIsUsed() + "\n    stopCrit_AMCoreRemesh:    "
                + stopCrit_AMCoreRemesh.getIsUsed() + "\n    stopCrit_AMShellRefine:    "
                + stopCrit_AMShellRefine.getIsUsed());

    // TIME-MODEL CRITERION
        //Steady-state
        stopCrit_FixedSteps.setIsUsed(false);
        stopCrit_MaxIters.setIsUsed(false);
        stopCrit_MaxIters.setMaximumNumberSteps(maxIters);
        simulation_0.println("    stopCrit_FixedSteps:    " + stopCrit_FixedSteps.getIsUsed()
                + "\n    stopCrit_MaxIters:    " + stopCrit_MaxIters.getIsUsed());
        if (stopCrit_MaxIters.getIsUsed()) {
           simulation_0.println("    --> Value set to:    "
                    + stopCrit_MaxIters.getMaximumNumberSteps());
        }

        // Transient
        stopCrit_FixedTime.setIsUsed(false);
        stopCrit_FixedTime.getFixedPhysicalTime().setValue(timeStep);
        stopCrit_MaxInnerIter.setIsUsed(true);
        stopCrit_MaxInnerIter.setMaximumNumberInnerIterations(innerIters_Normal);
        stopCrit_MaxTime.setIsUsed(true);
        stopCrit_MaxTime.getMaximumTime().setValue(maxTime);
        stopCrit_MinInnerIter.setIsUsed(false);
```

```
        stopCrit_MinInnerIter.setMinimumNumberInnerIterations(10);
        simulation_0.println("    stopCrit_FixedTime:    " + stopCrit_FixedTime.getIsUsed());
        if (stopCrit_FixedTime.getIsUsed()) {
            simulation_0.println("    --> Value set to:    "
                        + stopCrit_FixedTime.getFixedPhysicalTime().getSIValue());
        }
        simulation_0.println("    stopCrit_MaxInnerIter:    " + stopCrit_MaxInnerIter.getIsUsed());
        if (stopCrit_MaxInnerIter.getIsUsed()) {
            simulation_0.println("    --> Value set to:    "
                        + stopCrit_MaxInnerIter.getMaximumNumberInnerIterations());
        }
        simulation_0.println("    stopCrit_MaxTime:    " + stopCrit_MaxTime.getIsUsed());
        if (stopCrit_MaxTime.getIsUsed()) {
            simulation_0.println("    --> Value set to:    "
                        + stopCrit_MaxTime.getMaximumTime().getSIValue());
        }

// 'NORMAL' RUN
        iterator_0.run();

// SAVE PROGRESS
        saveProgress();
    }



//*----------------------------------------------------------------------------------------------------------*//
//*--------------------------------* NEW-MESH RUN PROCEDURE *--------------------------------*//
//*----------------------------------------------------------------------------------------------------------*//
    private void meshedRUN() {
        simulation_0.println("\n\nINITIATING OPERATION:    'meshedRUN()'");

// UPDATE STOPPING CRITERIA
    // RESID CRITERION
        stopCrit_Continuity.setIsUsed(true);
        stopCrit_Energy.setIsUsed(true);
        stopCrit_Sdr.setIsUsed(true);
        stopCrit_SteadyShellGrowth.setIsUsed(false);
        stopCrit_Tke.setIsUsed(true);
        stopCrit_Xmomentum.setIsUsed(true);
        stopCrit_Ymomentum.setIsUsed(true);
        stopCrit_Zmomentum.setIsUsed(true);
        simulation_0.println("\nStopping Criteria Activation:\n    stopCrit_AMShellRefine:    true"
                    + "\n    stopCrit_Energy:            true\n    stopCrit_Sdr:               true\n    "
                    + "stopCrit_SteadyShellGrowth:       " + stopCrit_SteadyShellGrowth.getIsUsed()
                    + "\n    stopCrit_Tke:               true\n    stopCrit_Xmomentum:       true\n    "
                    + "stopCrit_Ymomentum:       true\n    stopCrit_Zmomentum:       true");
```

```
// AM CRITERION
   stopCrit_ActivateAM_ShellThick.setIsUsed(false);
   stopCrit_ActivateAM_Temp.setIsUsed(false);
   stopCrit_AMCoreRemesh.setIsUsed(false);
   stopCrit_AMShellRefine.setIsUsed(false);
   simulation_0.println("    stopCrit_ActivateAM_ShellThick:    "
             + stopCrit_ActivateAM_ShellThick.getIsUsed()
             + "\n    stopCrit_ActivateAM_Temp:    "
             + stopCrit_ActivateAM_Temp.getIsUsed()
             + "\n    stopCrit_AMCoreRemesh:    " + stopCrit_AMCoreRemesh.getIsUsed()
             + "\n    stopCrit_AMShellRefine:    " + stopCrit_AMShellRefine.getIsUsed());

// TIME-MODEL CRITERION
   //Steady-state
   stopCrit_FixedSteps.setIsUsed(false);
   stopCrit_MaxIters.setIsUsed(false);
   stopCrit_MaxIters.setMaximumNumberSteps(maxIters);
   simulation_0.println("    stopCrit_FixedSteps:    " + stopCrit_FixedSteps.getIsUsed()
             + "\n    stopCrit_MaxIters:    " + stopCrit_MaxIters.getIsUsed());
   if (stopCrit_MaxIters.getIsUsed()) {
      simulation_0.println("    --> Value set to:    "
               + stopCrit_MaxIters.getMaximumNumberSteps());
   }

   // Transient
   stopCrit_FixedTime.setIsUsed(true);
   stopCrit_FixedTime.getFixedPhysicalTime().setValue(timeStep);
   stopCrit_MaxInnerIter.setIsUsed(true);
   stopCrit_MaxInnerIter.setMaximumNumberInnerIterations(innerIters_Meshed);
   stopCrit_MaxTime.setIsUsed(true);
   stopCrit_MaxTime.getMaximumTime().setValue(maxTime);
   stopCrit_MinInnerIter.setIsUsed(false);
   stopCrit_MinInnerIter.setMinimumNumberInnerIterations(10);
   simulation_0.println("    stopCrit_FixedTime:    " + stopCrit_FixedTime.getIsUsed());
   if (stopCrit_FixedTime.getIsUsed()) {
      simulation_0.println("    --> Value set to:    "
               + stopCrit_FixedTime.getFixedPhysicalTime().getSIValue());
   }
   simulation_0.println("    stopCrit_MaxInnerIter:    " + stopCrit_MaxInnerIter.getIsUsed());
   if (stopCrit_MaxInnerIter.getIsUsed()) {
      simulation_0.println("    --> Value set to:    "
               + stopCrit_MaxInnerIter.getMaximumNumberInnerIterations());
   }
   simulation_0.println("    stopCrit_MaxTime:    " + stopCrit_MaxTime.getIsUsed());
   if (stopCrit_MaxTime.getIsUsed()) {
```

```
            simulation_0.println("    --> Value set to:    "
                        + stopCrit_MaxTime.getMaximumTime().getSIValue());
        }

// 'MESHED' RUN
        iterator_0.run();

// SAVE PROGRESS
        saveProgress();

        statusNewMeshRUN = false;
    }



//*--------------------------------------------------------------------------------------------------------------*//
//*---------------------------------* MESH PIPELINE EXECUTION *---------------------------------*//
//*--------------------------------------------------------------------------------------------------------------*//
    private void meshPipeline() {
        simulation_0.println("\n\nInitiating AM Refinement procedure...");

// UPDATE GLOBAL PARAMETERS
        // MESH_Shell_MinThickness_Factor
        ScalarGlobalParameter scalarGlobalParameter_0 = ((ScalarGlobalParameter)
                    simulation_0.get(GlobalParameterManager.class).getObject(
                    "MESH_Shell_MinThickness_Factor"));
        if (statusActiveAM) {
            scalarGlobalParameter_0.getQuantity().setValue(3.0);
        } else {
            scalarGlobalParameter_0.getQuantity().setValue(4.0);
        }

// UPDATE GEOMETRY PARTS
        // REFINE_Initial_Block Thickness (NF1)
        SolidModelPart solidModelPart_0 = ((SolidModelPart)
                    simulation_0.get(SimulationPartManager.class).getPart(
                    "REFINE_Initial_Block Thickness (NF1)"));
        simulation_0.get(SimulationPartManager.class).updateParts(new NeoObjectVector(
                    new Object[] {solidModelPart_0}));

    // UPDATE EXTRACTED THRESHOLD-PARTS
        ExtractedPart extractedPart_0 = ((ExtractedPart)
                    simulation_0.get(SimulationPartManager.class).getPart(
                    "Threshold - Core Remesh Cells"));
        ExtractedPart extractedPart_1 = ((ExtractedPart)
                    simulation_0.get(SimulationPartManager.class).getPart(
                    "Threshold - Shell Refinement Cells"));
        simulation_0.get(SimulationPartManager.class).updateParts(new NeoObjectVector(
                new Object[] {extractedPart_0}));
        simulation_0.get(SimulationPartManager.class).updateParts(new NeoObjectVector(
                new Object[] {extractedPart_1}));

// BOUNDED SHAPE OPERATION
        BoundedShapeCreatingOperation boundedShapeCreatingOperation_0 =
                ((BoundedShapeCreatingOperation) simulation_0
                .get(MeshOperationManager.class).getObject(
                "Bounded Shape - Refinement Volume_NF1"));
        if (n_Mesh < 1) {
            BoxShapeInflationControl boxShapeInflationControl_0 =
                boundedShapeCreatingOperation_0.getBoundedShapeValuesManager()
                .get(BoxShapeInflationControl.class);
            boxShapeInflationControl_0.setInflationMode(BoxShapeInflationControl
                .InflationMode.INDIVIDUAL_OFFSETS);

            BoxShapeIndividualOffsetsInflation boxShapeIndividualOffsetsInflation_0 =
                boxShapeInflationControl_0.getIndividualInflationOffsets();
            boxShapeIndividualOffsetsInflation_0.getPXOffset()
                .setDefinition("4*${MESH_Shell_Core_Thickness}");  // 0.005 meters
            boxShapeIndividualOffsetsInflation_0.getPYOffset()
                .setDefinition("4*${MESH_Mold_Core_Size}");  // 0.020 meters
            boxShapeIndividualOffsetsInflation_0.getPZOffset()
                .setDefinition("2*${MESH_Mold_Core_Size}");  // 0.010 meters
            boxShapeIndividualOffsetsInflation_0.getNXOffset()
                .setDefinition("${MESH_Trans_PLShell_LayerThickness}");  // 0.000875 meters
            boxShapeIndividualOffsetsInflation_0.getNYOffset()
                .setDefinition("4*${MESH_Mold_Core_Size}");  // 0.020 meters
            boxShapeIndividualOffsetsInflation_0.getNZOffset()
                .setDefinition("2*${MESH_Mold_Core_Size}");  // 0.010 meters
        }

        boundedShapeCreatingOperation_0.getInputGeometryObjects().setQuery(null);
        boundedShapeCreatingOperation_0.getInputGeometryObjects()
                .setObjects(extractedPart_0, extractedPart_1);

    // EXECUTE OPERATION
        boundedShapeCreatingOperation_0.execute();

// SURFACE WRAPPER OPERATION
        SurfaceWrapperAutoMeshOperation surfaceWrapperAutoMeshOperation_0 =
                ((SurfaceWrapperAutoMeshOperation) simulation_0
                .get(MeshOperationManager.class).getObject("Surface Wrapper_NF1"));
    // EXECUTE OPERATION
        surfaceWrapperAutoMeshOperation_0.execute();
```

```
// AUTOMATED MESH OPERATION
   // REFERENCED GEOMETRY
      MeshOperationPart meshOperationPart_0 = ((MeshOperationPart) simulation_0
                .get(SimulationPartManager.class)
                .getPart("Bounded Shape - Refinement Volume_NF1"));
      MeshOperationPart meshOperationPart_1 = ((MeshOperationPart) simulation_0
                .get(SimulationPartManager.class).getPart("Surface Wrapper_NF1"));

   // AUTOMATED MESHER
      AutoMeshOperation autoMeshOperation_0 = ((AutoMeshOperation) simulation_0
                .get(MeshOperationManager.class).getObject("Automated Mesh"));
      if (use_ParallelMesher) {
         autoMeshOperation_0.getMesherParallelModeOption()
                .setSelected(MesherParallelModeOption.Type.PARALLEL);
      } else {
         autoMeshOperation_0.getMesherParallelModeOption()
                .setSelected(MesherParallelModeOption.Type.SERIAL);
      }
      autoMeshOperation_0.setLocalSurfaceMeshing(statusActiveAM);

   // LOCAL REFINEMENT: VOLUME EXTENT
      VolumeLocalMeshingExtent volumeLocalMeshingExtent_0 =
                ((VolumeLocalMeshingExtent) autoMeshOperation_0
                .getLocalMeshingExtents().getObject("Volume Extent_NF1"));
      volumeLocalMeshingExtent_0.getLocalMeshingObjects().setQuery(null);
      volumeLocalMeshingExtent_0.getLocalMeshingObjects()
                .setObjects(meshOperationPart_0);
      volumeLocalMeshingExtent_0.setEnableExtent(statusActiveAM);

   // CUSTOM CONTROL: NF1 SHELL REFINEMENT
      VolumeCustomMeshControl volumeCustomMeshControl_0 =
                ((VolumeCustomMeshControl) autoMeshOperation_0
                .getCustomMeshControls().getObject("Vol: Refine Dynamic Shell - NF"));
      volumeCustomMeshControl_0.getGeometryObjects().setQuery(null);
      volumeCustomMeshControl_0.getGeometryObjects().setObjects(meshOperationPart_1);
      volumeCustomMeshControl_0.setEnableControl(statusActiveAM);

   // EXECUTE OPERATION
      autoMeshOperation_0.execute();
      n_Mesh++;


      // 'ADAPTIVE MESH'
      if (statusActiveAM) {
         saveFileVersion("@meshed");
```

```
        // STATUS UPDATES
            statusRemesh = 0;
            statusNewMeshRUN = true;
        } else {// 'CONTROL'
          simulation_0.saveState(dirName + sep + simBasename + ".sim");
        }

    }


//*-------------------------------------------------------------------------------------------------------------*//
//*----------------------------* FILENAME PROGRESS-DESCRIPTOR *----------------------------*//
//*-------------------------------------------------------------------------------------------------------------*//
    private String progressDescriptor() {
        String descriptor;    // String to be returned string
        String sProg = "";
        String sExp = "";    // String representation of the exponent
        String zeros = "";    // String of 'placeholder' zeros to fill out the filename descriptor
        int nChars = autoSave_nDigits + ((autoSave_Type.equals("DELTATIME")) ? 1 : 0);
                    // Number of sim 'progress' characters in descriptor
        int nDigits;    // Number of 'placeholder' zeros to remove
        int nZeros = nChars;    // Starting number of 'placeholder' zeros
        for (int i=0; i < nZeros; i++) {
            zeros += "0";
        }

        switch (autoSave_Type) {
            case "ITERATION": int iter = iterator_0.getCurrentIteration();
                sProg = String.valueOf(iter);
                break;
            case "TIMESTEP":
                int step = iterator_0.getCurrentTimeLevel();
                sProg = String.valueOf(step);
                break;
            case "DELTATIME":
                double t = solution_0.getPhysicalTime();
                sExp += "e";
                int power = 0;
                int z = 0;
            // Obtain the scientific notation for the physical time
                if (t < 1) {
                    sExp += "-";
                    while (z < 1) {
                        if ((10*t) > 1) {
                            z = 2;
                        } else {
```

```
                    t *= 10;
                    power--;
                }
            }
        } else {
            sExp += "+";
            if (t > 10) {
                while (z < 1) {
                    if ((t/10) < 1) {
                        z = 2;
                    } else {
                        t /= 10;
                        power++;
                    }
                }
            }
        }
        sExp += ((Math.abs(power) < 10) ? "0" : "") + String.valueOf(Math.abs(power));
        sProg = String.valueOf(t);
        break;
    }
    nDigits = sProg.length();
    zeros = zeros.substring(nDigits);    // Remove excess zeros
    descriptor = ((autoSave_Type.equals("DELTATIME")) ? (sProg + zeros) : (zeros + sProg));

    if (descriptor.length() > nChars) {
        descriptor = ((autoSave_Type.equals("DELTATIME"))
                ? descriptor.substring(0, (descriptor.length() - 1)) : descriptor.substring(1));
    }

    return ("@" + descriptor + sExp);
}


//*-----------------------------------------------------------------------------------------------------*//
//*-----------------------* SAVE CURRENT SIMULATION PROGRESS *------------------------*//
//*-----------------------------------------------------------------------------------------------------*//
// SAVE SIMULATION AT CURRENT ITER/TIME-STEP/TIME
  private void saveProgress() {
    String desc = progressDescriptor();
    simulation_0.saveState(dirName + sep + simBasename + desc + ".sim");

  // UPDATE SIMNAME
    simName = simulation_0.getPresentationName();
    simNameDescriptor = ((simName.split("@",0).length > 1)
              ? simName.substring(simName.indexOf("@")) : "");
        opDescriptor = ((simName.split("@",0).length > 1)
                ? simName.substring(simName.lastIndexOf("@")) : "");
    }


//*-------------------------------------------------------------------------------------------------------------*//
//*---------------* SAVE FILE-VERSION WITH IRREGULAR DESCRIPTOR *---------------*//
//*-------------------------------------------------------------------------------------------------------------*//
    private void saveFileVersion(String version) {
        String desc = progressDescriptor();
        simulation_0.saveState(dirName + sep + simBasename + desc + version + ".sim");

    // UPDATE SIMNAME
        simName = simulation_0.getPresentationName();
        simNameDescriptor = ((simName.split("@",0).length > 1)
                ? simName.substring(simName.indexOf("@")) : "");
        opDescriptor = ((simName.split("@",0).length > 1)
                ? simName.substring(simName.lastIndexOf("@")) : "");
    }

}  // END OF MACRO
```

REFERENCES

[1] S. Louhenkilpi, “Continuous Casting of Steel,” *Treatise Process Metall.*, pp. 373–434, Jan. 2014.

[2] “From steel to semi-finished products - tec-science,” *tec-science*. [Online]. Available: https://www.tec-science.com/material-science/steel-making/steel-semi-finished-products-continuous-ingot-casting/. [Accessed: 14-Nov-2019].

[3] B. G. Thomas, “Fluid Flow in the Mold,” in *AISE - Making, Shaping, and Treating of Steel*, 2003, pp. 14.1-14.41.

[4] B. G. Thomas, “Continuous Casting: Modeling,” *Encycl. Adv. Mater.*, vol. 2, pp. 1–23, 1999.

[5] J. K. Brimacombe and K. Sorimachi, “Crack formation in the continuous casting of steel,” *Metall. Trans. B*, vol. 8, no. 2, pp. 489–505, 1977.

[6] F. R. Camisani-Calzolari, I. K. Craig, and P. C. Pistorius, “A review on causes of surface defects in continuous casting,” in *IFAC Proceedings Volumes (IFAC-PapersOnline)*, 2003, vol. 36, no. 24, pp. 113–121.

[7] S. Yu *et al.*, “Stress and Friction Distribution around Slab Corner in Continuous Casting Stress and Friction Distribution around Slab Corner in Continuous Casting Mold with Different Corner Structures,” *Metall. Mater. Trans. B*, vol. 49, no. 3, pp. 866–876, 2018.

[8] B. G. Thomas, G. Li, A. Moitra, and D. Habing, “Analysis of thermal and mechanical behavior of copper molds during continuous casting of steel slabs,” *80th Steelmak. Conf. Chicago*, pp. 1–19, 1998.

[9] M. Gonzalez, M. B. Goldschmit, A. P. Assanelli, E. N. Dvorkin, and E. F. Berdaguer, “Modeling of the solidification process in a continuous casting installation for steel slabs,” *Metall. Mater. Trans. B*, vol. 34, no. 4, pp. 455–473, 2003.

[10] M. R. R. I. Shamsi and S. K. Ajmani, “Analysis of mould, spray and radiation zones of continuous billet caster by three-dimensional mathematical model based on a turbulent fluid flow,” *Steel Res. Int.*, vol. 81, no. 2, pp. 132–141, Feb. 2010.

[11] D. Jiang and M. Zhu, “Flow and Solidification in Billet Continuous Casting Machine with Dual Electromagnetic Stirrings of Mold and the Final Solidification,” *Steel Res. Int.*, vol. 86, no. 9, pp. 993–1003, 2015.

[12] V. Singh *et al.*, “Modeling of Solidified Shell Formation in Slab Caster and the Role of Process Parameters,” in *AISTech 2016 Proceedings*, 2016, pp. 1531–1546.

[13] Z. Cai and M. Zhu, “Non-uniform heat transfer behavior during shell solidification in a wide and thick slab continuous casting mold,” *Int. J. Miner. Metall. Mater.*, vol. 21, no. 3, pp. 240–250, Mar. 2014.

[14] B. G. Thomas, “Modeling of Continuous-Casting Defects Related to Mold Fluid Flow,” in *AIST - 3rd Internat. Congress on Science & Technology of Steelmaking*, 2005, vol. 3, no. 7, pp. 847–861.

[15] L. Zhang, S. Yang, K. Cai, J. Li, X. Wan, and B. G. Thomas, “Investigation of fluid flow and steel cleanliness in the continuous casting strand,” *Metall. Mater. Trans. B Process Metall. Mater. Process. Sci.*, vol. 38, no. 1, 2007.

[16] V. Singh, S. K. Dash, J. S. Sunitha, S. K. Ajmani, and A. K. Das, “Experimental Simulation and Mathematical Modeling of Air Bubble Movement in Slab Caster Mold,” *ISIJ Int.*, vol. 46, no. 2, pp. 210–218, 2006.

[17] A. Kumar, M. Založnik, H. Combeau, B. Goyeau, and D. Gobin, “Numerical Simulation of Columnar Solidification: Influence of Inertia on a Channel Segregation,” *Model. Simul. Mater. Sci. Eng.*, vol. 21, no. 4, pp. 1–16, 2013.

[18] C. Beckermann, H.-J. Diepers, I. Steinbach, A. Karma, and X. Tong, “Modeling Melt Convection in Phase-Field Simulations of Solidification,” *J. Comput. Phys.*, vol. 154, no. 2, pp. 468–496, 1999.

[19] T. Telejko, Z. Malinowski, and M. Rywotycki, “Analysis of Heat Transfer and Fluid Flow in Continuous Steel Casting,” *Arch. Metall. Mater.*, vol. 54, no. 3, pp. 837–844, 2009.

[20] M. Rappaz and C.-A. Gandin, “Probabilistic Modelling of Microstructure Formation in Solidification Processes,” *Acta Metall. Mater.*, vol. 41, no. 2, pp. 345–360, 1993.

[21] A. Jacot and M. Rappaz, “A pseudo-front tracking technique for the modelling of solidification microstructures in multi-component alloys [Acta Materialia 50(8), pp. 1909–1926],” *Acta Mater.*, vol. 50, pp. 1909–1926, 2002.

[22] Y. M. Won and B. G. Thomas, “Simple model of microsegregation during solidification of steels,” *Metall. Mater. Trans. A Phys. Metall. Mater. Sci.*, vol. 32, no. 7, pp. 1755–1767, 2001.

[23] B. Wiwatanapataphee, Y. H. Wu, J. Archapitak, P. F. Siew, and B. Unyong, “A numerical study of the turbulent flow of molten steel in a domain with a phase-change boundary,” *J. Comput. Appl. Math.*, vol. 166, no. 1, pp. 307–319, 2004.

[24] B. Chen, M. T. Moore, J. Trimble, K. Morales, and A. Silaen, “Analysis of Continuous Caster Flow Pattern Variations Relating to Argon Injection,” in *Thermal and Fluids Engineering Conference*, 2018.

[25] C. Pfeiler, “Modeling of Turbulent Particle / Gas Dispersion in the Mold Region and Particle Entrapment into the Solid Shell of a Steel Continuous Caster,” Montanuniversität Leoben, 2008.

[26] F. R. Menter, “Two-equation eddy-viscosity turbulence models for engineering applications,” *AIAA J.*, vol. 32, no. 8, pp. 1598–1605, Aug. 1994.

[27] D. Currey, R. Global, D. Hamilton, and B. G. Thomas, “Utilization of CON1D at ArcelorMittal Dofasco ’ s No . 2 Continuous Caster for Crater End Determination,” *AISTech - Iron Steel Technol. Conf. Proc.*, vol. I, no. 2, pp. 1177–1186, 2009.

[28] Y. Meng and B. G. Thomas, “Heat-Transfer and Solidification Model of Continuous Slab Casting: CON1D,” *Metall. Mater. Trans. B*, vol. 34B, no. October, pp. 685–705, 2003.