Final Technical Report

Castability Assessment and Data Integration

DOE Award Number: DE-FC07-00ID13978
Project Period: 9/30/2000 to 9/29/2004

Submitted to Department of Energy

March 31, 2005

R. Allen Miller, PI
614-292-7067
miller.6@osu.edu

The Ohio State University
Industrial, Welding and Systems Engineering
1971 Neil Avenue
Columbus, OH 43210

Project Team:
Mahesh Jha, DOE Golden, CO
Michael Welther, OSU Research Foundation
Steve Udvardy, NADCA Technical Director

Acknowledgment: “This report is based upon work supported by the U. S. Department of Energy under Award No. DE-FC07-00ID13978 “.

EXECUTIVE SUMMARY

This report is primarily based on the PhD dissertation of Dongtao Wang[1] and the MS thesis of Xiaorui Chen[2]. Section I, based on Wang's work, addresses algorithms and methods developed to perform equilibrium die temperature calculations without the need for simulation of multiple casting cycles as typically required. Wang's work also includes improvement to the qualitative reasoning algorithms used for die filling visualization in the CastView program plus preliminary work designed to extend the die filling visualization techniques used to slower fill processes such as permanent mold and gravity casting. Section II is based on Chen's work and addresses techniques to rapidly locate cooling lines for die temperature analysis and extract quantitative data useful for castability assessment from voxel data created by CastView.

There are usually two concerns for die casting designers, thermal characteristics and fill pattern because they are closely related to castability, casting quality, and die life. The traditional way to obtain information about these phenomena is numerical simulation. However, due to the complexity of the equation system to be solved, the computational cost is high and numerical simulation is very time consuming particularly for a quasi-equilibrium process using a permanent mold such as die casting. This makes numerical

[1] Dongtao Wang, "Equilibrium Temperature Analysis And Fill Pattern Reasoning For Die Casting Process," PhD Dissertation, Industrial and Systems Engineering, The Ohio State University, 2004.

[2] Xiaorui Chen , "Graphical User Interface For Cooling Line Functions And Surface Rendering," MS Thesis, Mechanical Engineering, The Ohio State University, 2003.

simulation an imperfect tool during the early stages of product development. It is very desirable for designers to have an efficient and reliable tool to quickly calculate the die and part temperature distributions and die fill pattern in order to support interactive design.

In this study, a fast algorithm to compute the equilibrium temperature of the die and ejection temperature of the part was developed. The equilibrium temperature is defined as the time average temperature over a cycle after the process reaches the quasi steady state. The spatial distribution of the average temperature is useful for cycle and die cooling/heating design.

The main challenge in computing the average temperature is determining the heat released from the part and accounting for the time varying conditions at the die/part interface and the die parting surfaces. Several models to compute the heat released from part were tested and a surrogate model developed for this purpose. Average conditions were used to account for heat transfer calculation at the interfaces and special attention was paid to computational efficiency improvement. The algorithm also addresses the modeling of cooling/heat line, spray effects and techniques for die splitting at the parting line. The algorithm has been implemented in the software CastView.

The equilibrium temperature methods described above would not be effective without techniques to place and describe the die cooling lines. Cooling channel placement is often designed by guesswork or by past experience. It is very expensive and time consuming to modify the improperly placed cooling channels after the die has been built.

Chen's work involved implementing a Graphical User Interface to design and modify the die cooling lines so that cooling could be incorporated in the termperature calculations. The system provides three methods to specify a cooling line:

1. direct input of the coordinates of the cooling line nodes,
2. orthogonal sketch and
3. free sketch.

The system also allows the user to modify an existing cooling line by changing its node coordinates, orthogonal editing, or graphical editing.

From a castability point of view it is highly desirable to design the part with a reasonably uniform wall thickness that avoids abrupt changes in wall thickness. In order to give the designers an overall view of the wall thickness throughout the part, skeletons computed using the existing thin section analysis were used to provide local wall thickness data. From the skeletons, the thickness values are propogated to the part surface so that the wall thickness can be displayed on the part surface. From the thickness associated with the surface, a histogram of the part wall thickness is constructed. The histogram makes it easy to compute minimum and average wall thicknesses needed for fill time specification and gate design.

The geometric reasoning algorithm used in CastView for fill pattern analysis was also redesigned. In this qualitative method, the flow behavior is calculated using the cavity geometric information. Many shortcomings in the old algorithm were fixed and improved. The new algorithm includes considerations which affect the flow behavior, such as flow resistance, more flow angle search and influence within neighborhood. Special attention is paid to computational efficiency improvement.

The fill pattern algorithm for die casting process, a high speed process, was adapted for slower fill processes including gravity casting and squeeze casting. The dominant term for flow behavior for different process is defined from dimensionless Navier-Stokes equations. Based on this analysis, the fill pattern algorithm for die casting is modified for slow fill processes.

The analysis results using the algorithms presented were compared with those obtained from numerical simulations, historical data and experiments. The comparisons generally show good agreement. Given the typical computational time of a few minutes, the efficiency of the algorithms is remarkable.

Currently, the application of equilibrium temperature analysis is die casting and fill pattern reasoning is die casting, gravity casting and squeeze casting. However, the algorithms developed in this study can be adapted and applied to other net shape processes.

The specific contributions made by this research are:

- Developed a mathematical algorithm to compute the equilibrium temperature of the die and the ejection temperature of part. The heat transfer concepts in steady state are introduced to quasi steady state to calculate the heat balance over a cycle. Special attention was paid to the calculation of heat released from the part and an approximate model developed that provides the part ejection temperature..
- Composite heat transfer at interface, average temperature at different stages of the casting cycle, cooling line effects, spray effects and die splitting at parting surface

were addressed. Computational efficiency was considered by applying special methods to reduce the computation.

- The algorithm was implemented in the CastView program providing a quick tool to evaluate the cycle and die cooling/heating design. Compared to hours or days of run time using numerical simulation to compute multiple cycles for a dynamic solution, the run time of CastView for equilibrium temperature is only a few minutes.

- The old fill pattern algorithm used in CastView for die casting processes was redesigned. Flow speed, flow resistance, flow potential and influence within neighboring flows were included. The method to compute new vector when the flow hits an obstruction and the searching for available vectors were both improved.

- The new algorithm for fill pattern was implemented in the program CastView, providing a quick and efficient tool to evaluate fill pattern in cavity. The run time for a typical case on CastView is only ~10 minutes while that on numerical simulation packages is hours.

- The dominant terms on flow behavior for die casting process, gravity casting process and squeeze casting process from Navier-Stokes equations were determined and fill pattern algorithms for gravity casting and squeeze casting were obtained by modifying the algorithm used for the die casting process.

The tool for equilibrium temperature can quickly compute the overall temperature distribution of die and ejection temperature of part. This helps the user evaluate the effect

of cooling and heating and verify the cycle design. Due to the efficiency of this tool, the user can explore numerous alternative designs, a task that is more difficult for numerical simulation packages.

The limitation of the analysis for equilibrium temperature is that only steady state conditions are considered. This is to simplify the computation but it also introduces some problems. The heat transfer is computed based on the time average temperature. However, the actual heat transfer happens with a dynamic changing temperature. Information such as peak temperature is therefore not available.

The tool for fill pattern analysis provides an alternative means other than numerical simulation. The advantage of geometric reasoning is the computational efficiency. It can produce a fill pattern prediction in a few minutes. This is particularly useful in early stages of design. Compared with the old fill pattern algorithm, the new algorithm improves the calculation and considers more factors. The results are more accurate but the computation time is only slightly increased. Based on an analysis of the Navier-Stokes equations and geometric reasoning, the fill pattern algorithms for gravity casting and squeeze casting can provide quick evaluation for fill patterns of these two processes.

The limitation for fill pattern analysis is that this method is solely based on geometric reasoning thus there is limited physics behind the actual calculations. There is no strict consideration of mass conservation, momentum conservation and energy conservation. There is no calculation for heat transfer and solidification during flow analysis. This lack

has an inevitable effect on the ultimate accuracy but it does not limit the utility of the approach for tasks such as parting plane evaluation and gate and vent placement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SECTION I:

EQUILIBRIUM TEMPERATURE ANALYSIS AND FILL PATTERN REASONING FOR DIE CASTING PROCESS[3]

[3] Based on the work of Dongtao Wang, "Equilibrium Temperature Analysis And Fill Pattern Reasoning For Die Casting Process," PhD Dissertation, Industrial and Systems Engineering, The Ohio State University, 2004.

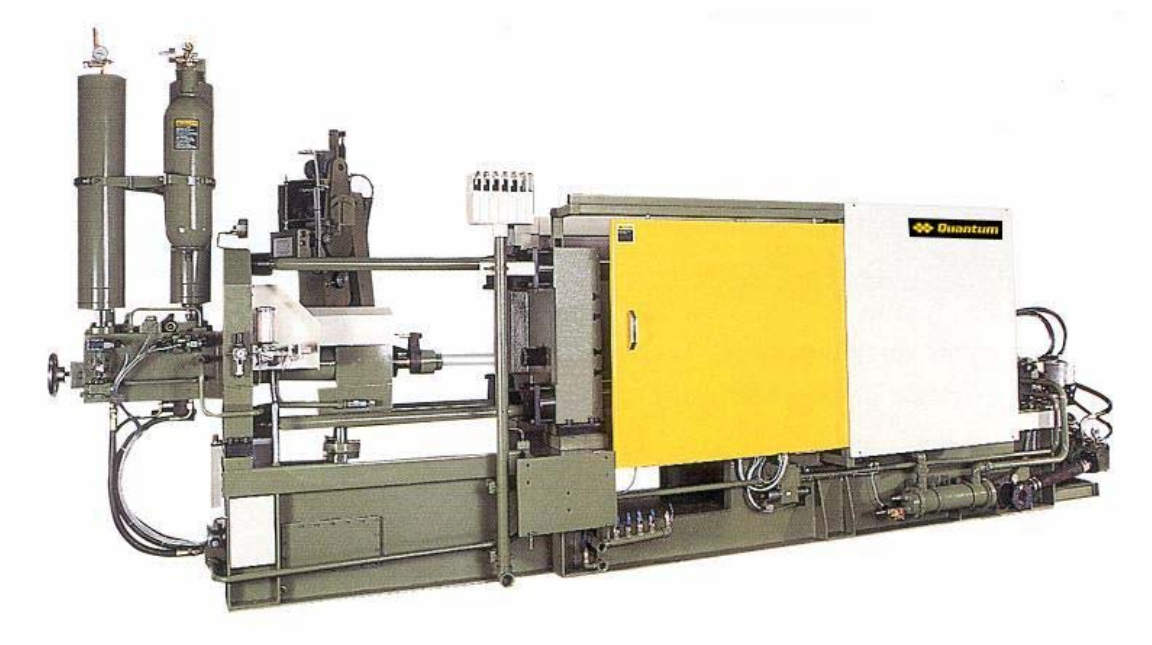
1. BACKGROUND AND INTRODUCTION

1.1 Die Casting Process

Die casting (or high pressure die casting in Europe) is one of the most important net shape manufacturing processes. In this process, metal liquid is injected at high pressure and high speed into metal die cavity, with subsequent solidification into useful shapes. Compared with other metal shaping processes such as sand casting, rolling and forging, the die casting process has some advantages such as capability to produce complex metal parts with clear surface and thin walls and with high productivity. These advantages make die casting a more and more important process in the modern industries. The applications of die castings increased significantly in the recent two decades, especially in the automobile, aerospace, electronics and medical apparatus industries. It was predicted that such applications will be further expanded in the future (Gu, 1995).

One requirement for metals to be die cast is they should have low melting point because the die and injection system are usually made by steel. The melting point of a metal to be die cast must be much lower than that of steel otherwise the die may be damaged or distorted. Thus, at present, the majority of die casting alloys are aluminum alloys, zinc alloys and magnesium alloys because of their low melting points. A die casting machine is used to produce die casting parts. There are two types of die casting machines, hot chamber machine and cold chamber machine depending on the metal delivery method. A hot chamber machine has a crucible, where the alloy is melted and is delivered to

injection system directly. In a cold chamber die casting machine, the melt is delivered from a separate furnace. Fig. 1.1 shows a typical cold chamber die casting machine.



Section I-Figure 1.1: typical cold chamber die casting machine (www.quantum-online.com, 2002)

It is generally acknowledged that the development of die casting began from the early type-letter production with lead alloys. In 1822, William Church introduced a machine with an output of up to 20,000 letters per day, which is believed the first successful use of the die casting process (Barton, 1991). In 1839, the first patent of die casting using a piston die casting machine was issued (Kaye and Street, 1982). During the 19th century, the principles of die casting were being applied to the specialized field of printing, and the process was beginning to extend to the production of other articles. But only in the

20th century, especially in the World War II, die casting industry experienced its fast expansion period due to the demand from defense industry (Kaye and Street, 1982).

An example may be able to show the rapid expansion of die casting industry. From 1991 to 2001, the amount of magnesium used in passenger vehicles had an annual growth rate of 12%. In 2001, vehicles in production averaged 2.3 kg (5 lb) of magnesium and the most magnesium-intensive vehicles tipped the content scale at 25 kg (55 lb). "We expect magnesium to grow even faster in the next 10 years, and that's not even talking about powertrain (components)," said Bob Powell, Product Leader of the USCAR/U.S. Automotive Materials Partnership. (Kami Buchholz, 2001).

In the recent years, great progressive development has been made in die casting alloys, die casting technology, die casting machines and control devices. Die casting industry is now in its "golden age" and will see its expansion in the future.

1.2 Problem Statement

During the die casting process, liquid or semi-solid metal is injected at high speed to fill a complex die cavity, and solidifies rapidly under high pressure. It is a complicated physical-chemical process. An incorrect process design often causes die casting defects, such as porosity and deformation, and also shortens the die life. To realize appropriate thermal balance and mold filling thus to ensure quality die casting products and prolong the die life, are two important issues for the die casting process design (Allchin, 1990).

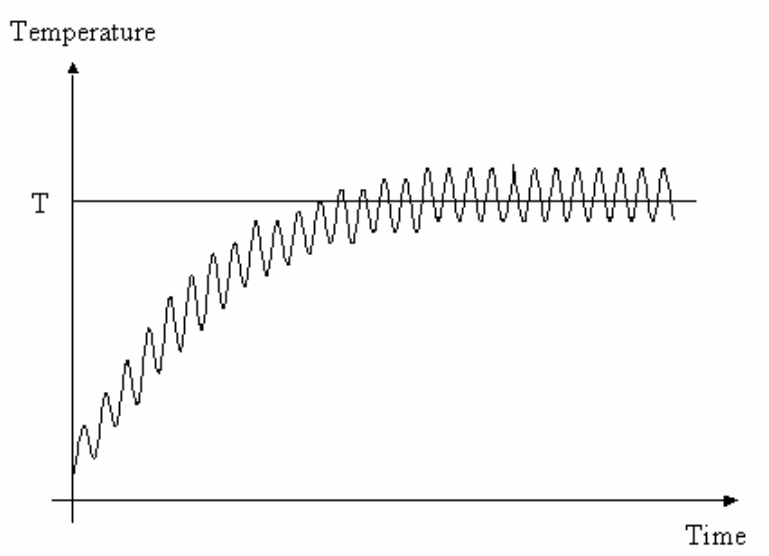
1.2.1 Heat balance at equilibrium status

The thermal characteristics of the die and casting are always fundamental considerations for designers. At the conceptual design stage, the typical questions about thermal characteristics include: 1) What does the spatial temperature distribution of the die and part look like? 2) Where are the hot spots? 3) What is the effect of cooling lines and spray? 4) What is the ejection temperature of the part when a certain cycle is applied? Numerical simulation of the transient temperature is the typical way to obtain the answer. The typical procedure for numerical simulation is: build CAD models for part and dies – input thermal property data and cycle parameters – run simulation – view analysis results. The analysis results include the dynamic temperature change of part and dies. Thus the user can read temperature at any location and at any time from the results.

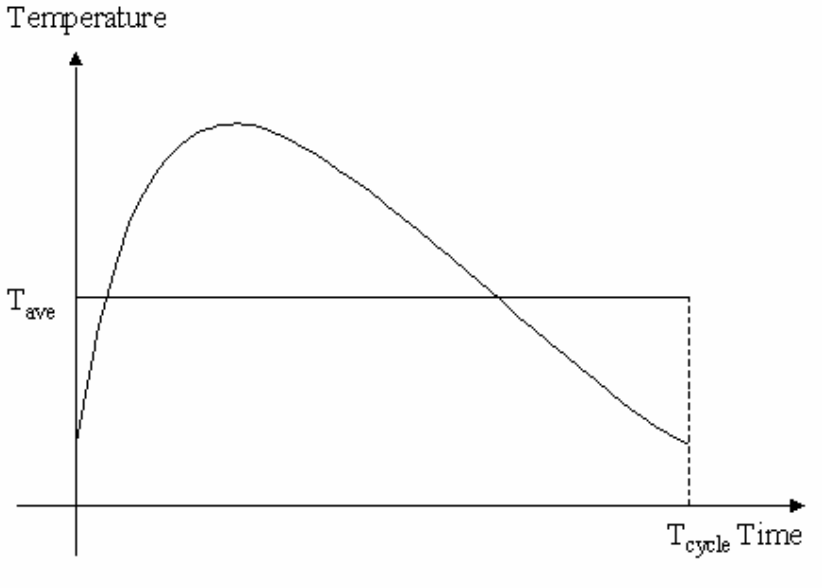
However, one disadvantage of transient solutions is that to obtain the thermal profile at quasi steady state literally hundreds of cycles are needed since the start-up is usually far away from the steady state. This is very time consuming and provides information that is beyond that needed for the cycle and die cooling design since cycle and cooling design only requires the overall temperature distribution and part ejection temperature at steady state. It does not require information about the dynamic temperature change in a cycle. In addition, at the conceptual design stage, there may be many alternate designs and exploration of the design space may be limited due to the time required to explore all alternatives. Thus it is very desirable to have a tool for thermal analysis that provides the needed information and runs very quickly supporting interactive redesign.

In quasi steady state, the temperature of any point at the die varies throughout the casting cycle but returns to the same value at the same relative time in each cycle. The equilibrium temperature is defined as the time average temperature over a cycle after the process reaches quasi steady state which is illustrated using Fig. 1.2 and Fig. 1.3. Considering any point in the die, its temperature change curve over time is shown in Fig. 1.2. After a number of start-up cycles, cycle to cycle change is small and the process reaches the quasi steady state. The temperature change of this point over a single cycle thus can be illustrated using Fig. 1.3. The cycle can be further separated into a few stages, die close, injection and solidification, die open, ejection, spray and idle stage. The temperature keeps changing in every stage. However, we define the equilibrium temperature, which is hypothetical and does not actually exist. The equilibrium temperature is defined as time average temperature over a cycle for die and ejection temperature for part when the process reaches the quasi steady state.

Note that this is time average and not a spatial average so the temperature still varies spatially. This equilibrium temperature is helpful for cycle and cooling design, especially at conceptual design stage. A part of the research is to develop and implement a quick approach to compute the equilibrium temperature of die and part.



Section I-Figure 1.2:Temperature change curve of a point in die over time. Each jag denotes a cycle.



Section I-Figure 1.3: Temperature change of the same point over a cycle in equilibrium state. At the end of a cycle, temperature comes back to the level at the beginning of the cycle.

1.2.2 Fill pattern in die cavity

Another common concern of designers is the fill pattern of metal liquid as it enters the cavity. Under high speed and high pressure, the liquid metal flows through the narrow and complex shaped die cavity. This is a very fast transient process which is usually atomized or turbulent flow. This filling process is one of the most important factors to determine the product quality. The metal flow, if not controlled precisely, can cause flow-related defects, which range from gas porosity and knit lines to incomplete-fill.

The most common flow-related defect might be gas porosity. During the cavity filling, ideally, the liquid metal pushes the cavity gas ahead to the vent as the flow front advances and the vent region is the last area to be filled. But if the flow is not controlled appropriately, the vent is sealed before the whole cavity is filled, thus some of the cavity gas is trapped in liquid metal. As the liquid metal solidifies and cools, the trapped gas in the product becomes the gas porosity. This defect usually results in poor product surface and low strength and is a common reason for rejection of die casting parts (Barone and Caulk, 2000).

Since the filling plays an important role in the product quality, many researchers put their effort on fill pattern prediction. At the present time, the prevailing method is still numerical simulation, which is based on mass conservation, momentum conservation and energy conservation (Lu, et. al. 1999, Khayat, 1998 and Sulaiman, et. al. 2000). Due to the complexity of the equations, the equation solving is usually a time consuming task. Furthermore, the simulation systems require that the user has much experience and

understanding of fluid dynamics. These disadvantages limit the utility of numerical simulation and the number of “what-if” questions, especially early in design.

An alternative method is qualitative reasoning for fill pattern prediction. This method is based on the geometric characteristics of the die cavity so it only requires the input of the part and gate geometry and does not require fluid dynamics background of the user. More important, there are no complex equations to be solved so the prediction can be obtained quickly. These reasons make it particularly suitable for the early stage design.

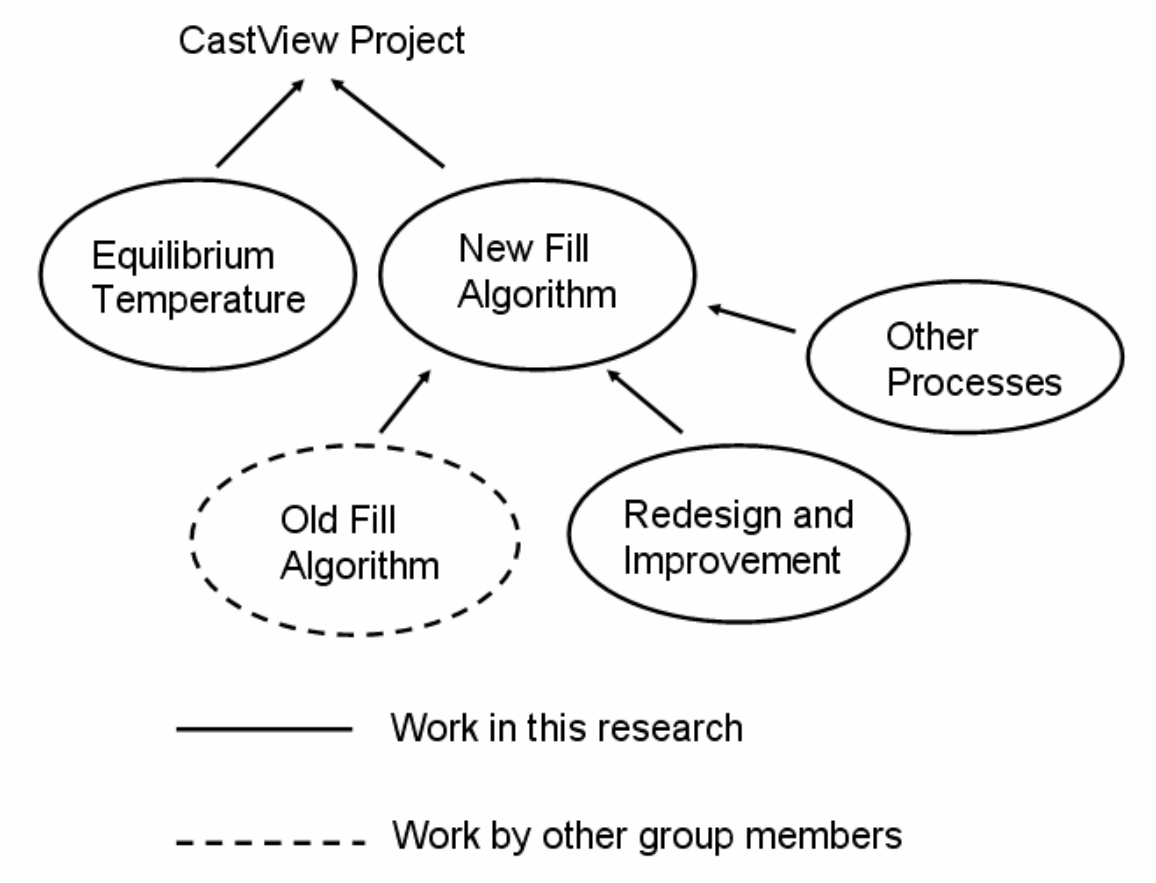
The CastView research group at The Ohio State University developed a qualitative reasoning algorithm for fill pattern prediction (Miller, 1998, Rebello, 1997 and Elfakharany, 1999). According to its application, it helps the improvement of gating system design at the qualitative design stage. There is, however, need to improve the algorithm for better results and for wider application. Another part of this research was to develop a better model for fill pattern to replace the existing one.

Problems for the old model include: 1) Velocity, pressure and resistance are not considered; 2) Calculation for multiple gate sometimes is wrong; 3) Flow pattern may be incorrect when there is an obstruction; 4) There is bias when filling a symmetric part. These problems, which will be discussed in detail in chapter 4 and chapter 5, contribute to an incorrect fill pattern prediction and should be addressed in the new model.

In addition, the calculation for fill pattern in CastView is limited to the high pressure die casting process and is not applicable for other similar processes, such as gravity die casting and squeeze casting. Study on filling of these processes shows that the major

difference of them is the so-called dominating force (will be further discussed in Chapter 7). Thus it is possible to modify the approach of geometric reasoning on die casting fill pattern for other processes. Another part of this research is to find a general algorithm (or minor modification from die casting algorithm) for these processes.

Fig. 1.4 illustrates the structure of this research cooperating with other work. CastView is a comprehensive CAD/CAE project for die casting process while equilibrium temperature analysis and fill pattern computation are two sub-projects. Research in this dissertation includes equilibrium temperature analysis and redesign and improvement of old algorithm for fill.



Section I-Figure 1.4: Structure of research in this report, which is associated to CastView project and previous work by other group member.

1.3 Literature Review

1.3.1 Numerical simulation for die casting process

In the traditional die casting design, engineers optimize the die casting process through a "try-revise" loop, which results in an expensive trial fee and a long trial cycle. To overcome these problems, the die casting researchers have introduced computer technology into die casting design. The traditional design is being replaced by the die casting CAD (computer aided design) and CAE (computer aided engineering). Currently, numerical simulation is a hot topic in this field. In numerical simulation, a mathematical

model based on physical-chemical phenomena in die casting process is set up. A collection of process data (including casting and gating system geometrical characteristics, material thermal properties and process parameters) is input into the computer then the simulation program is run and finally the result (prediction) is output (Chen, et. al. 1990, Rosbrook, et. al. 2000).

In the early 1960s, shortly after the appearance of computers, some researchers began studies on numerical simulation of metal solidification (Wang, 1992). According to the one-dimensional heat equilibrium equation, Tuten and Kaiser developed a program for optimizing one-dimensional cooling design with programmable calculators TI-58 and HP-67 in 1979 (Tuten, et. al. 1979). Booth and Allsop made a program for computing heat equilibrium of dies and cooling system parameters with a PET personal computer in 1981 (Booth and Allsop, 1981). Granchi computed the heat transfer between planar nodes thus obtained the two-dimensional temperature distribution of die sections by dividing the calculation field, which was based on energy conservation principle (Granchi and Vettori, 1983).

Most of the research on thermal analysis for die casting is focused on transient solution. But in recent years, some research has been carried out on the equilibrium thermal analysis at steady state. The objective is also to increase the efficiency of the simulation without sacrificing too much accuracy. Barone and Caulk (1993) presented a method to calculate the periodic die temperature at steady state without solving for the start-up transient. Rosidale and Davey's (1998) addressed the thermal behavior of the injection system of hot chamber machines. Their thermal analysis model was based on boundary

element method and a series of papers were published. Ma (2000) computed the steady state thermal profile of part and die using purely geometric reasoning method.

In Barone and Caulk's method, some assumptions were made, such as the casting being comprised of piecewise elements with constant element thickness. These assumptions decreased the accuracy of the result. Rosidale and Davey calculated the whole injection system including nozzle and adaptor, which may increase the computation time. They applied boundary element method, where mesh generation may be a difficulty and only the boundary temperature was represented. Ma's method did not consider the actual heat transfer so the accuracy was quite poor. These observations make it necessary to develop a quick and easy-to-use tool to obtain equilibrium temperature with reasonable accuracy.

In the early research on die casting simulation, most of the efforts were placed on thermal profile analysis of the casting or die. But subsequently, some researchers carried out research on mold filling simulation due to the essential effect of the filling process on casting quality. There is not yet a satisfactory model describing the filling process accurately because of the complexity. Zhang (1996) developed a two-phase turbulence model, which successfully simulated some cases of filling process in die casting. Lu and Lee (1999) took the effects of the wall thickness and velocity profile between two bounding walls into account of governing equations. They simulated the metal flow in a cylindrical sleeve cavity using cylindrical coordinates system. Khayat (1998) proposed a three-dimensional boundary element method to confined free-surface flow for die casting. His approach is particularly applicable to flow driven by a plunger in cylindrical cavities. Lee and Sheu (2001) presented a new numerical method for incompressible

viscous free surface flow without smearing the free surface. Both liquid and air are considered as the physical domain in their model. Though many researchers made much progress on their studies, at present, filling process simulation is still a difficulty in die casting simulation.

Some researchers have concentrated on other problems, such as thermal stress, distortion and peripheral equipment. In 1985, Kim and Ruhlanddt (1985) developed a model to compute thermal stress using the Finite Element Method. Rosindale and Davey (1998) studied the thermal behavior of injection system of a hot chamber machine using boundary element method (BEM).

At present, MagmaSoft and Procast are two successful commercial simulation software packages for die casting process. MagmaSoft is based on the Finite Difference Method and Procast is based on the Finite Element Method. ABAQUS is a popular general purpose simulation package. Though it is not particular for die casting, some researchers often apply it for die casting simulation due to its convenience. Flow-3D is another package to simulate general flow problem for engineering. Some die casting engineers also apply Flow-3D for fill pattern. (www.magmasoft.com, www.use-software.com, www.hks.com, www.flow3d.com, 2002).

For thermal analysis, these and other packages compute transient temperature primarily due to the way the problem is set up. Thus, inevitably, start-up phase needs to be computed, which can be very time consuming.

1.3.2 Geometric reasoning

Geometric reasoning is another important research topic in die casting CAD/CAE technology. Since the thermal profile, fill pattern and other process features are closely related to the geometry of casting and die, it is possible to obtain the feature information based on the pure geometric reasoning. Though the analysis accuracy of geometric reasoning is lower than that of numerical simulation, geometric reasoning is particularly important at the early design phase since at this stage, only the geometric characteristics are known and other process parameters, which are usually required by numerical simulation, are still unknown.

The application of geometric analysis for casting process began in the 1940s. At that time, computers had not appeared yet. But the Chvoinov's Rule had been widely known as (www.egr.msu.edu, 2002):

$$t = C(V/A)^2$$

where t is the solidification time for a point in the casting section; C is a constant relating to casting material, mould material and mould temperature; V is the volume of casting section; and A is the surface area of the casting section. Therefore, based on the geometrical characteristic of a casting section, the solidification time for this section can be estimated. Since the requirement for casting design is "sequential solidification" to reduce porosity, the Chvoinov's Rule can help to optimize casting design. A simpler term "modulus" is thus defined as:

$$M = V/A$$

which means the ratio of volume over surface area in 3-D space or surface over perimeter in 2-D space. With the modulus of each casting section, the solidification sequence of the casting can be calculated easily. Though Chvoinov's Rule was originally applied for sand casting, its application for die casting, which constructs the theoretic basis for computational geometric reasoning, is straightforward.

Heine and Uicker (1984) were the pioneers to apply computational geometric reasoning on casting analysis. They published their application and extension of section modulus approach in 1984 and 1985. In the following years, other researchers also carried out the research on geometric reasoning approaches, including Neises, Ravi, Kotschi, Upadhyay (Ma, 2000).

As described in previous section (see Fig. 1.4), there are three tasks for the research in this dissertation, 1) computation of equilibrium temperature of the die and ejection temperature of the part for die casting process; 2) fill pattern algorithm redesign for die casting process. The three projects are sub-projects of the CastView project. This report will discuss the development, implement and analysis results of these three projects.

1.4 Summary

1) Die casting is an important net shape manufacturing process with rapid growth rate of the application in recent years.

2) Thermal characteristics and fill pattern are two common concerns for die casting engineers. The traditional way to obtain the solution is numerical simulation based on conservation equations.

3) Due to the complicated equation solving, numerical simulation is very time consuming and may be not suited for the conceptual design stage.

4) The research in this dissertation is to develop a mathematical model to calculate equilibrium temperature of die and ejection temperature of part. The purpose is to provide a quick tool to obtain the thermal profile of die/part.

5) The algorithms need to be compatible with current CastView program. The fill pattern algorithm should be based on the old fill pattern algorithm which exists in the current CastView.

2. EQUILIBRIUM TEMPERATURE ANALYSIS FOR DIE AND PART

2.1 Introduction

As discussed in the previous chapter, prediction of equilibrium temperature of the die and ejection temperature of the casting is important for die cooling and cycle design. One part of this research is to develop a quick mathematical algorithm for equilibrium temperature computation. The objective is to provide a quick computer tool for steady state thermal analysis while providing reasonable accuracy. The intent is to help speed-up die and part design and help optimize cooling line placement and the casting cycle design. With the analysis results, a user can quickly know where the hot spot is and what the overall temperature looks like. This is very helpful at the conceptual design stage. A typical numerical simulation can also do this but it is too time consuming and the information provided is more than needed. Thus the traditional numerical simulation is not suited for this stage. In addition, the equilibrium results can provide a start-up profile for transient temperature simulation with greatly reducing total simulation time. For example, simulation software may have to finish 100 cycles from initial state to reach quasi steady state for analysis of thermal stress or distortion, which may take days of computation. However, if the simulation starts from equilibrium temperature profile, it may need only 10 cycles to reaches steady state, which takes much less time. This chapter discusses the algorithm to compute the equilibrium temperature in CastView.

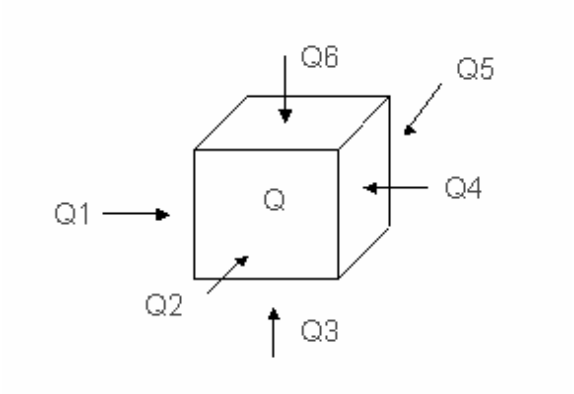
2.2 Heat Balance for Equilibrium Process

The numerical method selected is based on several considerations: 1) Compatibility with available software; 2) Speed; 3) Computer memory requirements. This study is a component of the existing CastView project thus compatibility with the CastView software is very important. Furthermore, pre-processor and post-processor functions of CastView can be used directly for thermal analysis. The CastView software uses the voxel model (a collection of uniform size cubes) to represent 3D geometric solids of die, part or other components. The voxel array is very similar to a finite difference grid. Therefore, finite difference method is employed as the numerical method.

If we consider a voxel in a true equilibrium process, the heat flows into this voxels plus the heat that it generates equals to the heat it flows out due to the energy balance. The die casting process is a quasi steady state and not in a true equilibrium. However, if we consider the whole cycle period, the heat balance still holds. The heat flows in plus heat generated should equal to heat flows out for a voxel when the process reaches quasi steady state. Since we define the equilibrium temperature as time average temperature over the cycle, we can establish the energy balance equations by applying the equilibrium temperature.

Considering the heat transfer in an interior voxel (Fig. 2.1) in the equilibrium state, the well-known energy balance equation is:

Q1+Q2+…+Q6+Q=0 (2-1a)



Section I-Figure 2.1: Heat balance for a voxel. Q1 ~ Q6 are heat from the 6 neighboring voxels and Q is heat released from current voxel.

Q1 ~ Q6 are heat transferred from the 6 neighboring voxels and Q is heat released from the voxel itself. In other words, in equilibrium the heat flowing into a voxel plus the heat released must be balanced by the total heat flowing out of the voxel.

Considering the heat flow from face 1, we can express the heat amount in the whole cycle using time average temperature:

$$Q_1 = \int_{t_0}^{t_{cycle}} \frac{(T - T_1)}{R_1} dt = \frac{\overline{T} - \overline{T_1}}{R_1}(t_{cycle} - t_0) \quad \text{(2-1b)}$$

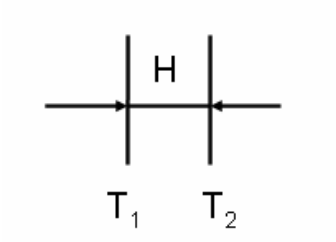
Where T and T_1 are the transient temperatures of the target voxel and the neighboring voxel. R1 is thermal resistance of the target voxel and the neighboring voxel. $\overline{T}$ and $\overline{T}_1$ are time average temperatures of target voxel and neighboring voxel. Equation (2-1b) is

for an interior voxel only where R_1 does not change over the cycle. For the voxels at interface where heat transfer condition changes during the cycle, the heat flow needs special treatment which will be discussed in later section.

Equation (2-1b) is for heat flowing from a face. If we consider the 6 faces of the target voxel, we will have the sum that heat flows in. Thus Eq. (2-1a) can be written as (after rearrangement):

$$\frac{\overline{T}-\overline{T}_1}{R_1}+\frac{\overline{T}-\overline{T}_2}{R_2}+\frac{\overline{T}-\overline{T}_3}{R_3}+\frac{\overline{T}-\overline{T}_4}{R_4}+\frac{\overline{T}-\overline{T}_5}{R_5}+\frac{\overline{T}-\overline{T}_6}{R_6}=\dot{q} \tag{2-2}$$

Where R_1 ~ R_6 are the thermal resistance between the target voxel and neighboring voxels and $\dot{q}$ is the interior heat rate. The thermal resistance to heat flow is defined by analogy to Ohm's law to current flow. To illustrate the definition of thermal resistance, let's consider a 1D example.



Section I-Figure 2.2: Definition of thermal resistance

In the example, the surface temperatures of the two sides of a wall are T_1 and T_2 with wall thickness H and conductivity *k*. The heat flux is then:

$$q = -k\frac{dT}{dx} = -\frac{k}{H}(T_2 - T_1) \tag{2-3}$$

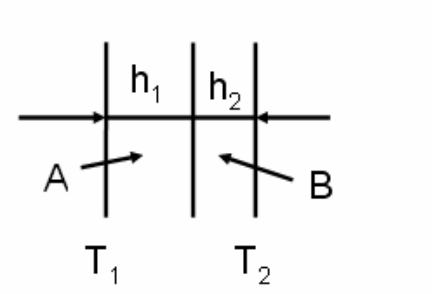
If the surface area is A, in the thermal resistance form, we have,

$$qA = -\frac{T_2 - T_1}{R} \tag{2-4}$$

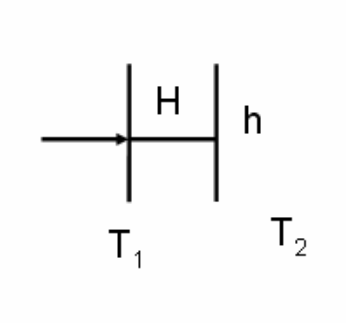
The rate of heat flow is analogous to the current, and the temperature difference is analogous to the potential (voltage) difference. Thus the thermal resistance R can be defined as,

$$R = \frac{H}{Ak} \tag{2-5}$$

It can be seen that to increase the thermal resistance, we can either increase the thickness of material or decrease the thermal conductivity. Using the similar analogy, we can develop heat resistance for a composite wall and complex heat transfer.



Section I-Figure 2.3: A composite wall (without heat transfer coefficient between two materials).



Section I-Figure 2.4: Thermal resistance with heat transfer coefficient at one side.

Suppose we have a composite wall with two different materials A and B shown in Fig. 2.3, the thermal resistance between the two surfaces (at T_1 and T_2) is

$$R = \frac{h_1}{Ak_A} + \frac{h_2}{Ak_B} \tag{2-6}$$

If there is convection at one side as shown in Fig. 2.4 with heat transfer coefficient *h*, the thermal resistance can be written as:

$$R = \frac{H}{Ak} + \frac{1}{Ah} \tag{2-7}$$

This shows that the heat transfer coefficient *h* at the surface contributes a resistance to heat flow given by *1/Ah*. Therefore, if there is heat transfer coefficient *h* between two materials in a composite wall, the thermal resistance should be

$$R = \frac{h_1}{Ak_A} + \frac{h_2}{Ak_B} + \frac{1}{Ah} \tag{2-8}$$

Back to Eq. (2-2), after rearranging, the equation can be rewritten as

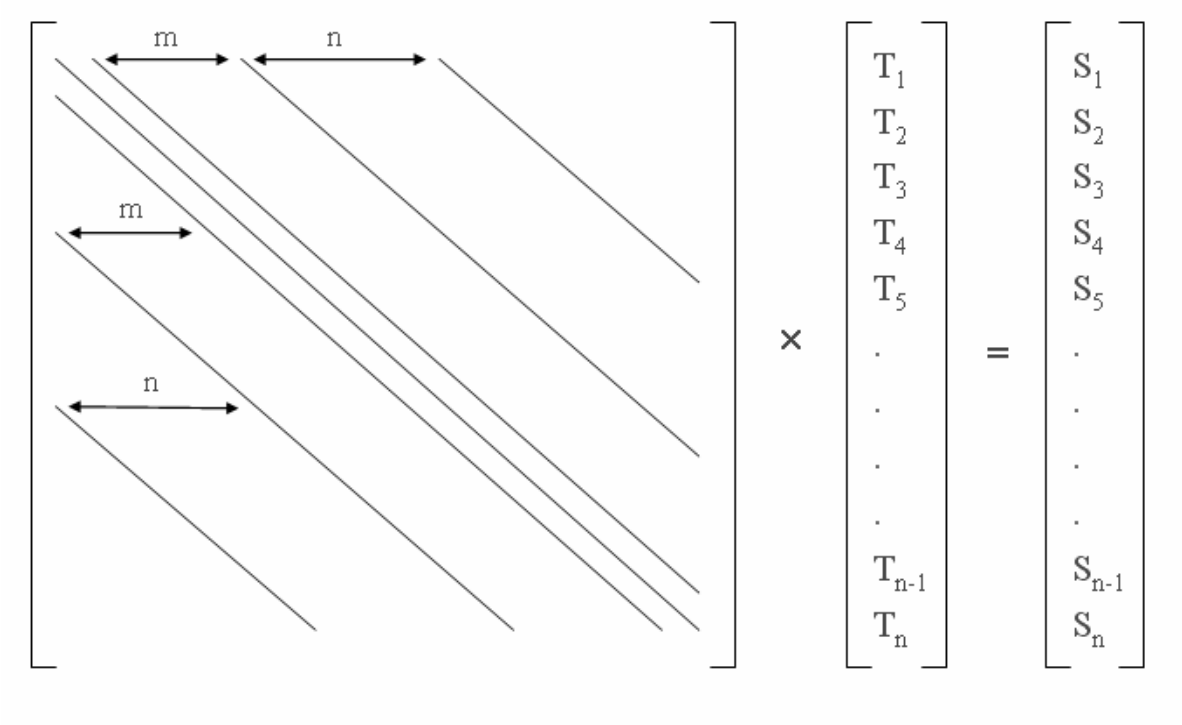
$$-(\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \frac{1}{R_4} + \frac{1}{R_5} + \frac{1}{R_6})\overline{T} + \frac{\overline{T}_1}{R_1} + \frac{\overline{T}_2}{R_2} + \frac{\overline{T}_3}{R_3} + \frac{\overline{T}_4}{R_4} + \frac{\overline{T}_5}{R_5} + \frac{\overline{T}_6}{R_6} = -\dot{q} \tag{2-9}$$

This is actually in the form (note that this is for interior voxels only and exterior voxels will be discussed later in this chapter):

$$a^C\overline{T} + a^L\overline{T}_1 + a^R\overline{T}_4 + a^B\overline{T}_3 + a^T\overline{T}_6 + a^F\overline{T}_2 + a^H\overline{T}_5 = S \tag{2-10}$$

where $\overline{T}$ is the equilibrium temperature of the voxel itself and $\overline{T}_1$ ~ $\overline{T}_6$ are temperatures of the neighboring voxels. *S* is a term collecting all factors that do not depend on temperature and a^C ~ a^H are coefficients that depend on interface and material properties.

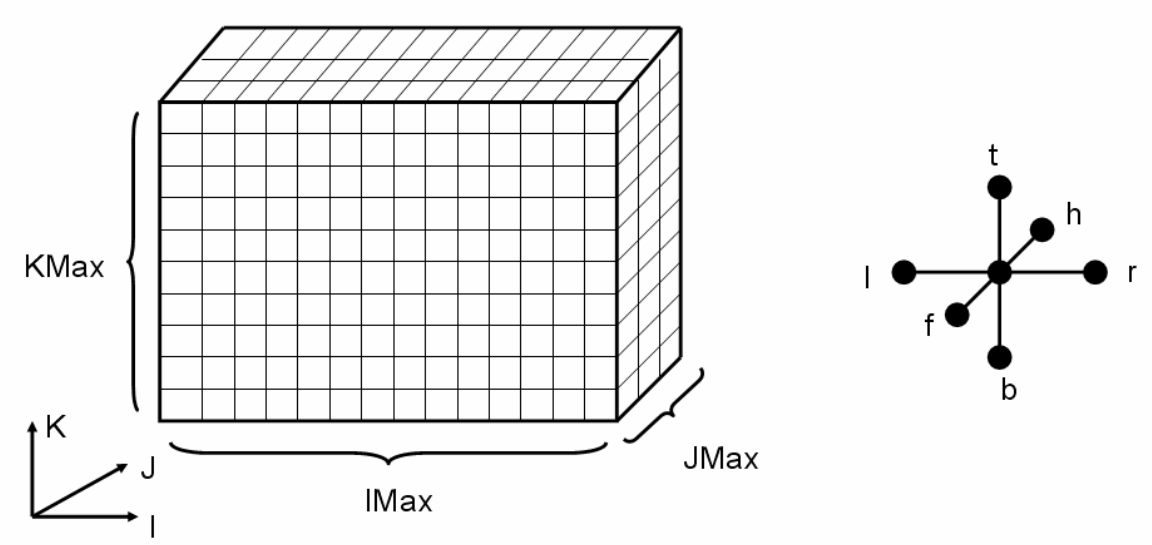
For each voxel, there is an equation of the form of Eq. (2-10) and the total equations for all voxels form a large system of equations (as shown in Fig. 2.5). The values of *m* and *n* depend on the dimensions of computational domain.



Section I-Figure 2.5: Matrix form for equation system. $T_1 \sim T_n$ are unknowns. The values of *m* and *n* depend on the dimensions of computational domain.

The relation of *m* and *n* with the computation domain can be illustrated using Fig. 2.6. In the figure, the three dimensions of the domain are IMax, JMax and KMax. This is a 3D array but we will use a 1D array to represent the voxels. We choose the marching direction on the three directions as I -> J -> K for voxel index. This means we start from (0, 0, 0) and walk to right until we reach (IMax-1, 0, 0). We then go to next row, starting from (0, 1, 0) and walk to (IMax-1, 1, 0). This procedure continues until we finish the

first layer where K = 0. Then we start the second layer K = 1. This marching scheme builds a relation between 1D array and 3D array. Considering a target voxel with 1D index *p* and the neighboring 6 voxels, the indices of left, right, front, back, bottom and top voxels are *p - 1, p + 1, p – IMax, p + IMax, p – IMax*JMax* and *p + IMax*JMax*. Thus the values of *m* and *n* in Fig. 2.5 are *IMax -1* and *IMax*JMax – IMax -1*.



Section I-Figure 2.6: Computational domain and neighboring voxels around the target voxel. The labels of l, r, t, b, f, h represent the left, right, top, bottom, front and back voxels.

Many existing solvers can solve the equation system (Fig. 2.5) in various ways such as Strongly Implicit Procedure (SIP), Alternating Direction Implicit (ADI) and CGSTAB (Ferziger and Peric, 1999). However, Successive Over Relaxation (SOR) is particularly suitable considering memory requirements. SOR has the simple iterative form:

$$\overline{T}_{i,j,k}^{(t)} = \frac{\omega[S_{i,j,k} - (a^L \overline{T}_{i-1,j,k}^{(t)} + a^R \overline{T}_{i+1,j,k}^{(t-1)} + a^B \overline{T}_{i,j,k-1}^{(t)} + a^T \overline{T}_{i,j,k+1}^{(t-1)} + a^F \overline{T}_{i,j-1,k}^{(t)} + a^H \overline{T}_{i,j+1,k}^{(t-1)})]}{a^C} + (1-\omega)\overline{T}_{i,j,k}^{(t-1)} \quad (2\text{-}11)$$

where ω is the over relaxation factor which must be greater than 1 for acceleration and *t* is the iteration counter and i, j, k are voxel position indices. The calculation continues for each voxel until it converges. SOR is a fast solver (though it is not the fastest one) and requires the minimum memory. Compared to other solvers, SOR only needs to store and use the non-zero coefficients ($a^C \sim a^H$), which increases the computation capability significantly.

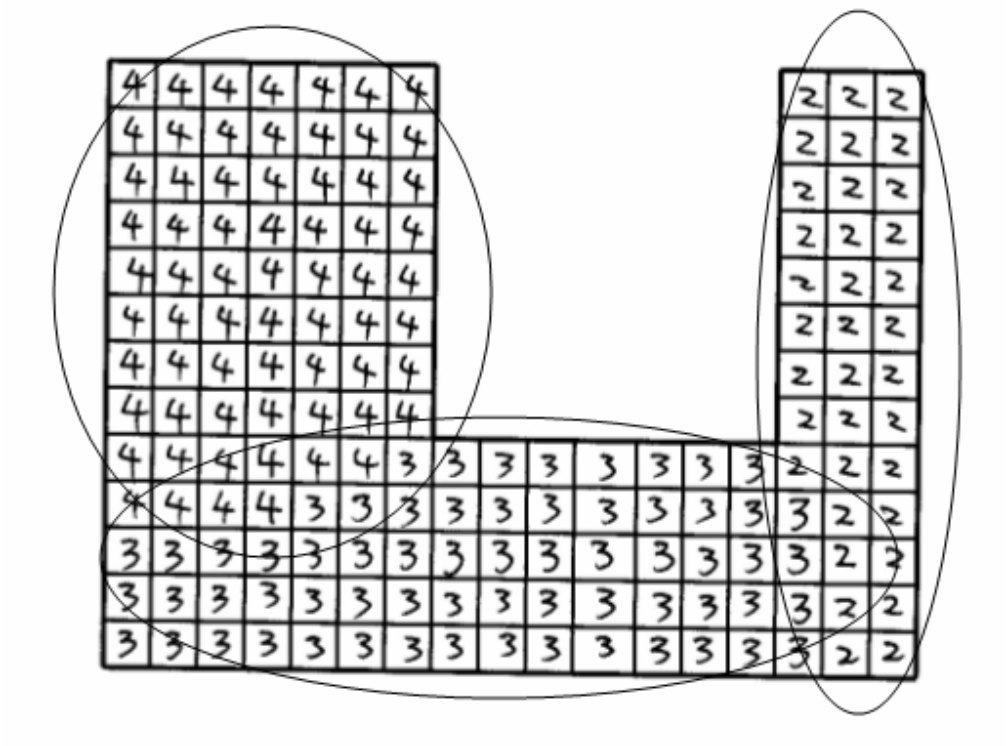
2.3 Models for Heat Source Calculation

The heat source of equilibrium is the heat released from part during the solidification or possible heating and cooling lines. The heat released from a part voxel is not independent of die, cooling, spray and other factors. Therefore, one difficulty is how to model the heat released from an individual part voxel. Several attempts were made to approximate the heat amount released, including a heat flux model, a skeleton model, a linear average model, an asymptotic model and a surrogate model. The combination of asymptotic model and surrogate model was finally selected for the heat source calculation.

In the heat flux model, the part is first divided into regions according to the wall thickness. In CastView, there is a computation module named thick section analysis to calculate the wall thickness of the part. As shown in Fig. 2.7, the part is divided into three regions, with wall thickness of 2, 3 and 4. It is assumed that at each region surface, the

heat flux from part to die is constant. The procedure to calculate heat flux for each region is as follows. 1) A global ejection temperature is assumed, which means all part voxels are ejected at the same temperature. 2) Calculate voxel number and surface area for each region. 3) Based on these values, calculate the heat released from each region and the average heat flux from region surface.

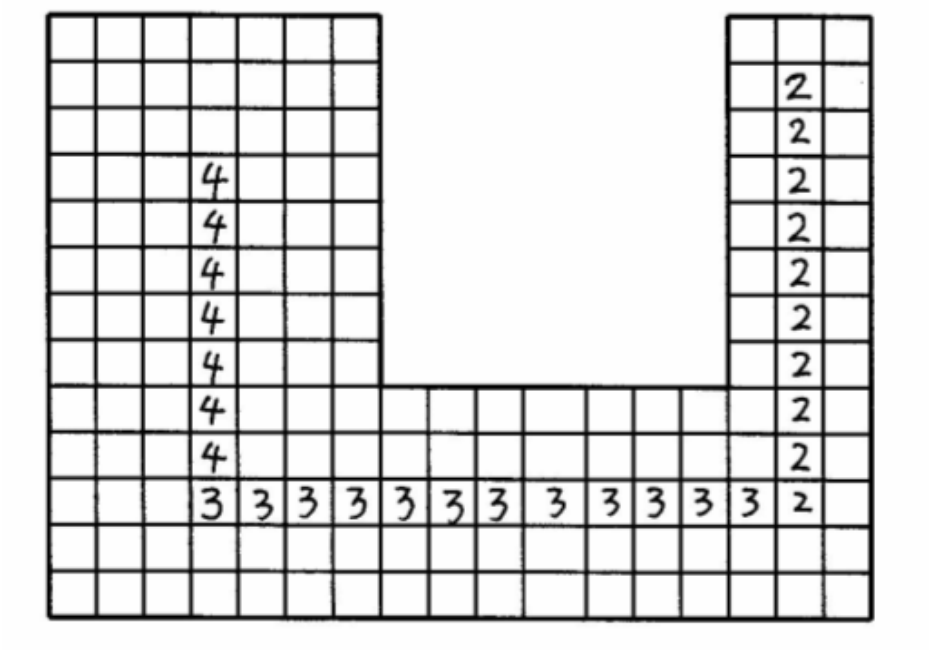
However, there are two disadvantages for this method: 1) The part temperature cannot be calculated; 2) A global ejection temperature is assumed. 3) The heat flux is also related to die geometry even the part thickness is the same. For example, considering the left side and right side of region with wall thickness of 4, as shown in Fig. 2.7, the heat flux from left side should be larger than that from right side because the die temperature at left side is lower than that at right side. Therefore, the assumption is not correct even for a simple case like that in Fig. 2.7.



Section I-Figure 2.7: Illustration of heat flux model. The part is divided into three regions according to part wall thickness.

In the skeleton model, it is assumed that only the skeleton voxel of the part can release heat. Like heat flux model, the part is also divided into regions according to the wall thickness. The skeleton line (also known as the central voxels of a region) is found for each region using a method calling "onion peeling", which actually peels the voxel model from surface to the center. In CastView, there is a computation module named thin section analysis to perform this calculation. As shown in Fig. 2.8, the skeleton line is found and marked with the wall thickness. The heat released from each skeleton voxel thus can be calculated. The procedure for this is as follows. 1) Divide the part into regions according to wall thickness. 2) Calculate the skeleton lines. 3) Calculate voxel number and the number of skeleton voxels for each region. 4) Assume a global ejection

temperature and calculate total heat released from each region. 5) Assign the heat to skeleton voxel of each region. This method can overcome the disadvantages of unbalance heat flux in heat flux model. However, it still makes an incorrect assumption for a global ejection temperature. Therefore, the result applying skeleton model is not good. .



Section I-Figure 2.8: Illustration of skeleton model. The part is divided into three regions according part wall thickness and the skeleton for each region is computed.

The shortcoming of these two models led to seeking methods linking the heat source and part average temperature. During the calculation, the approximate equilibrium temperature (time average temperature) of each part voxel is known. This equilibrium temperature should have a relation to part ejection temperature, which determines the heat released by each part voxel. The linear average model is a simple attempt for this

idea. In this model, it is assumed that the equilibrium temperature of a part voxel is simply the mid point of the injection temperature and ejection temperature, i.e.

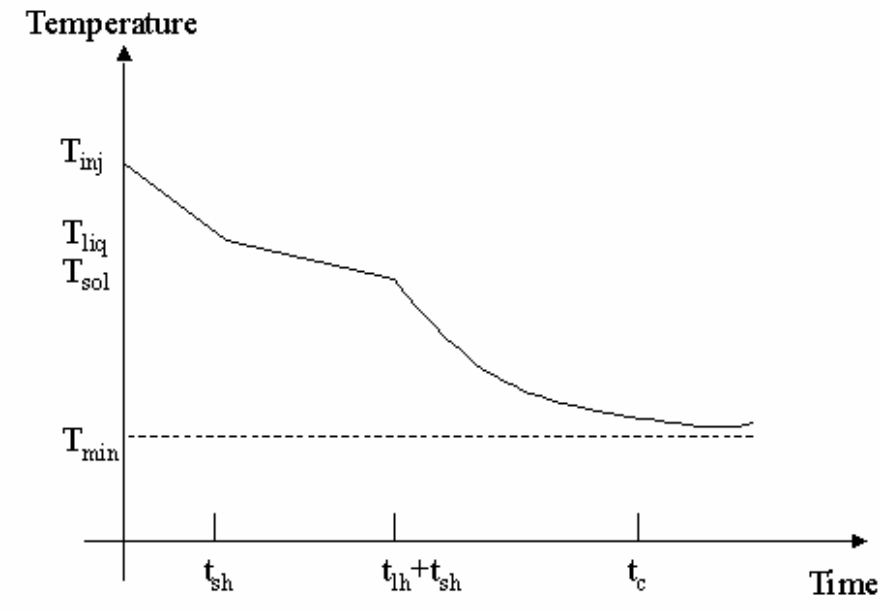
$$T_{ave} = \frac{(T_{inj} + T_{ej})}{2} \tag{2-12}$$

The mid point actually comes from the average assuming temperature in a linear formation of time. Thus the heat released by each part voxel can be easily calculated.

The linear average model is a good attempt but obviously the relation between equilibrium temperature and ejection temperature is not that simple. This led to the asymptotic model.

2.4 Asymptotic Model

The basic idea of this model is to mimic the general form of the typical cooling curve for a point in the casting and assume a lower bound for the ejection temperature as shown in Fig. 2.9. The minimum temperature of lower bound must be equal to the temperature of the environment or higher. If the cycle is very long, the ejection temperature will approach the environment temperature but for a practical cycle time, a higher temperature is reasonable.



Section I-Figure 2.9: Temperature change of a part voxel in a cycle. The part may be ejected when its temperature is at any point on the curve.

For a part voxel, if the ejection temperature is below solidus temperature, the released heat during solidification and cooling can be represented by Eq. (2-13), which says that the heat released by a voxel over the cycle is composed of the super heat, the latent heat, and the heat released due to cooling after solidification.

$$Q' = \frac{\rho * V}{t_{cycle}} * \left(c_p * [(T_{inj} - T_{ej})] + \lambda\right) \qquad (2\text{-}13)$$

Where,

Q' – heat rate, (Watts)

ρ -- density, (kg/m^3)

t_{cycle} -- cycle time, (sec)

V -- volume of the part voxel, (m^3)

c_p -- specific heat, (J/kg/^{o}C)

T_{inj} -- injection temperature, (^{o}C)

T_{liq} -- liquidus temperature, (^{o}C) T_{sol} -- solidus temperature, (^{o}C)

T_{ej} -- ejection temperature, (^{o}C) λ -- latent heat, (J/kg)

For the ejection temperature of a part voxel, there are three possibilities: $T_{ej} > T_{liq}$, $T_{liq} > T_{ej} > T_{sol}$ and $T_{ej} < T_{sol}$. The relationship between T_{ej} and $\overline{T}$ can be set up for each case.

1) $T_{ej} > T_{liq}$

The temperature is assumed to change in a linear way in this region. Hence the relationship is:

$$\overline{T} = \frac{T_{inj} + T_{ej}}{2} \tag{2-14}$$

$$T_{ej} = 2 * \overline{T} - T_{inj} \tag{2-15}$$

2) $T_{liq} > T_{ej} > T_{sol}$

It is assumed that the heat rate is constant in this period, i.e., $dQ / dt = C$. A little calculus and algebra leads to:

$$T_{ej} = 2 * \overline{T} * \frac{Q}{Q - Q_{sh}} - (T_{inj} + T_{liq}) * \frac{Q_{sh}}{Q - Q_{sh}} - T_{liq} \tag{2-16}$$

$$Q_{sh} = \rho * v * c_p * (T_{inj} - T_{liq}) \tag{2-17}$$

$$Q = Q_{sh} + \frac{\rho * V * c_p * \lambda * \{\sqrt{(T_{liq} - \overline{T})^2 + [(T_{inj} + T_{liq} - 2 * \overline{T}) * (T_{inj} - T_{liq})] * (T_{liq} - T_{sol}) * \frac{c_p}{\lambda}} + T_{liq} - \overline{T}\}}{c_p * (T_{liq} - T_{sol})} \quad (2\text{-}18)$$

Where Q is total heat released by this voxel and Q_{sh} represents its super heat.

3) $T_{ej} < T_{sol}$

In this period, it is assumed that rate of temperature change is proportional to its difference with the minimum temperature, i.e. Eq. (2-19) describes the time relationship between average temperature and minimum temperature and the proportionality constant is unknown.

$$\frac{dT(t)}{dt} = \alpha * (T(t) - T_{\min}) \text{ with initial condition } T(t_{sh} + t_{lh}) = T_{sol} \quad (2\text{-}19)$$

From this assumption, the ejection temperature is found as a function of other parameters by solving the equation resulting in Eq. (2-19).

$$T_{ej} = e^{-\alpha(t_c - t_{sh} - t_{lh})} * T_{sol} + (1 - e^{-\alpha(t_c - t_{sh} - t_{lh})}) * T_{\min} \quad (2\text{-}20)$$

where,

α -- time constant

t_{sh} -- time to release super heat

t_c -- cooling time (from injection to ejection)

T_{sol} – Solidus temperature

t_{lh} -- time to release latent heat

Starting from this equation and, after considerable algebra, the relationship of average temperature and heat released from the part voxel can be reduced to Eq. (2-21). The key property of Eq. (2-21) is that it relates heat released, *Q*, to the average temperature, exactly the inverse of the relationship needed for the calculation. The heat released by a part voxel can be computed based on the equilibrium temperature of this part voxel. The calculated relation between heat released and ejection temperature using asymptotic model can be illustrated using Fig. 2.20.

$$(\overline{T} - T_{\min}) = \frac{(\overline{T}_{sh} - T_{\min}) * Q_{sh} + (\overline{T}_{lh} - T_{\min}) * Q_{lh} + (Q - Q_{sh} - Q_{lh}) * (T_{sol} - T_{\min})}{(Q_{sh} + Q_{lh}) - (Q_{\max} - Q_{sh} - Q_{lh}) * \ln[\frac{(Q_{\max} - Q)}{(Q_{\max} - Q_{sh} - Q_{lh})}]} \qquad (2\text{-}21)$$

$\overline{T}_{sh}$ -- average temperature for release of superheat phase

$\overline{T}_{lh}$ -- average temperature for release of latent phase

Q_{max} -- maximum heat that can be released (if ejection temperature is the minimum temperature)

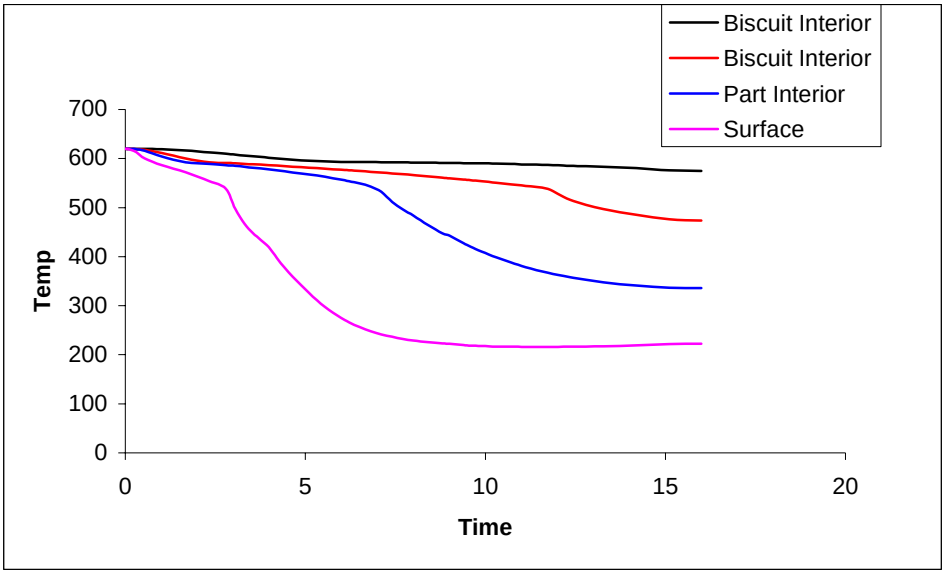
2.5 Surrogate Model for Ejection Temperature

A drawback of the asymptotic model discussed previously is that a minimum temperature is assumed. This temperature is the lower bound of the ejection temperature. Theoretically, this minimum temperature should be the environment temperature since the ejection temperature will never be lower than the environment temperature. However,

it is found from computational experience that the environment temperature does not give a good result for the equilibrium temperature. The typical values for minimum temperatures in calculation of equilibrium temperature are 150 ^{o}C for an aluminum part and 130 ^{o}C for a zinc part. Occasionally, the minimum temperature needs to be adjusted after the first try.

In addition, the “optimum” minimum temperature varies with part geometry and heat transfer conditions. An example is shown in Fig. 2.10, which comes from numerical simulation. There are four temperature change curves over a cycle, corresponding to 4 different points on a part. It can be seen that at the ejection, the ejection temperatures of 4 points are very different. This indicates that the asymptotic minimum temperature should be locally defined because the asymptote is a pseudo asymptote established when the initial temperature difference is reduced. This “asymptote” will slowly decrease with time as the die cools.

A surrogate model has been developed to overcome this problem, i.e. to remove the minimum temperature assumption for the calculation.



Section I-Figure 2.10: Temperature change curves over a cycle for 4 different points at the part, which are from Abaqus numerical simulation.

The rate of change of heat content equation for any voxel other than a part voxel during the period of time latent heat is being released can be described in general terms as follows:

$$\frac{dQ(t)}{dt} = \rho V C_p \frac{dT_{ijk}(t)}{dt} = \begin{Bmatrix} \psi_{i+1jk}\left(T_{i+1jk}(t) - T_{ijk}(t)\right) + \psi_{i-1jk}\left(T_{i-1jk}(t) - T_{ijk}(t)\right) + \\ \psi_{ij+1k}\left(T_{ij+1k}(t) - T_{ijk}(t)\right) + \psi_{ij-1k}\left(T_{ij-1k}(t) - T_{ijk}(t)\right) + \\ \psi_{ijk+1}\left(T_{ijk+1}(t) - T_{ijk}(t)\right) + \psi_{ijk-1}\left(T_{ijk-1}(t) - T_{ijk}(t)\right) \end{Bmatrix} \quad (2\text{-}22)$$

The terms ψ_{lmn} in (2-22) are functions of heat transfer coefficients and conductivities depending on the specific interface involved. The subscripts index the target voxel relative to its neighbors in the x, y and z directions. This equation describes the relation

of heat change and temperature difference as a function of the temperature of the neighboring voxels. After rearrangement, the equation (2-22) becomes

$$\frac{dT_{ijk}(t)}{dt} = \frac{1}{\rho\delta^3 C_p}\begin{Bmatrix} -\left(\psi_{i+1\,jk} + \psi_{i-1\,jk} + \psi_{ij+1k} + \psi_{ij-1k} + \psi_{ijk+1} + \psi_{ijk-1}\right)T_{ijk}(t) \\ +\psi_{i+1\,jk}T_{i+1\,jk}(t) + \psi_{i-1\,jk}T_{i-1\,jk}(t) + \psi_{ij+1k}T_{ij+1k}(t) \\ +\psi_{ij-1k}T_{ij-1k}(t) + \psi_{ijk+1}T_{ijk+1}(t) + \psi_{ijk-1}T_{ijk-1}(t) \end{Bmatrix} \tag{2-23}$$

Using simpler notation, we can rewrite equation (2-23) as:

$$\frac{dT_{ijk}(t)}{dt} = \begin{Bmatrix} -\left(\lambda_{i+1\,jk} + \lambda_{i-1\,jk} + \lambda_{ij+1k} + \lambda_{ij-1k} + \lambda_{ijk+1} + \lambda_{ijk-1}\right)T_{ijk}(t) \\ +\lambda_{i+1\,jk}T_{i+1\,jk}(t) + \lambda_{i-1\,jk}T_{i-1\,jk}(t) + \lambda_{ij+1k}T_{ij+1k}(t) + \lambda_{ij-1k}T_{ij-1k}(t) + \lambda_{ijk+1}T_{ijk+1}(t) + \lambda_{ijk-1}T_{ijk-1}(t) \end{Bmatrix}$$

(2-24)

If we define

$$\underline{\boldsymbol{\lambda}}^T = \begin{bmatrix} \lambda_{i+1\,jk} & \lambda_{i-1\,jk} & \lambda_{ij+1k} & \lambda_{ij-1k} & \lambda_{ijk+1} & \lambda_{ijk-1} \end{bmatrix}$$

$$\underline{\mathbf{T}}_{ijk_n}^T = \begin{bmatrix} T_{i+1\,jk} & T_{i-1\,jk} & T_{ij+1k} & T_{ij-1k} & T_{ijk+1} & T_{ijk-1} \end{bmatrix}$$

Then we have a simple first order matrix equation (symbol $\underline{\mathbf{1}}$ denotes a column vector of 1's):

$$\frac{dT_{ijk}(t)}{dt} = -\underline{\boldsymbol{\lambda}}^T \underline{\mathbf{1}} T_{ijk}(t) + \underline{\boldsymbol{\lambda}}^T \underline{\mathbf{T}}_{ijk_n}(t) \tag{2-25}$$

The differential equation for the part is a little more complex because of the latent heat release. We will use the concept of enthalpy, where the enthalpy content per unit volume of material is defined as (f_l is the fraction liquid):

$$h = \int_0^T C_p(T)dT + Lf_l(T) \tag{2-26}$$

If the specific heat is assumed constant with temperature, the above equation can be written as:

$$h = C_p T + Lf_l(T) \tag{2-27}$$

Thus:

$$\rho_p \delta^3 C_p \frac{dT_{ijk}(t)}{dt} + \rho_p \delta^3 L \frac{\partial f_l(T(t))}{\partial t} = \begin{Bmatrix} \psi_{i+1\,jk}\left(T_{i+1\,jk}(t) - T_{ijk}(t)\right) + \psi_{i-1\,jk}\left(T_{i-1\,jk}(t) - T_{ijk}(t)\right) + \\ \psi_{ij+1k}\left(T_{ij+1k}(t) - T_{ijk}(t)\right) + \psi_{ij-1k}\left(T_{ij-1k}(t) - T_{ijk}(t)\right) + \\ \psi_{ijk+1}\left(T_{ijk+1}(t) - T_{ijk}(t)\right) + \psi_{ijk-1}\left(T_{ijk-1}(t) - T_{ijk}(t)\right) \end{Bmatrix}$$

(2-28)

Assuming that the fraction liquid is a linear function of the temperature difference from liquidus temperature:

$$f_l(T_{ijk}(t)) = \begin{cases} \dfrac{T_{ijk}(t) - T_{sol}}{T_{liq} - T_{sol}} \text{ if } T_{sol} \le T_{ijk}(t) \le T_{liq} \\ 0 \textit{ otherwise} \end{cases} \tag{2-29}$$

Using the chain rule and we have:

$$\frac{dT_{ijk}(t)}{dt} = \frac{C_p * \left(T_{liq} - T_{sol}\right)}{\left(C_p * \left(T_{liq} - T_{sol}\right) + L * \xi(T_{i,j,k}(t))\right)} \begin{Bmatrix} -\left(\lambda_{i+1\,jk} + \lambda_{i-1\,jk} + \lambda_{ij+1k} + \lambda_{ij-1k} + \lambda_{ijk+1} + \lambda_{ijk-1}\right) T_{ijk}(t) \\ +\lambda_{i+1\,jk} T_{i+1\,jk}(t) + \lambda_{i-1\,jk} T_{i-1\,jk}(t) + \lambda_{ij+1k} T_{ij+1k}(t) \\ +\lambda_{ij-1k} T_{ij-1k}(t) + \lambda_{ijk+1} T_{ijk+1}(t) + \lambda_{ijk-1} T_{ijk-1}(t) \end{Bmatrix}$$

(2-30)

$$\text{Where } \xi(T) = \begin{cases} 1 \text{ if } T_{sol} < T < T_{liq} \\ 0 \text{ otherwise} \end{cases} \tag{2-31}$$

Combining (2-25) and (2-31), we have the differential equation that defines a part voxel temperature in the form:

$$\frac{dT(t)}{dt} = \alpha(T(t))\left(-\underline{\boldsymbol{\lambda}}^T \underline{\mathbf{1}} T(t) + \underline{\boldsymbol{\lambda}}^T \underline{\mathbf{T}}_n(t)\right);\ T(t_0) = T_{inj} \tag{2-32}$$

$$\alpha(T(t)) = \frac{C_p(T_{liq} - T_{sol})}{C_p(T_{liq} - T_{sol}) + L} \text{ if part voxel and } T_{sol} \le T(t) \le T_{liq} \tag{2-33}$$

$$\alpha(T(t)) = 1 \text{ otherwise}$$

Thus, if the ejection temperature is above liquidus, we can write

$$T_{eject} - T_{inj} = \int_{t_o}^{t_{eject}} \left(-\underline{\boldsymbol{\lambda}}^T \left(\underline{\mathbf{1}} T(s) - \underline{\mathbf{T}}_n(s) \right) \right) ds \tag{2-34}$$

If it is between liquidus and solidus

$$T_{liq} - T_{inj} = \int_{t_o}^{t_{sh}} \left(-\underline{\boldsymbol{\lambda}}^T \left(\underline{\mathbf{1}} T(s) - \underline{\mathbf{T}}_n(s) \right) \right) ds \tag{2-35}$$

$$T_{eject} - T_{liq} = \left(\frac{C_p \left(T_{liq} - T_{sol} \right)}{C_p \left(T_{liq} - T_{sol} \right) + L} \right) \int_{t_{sh}}^{t_{eject}} \left(-\underline{\boldsymbol{\lambda}}^T \left(\underline{\mathbf{1}} T(s) - \underline{\mathbf{T}}_n(s) \right) \right) ds$$

If it is below solidus

$$T_{liq} - T_{inj} = \int_{t_o}^{t_{sh}} \left(-\underline{\boldsymbol{\lambda}}^T \left(\underline{\mathbf{1}} T(s) - \underline{\mathbf{T}}_n(s) \right) \right) ds \tag{2-36}$$

$$T_{sol} - T_{liq} - \frac{L}{C_p} \frac{\left(T_{liq} - T_{sol} \right)}{\left(T_{liq} - T_{sol} \right)} = \int_{t_{sh}}^{t_{lh}} \left(-\underline{\boldsymbol{\lambda}}^T \left(\underline{\mathbf{1}} T(s) - \underline{\mathbf{T}}_n(s) \right) \right) ds$$

$$T_{eject} - T_{sol} = \int_{t_{lh}}^{t_{eject}} \left(-\underline{\boldsymbol{\lambda}}^T \left(\underline{\mathbf{1}} T(s) - \underline{\mathbf{T}}_n(s) \right) \right) ds$$

Using fraction solid, equation (2-34) ~ (2-36) can be combined as:

$$T_{eject} - T_{inj} = \frac{L}{C_p} f_s + \int_{t_0}^{t_{eject}} -\underline{\boldsymbol{\lambda}}^T \left(\underline{\mathbf{1}} T(s) - \underline{\mathbf{T}}_n(s) \right) ds \tag{2-37}$$

$$T_{eject} - T_{inj} = \frac{L}{C_p} f_s - \underline{\boldsymbol{\lambda}}^T \left(\underline{\mathbf{1}} \overline{T} - \underline{\overline{\mathbf{T}}}_n \right) \left(t_{eject} - t_0 \right) \tag{2-38a}$$

Where f_s is fraction solid. This implies that the change in temperature is proportional to the average difference in termperature between the voxel that is the target of the analysis and the surrounding voxels plus a term proportional to the latent heat (heat source). Equation 2-38 is exact (subject to the assumptions) in the sense that it describes the average temperature as function of the injection and ejection temperatures. If the ejection temperature were known for each voxel, the system of equations built using 2-38 for each voxel would produce the average temperature. The problem is we don't know the ejection temperature. We get around this problem using a surrogate equation.

In Eq. (2-19), T_{min} is an unknown constant and only is applied in the cooling phase. This is modified slightly for the entire interval as an estimator or surrogate for the ejection temperature.

$$\frac{dz(t)}{dt} = -\alpha(z(t))\gamma\left(z(t) - T_c\right);\ z(t_0) = T_{inj} \tag{2-39}$$

with T_c an unknown constant. The notation has been changed from the asymptotic model to minimize confusion, where T_c is minimum temperature and z is surrogate of temperature. Equation (2-39) is an approximation of (2-32) but it does not depend on neighboring voxels. We are using a single constant T_c and an unknown constant γ to approximate the relationship with the neighbors.

Requiring the average and ejection temperatures of the actual equation and the surrogate to be the same, ie $\bar{z} = \bar{T}$ and $z_{eject} = T_{eject}$, equation (2-38a) can be written as:

$$z_{eject} - T_{inj} - f_s(z)\frac{L}{C_p} = -\lambda^T(\underline{1}\overline{T} - \underline{\overline{T}}_n)(t_{eject} - t_0) \tag{2-38b}$$

Comparing Eq. (2-39) and Eq. (2-32) both describing temperature change, we know that the right hand sides should be equal, which means $-\lambda^T(\underline{1}\overline{T} - \underline{\overline{T}}_n) = \gamma(z(t) - T_c)$. Therefore, Eq. (2-38b) can be written as:

$$z_{eject} - T_{inj} = f_s(z)\frac{L}{C_p} + \left(-\gamma\bar{z} + \gamma T_c\right)\left(t_{eject} - t_0\right) \tag{2-40}$$

For convenience we will define the function g from Eq. (2-40) with the following notation

$$g(T_c, z_{eject}, \bar{z}) = z_{eject} - T_{inj} - f_s(z)\frac{L}{C_p} + \gamma\left(t_{eject} - t_0\right)\left(\bar{z} - T_c\right) = 0 \tag{2-41}$$

Comparing equation (2-40) and (2-38), if we assume the surrogate is a good approximation, we can assume the left hand sides of both equations are approximately the same. With the assumption $\bar{z} \approx \overline{T}$, we have:

$$-\gamma\left(t_{eject} - t_0\right)\left(\bar{z} - T_c\right) \approx \left(-\underline{\boldsymbol{\lambda}}^T\underline{\mathbf{1}}\overline{T} + \underline{\boldsymbol{\lambda}}^T\underline{\overline{\mathbf{T}}}_n\right)\left(t_{eject} - t_0\right) \tag{2-42}$$

$$T_c = \overline{T} - \frac{\left(-\underline{\boldsymbol{\lambda}}^T\underline{\mathbf{1}}\overline{T} + \underline{\boldsymbol{\lambda}}^T\underline{\overline{\mathbf{T}}}_n\right)\left(t_{eject} - t_0\right)}{-\gamma\left(t_{eject} - t_0\right)} \tag{2-43}$$

From this equation, we define a function *h* as

$$h(T_c, z_{eject}, \overline{T}, \underline{\overline{\mathbf{T}}}_n) = T_c - \overline{T} + \frac{\left(-\underline{\boldsymbol{\lambda}}^T \underline{\mathbf{1}}\overline{T} + \underline{\boldsymbol{\lambda}}^T \underline{\overline{\mathbf{T}}}_n\right)\left(t_{eject} - t_0\right)}{-\gamma\left(t_{eject} - t_0\right)} = 0 \tag{2-44}$$

Similar to asymptotic model, there are three cases for the *h* and *g* functions for ejection temperature corresponding to super heat phase, latent heat phase and cooling phase, which are summarized in Table 1.1 and Table 1.2.

Section I-Table 2-1: Ejection Temperature Relationships

Condition	**Ejection Temperature (source terms),** $g(T_c, z_{eject}, \overline{T})$
$z_{eject} \geq T_{liq}$	$z_{eject} - T_{inj} - \left(\overline{T} - T_c\right)\left[\ln\left(z_{eject} - T_c\right) - \ln\left(T_{inj} - T_c\right)\right] = 0$
$T_{sol} \leq z_{eject} \leq T_{liq}$	$z_{eject} - T_{inj} - \left(\frac{T_{liq} - z_{eject}}{T_{liq} - T_{sol}}\right)\frac{L}{C_p} - \left(\overline{T} - T_c\right)\{\ln\left(z_{eject} - T_c\right) - \ln(T_{inj} - T_c)$ $+ \frac{L}{C_p(T_{liq} - T_{sol})}[\ln(z_{eject} - T_c) - \ln(T_{liq} - T_c)]\} = 0$
$z_{eject} \leq T_{sol}$	$z_{eject} - T_{inj} - \frac{L}{C_p} - \left(\overline{T} - T_c\right)\{\ln\left(z_{eject} - T_c\right) - \ln(T_{inj} - T_c)$ $+ \frac{L}{C_p(T_{liq} - T_{sol})}[\ln(T_{sol} - T_c) - \ln(T_{liq} - T_c)]\} = 0$

Section I-Table 2-2: Surrogate Constant Relationships

Condition	Ejection Temperature (source terms), $g(T_c, z_{eject}, \overline{T})$
$z_{eject} \geq T_{liq}$	$T_c - \overline{T} + \dfrac{\left(-\underline{\boldsymbol{\lambda}}^T \underline{\mathbf{1}}\overline{T} + \underline{\boldsymbol{\lambda}}^T \overline{\underline{\mathbf{T}}}_n\right)\left(t_{eject} - t_0\right)}{\ln\left(z_{eject} - T_c\right) - \ln\left(T_{inj} - T_c\right)} = 0$
$T_{sol} \leq z_{eject} \leq T_{liq}$	$T_c - \overline{T} + \dfrac{\left(-\underline{\boldsymbol{\lambda}}^T \underline{\mathbf{1}}\overline{T} + \underline{\boldsymbol{\lambda}}^T \overline{\underline{\mathbf{T}}}_n\right)\left(t_{eject} - t_0\right)}{\ln\left(z_{eject} - T_c\right) - \ln\left(T_{inj} - T_c\right) + \dfrac{L}{C_p\left(T_{liq} - T_{sol}\right)}\left[\ln\left(z_{eject} - T_c\right) - \ln\left(T_{liq} - T_c\right)\right]} = 0$
$z_{eject} \leq T_{sol}$	$T_c - \overline{T} + \dfrac{\left(-\underline{\boldsymbol{\lambda}}^T \underline{\mathbf{1}}\overline{T} + \underline{\boldsymbol{\lambda}}^T \overline{\underline{\mathbf{T}}}_n\right)\left(t_{eject} - t_0\right)}{\ln\left(z_{eject} - T_c\right) - \ln\left(T_{inj} - T_c\right) + \dfrac{L}{C_p\left(T_{liq} - T_{sol}\right)}\left[\ln\left(T_{sol} - T_c\right) - \ln\left(T_{liq} - T_c\right)\right]} = 0$

The curves shown in Fig. 2.10 were used to test the surrogate equation as an approximation. This test is not a computational test but a test of the approximation and demonstrates that the surrogate, in principle, is a good approximation. It does not demonstrate that a good result can be computed in our equilibrium temperature analysis.

The test actually substitutes the numerical simulation result back into the surrogate model and computes the temperature response with time. The procedure used is as follows:

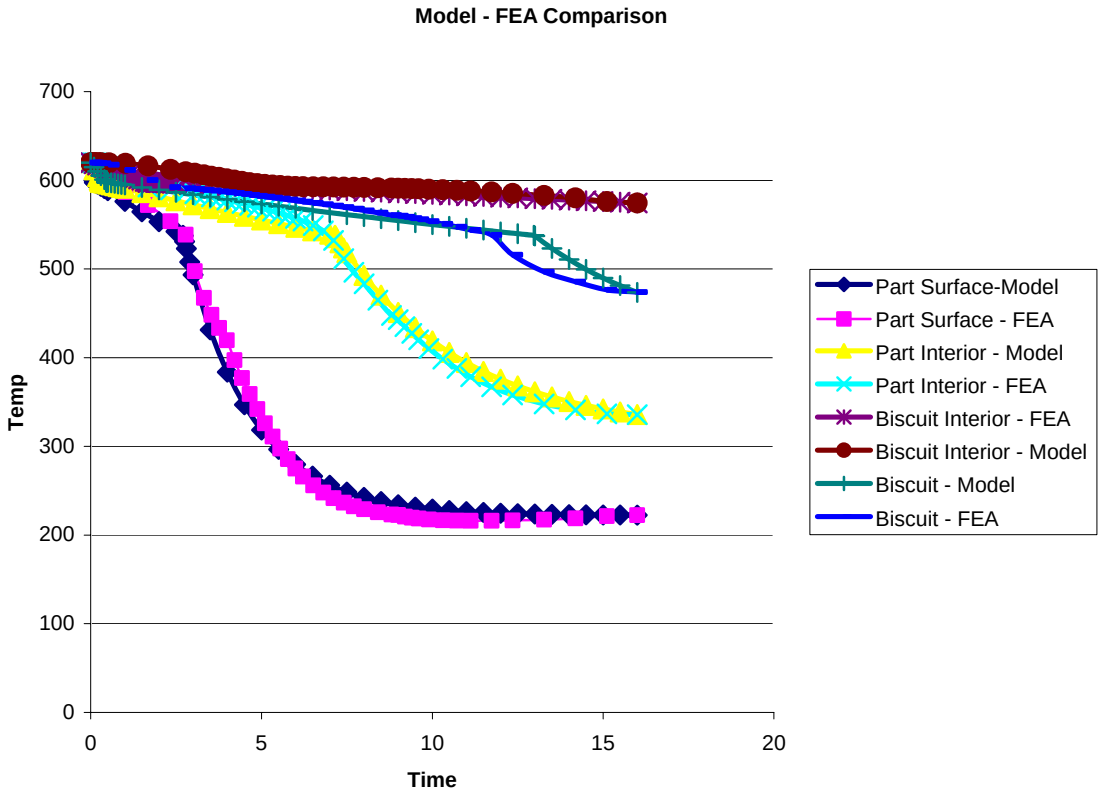
1) A table of $\overline{z}$ values was computed as a function of z_{eject} and T_c using the g function.

2) Given the ejection temperature and the average temperature from numerical simulation result, a value for T_c can be determined.

3) Once values for T_c and z_{eject} are available, γ can be computed since t_{eject} is known.

4) The time response can then be computed from the solutions of the differential equation.

The results are shown in Fig. 2.11. It can be seen that the two sets of curves match very well. This means that if we know the ejection temperature, average temperature and ejection time, we can compute the temperature change curve using surrogate model. The test shows that the surrogate model is a good approximation. However, since in our computation for equilibrium temperature, we do not have ejection temperature and average temperature (they are to be solved), the surrogate model does not guaranty good results.



Section I-Figure 2.11: Test of surrogate equation using data generated by numerical simulation. This is to demonstrate in principle the surrogate model is a good approximation

2.6 Implementation of Surrogate Model

There are two functions $h(T_c, z_{eject}, \overline{T}, \underline{\overline{\mathbf{T}}}_n) = 0$ and $g(T_c, z_{eject}, \overline{T}) = 0$ in the surrogate model, which are listed in table 2-1 and 2-2. During the calculation, the estimated average temperature $\overline{T}$ for each voxel (element) is available. The *h* and *g* functions then need to be solved to obtain T_c and z_{eject} for each voxel, corresponding to minimum temperature and ejection temperature. The heat released then can be calculated with ejection temperature, which can be used to update the average temperature $\overline{T}$. Such a

procedure is continued until the computation converges. To obtain the initial average temperature $\overline{T}$, the asymptotic model is applied for the first few steps, say, 20 steps with an assumed minimum temperature. This assumed temperature is not important because it is for the initial guess for surrogate model and will be adjusted by surrogate model in later calculations.

The procedure to solve *h* and *g* functions for each voxel at each step is to solve two non-linear equations simultaneously, which is much more difficult than solving linear equations. The first attempt is Newton-Raphson method which comes from Taylor expansion. The Taylor series at first order for *h* function is:

$$0 = h(T_c, z_{eject}, \overline{T}) \approx h(T'_c, z'_{eject}, \overline{T}) + \frac{\partial h(T_c, z_{eject}, \overline{T})}{\partial T_c} \Delta T_c + \frac{\partial h(T_c, z_{eject}, \overline{T})}{\partial z_{eject}} \Delta z_{eject} \qquad (2\text{-}45)$$

Similar equation can be obtained for *g* function. Both equations can be combined in a matrix form:

$$\begin{bmatrix} h(T_c, z_{eject}, \overline{T}, \underline{\overline{\mathbf{T}}}_n) \\ g(T_c, z_{eject}, \overline{T}) \end{bmatrix} + \begin{bmatrix} \dfrac{\partial h(T_c, z_{eject}, \overline{T}, \underline{\overline{\mathbf{T}}}_n)}{\partial T_c} & \dfrac{\partial h(T_c, z_{eject}, \overline{T}, \underline{\overline{\mathbf{T}}}_n)}{\partial z_{eject}} \\ \dfrac{\partial g(T_c, z_{eject}, \overline{T})}{\partial T_c} & \dfrac{\partial g(T_c, z_{eject}, \overline{T})}{\partial z_{eject}} \end{bmatrix} \begin{bmatrix} \Delta T_c \\ \Delta z_{eject} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad (2\text{-}46)$$

The equations of $\frac{\partial h}{\partial T_c}, \frac{\partial g}{\partial T_c}, \frac{\partial h}{\partial z_{eject}}$ and $\frac{\partial g}{\partial z_{eject}}$ are listed in Eq. (2-47) ~ (2-50):

$$\frac{\partial h(T_c, z_{eject}, \overline{T}, \underline{\overline{T}}_n)}{\partial T_c} = 1 + \frac{-(-\underline{\lambda}^T \underline{1}\overline{T} + \underline{\lambda}^T \underline{\overline{T}}_n)(t_{eject} - t_0)(\frac{-1}{z_{eject} - T_c} + \frac{1}{T_{inj} - T_c})}{(\ln(z_{eject} - T_c) - \ln(T_{inj} - T_c))^2} \tag{2-47}$$

$$\frac{\partial g(T_c, z_{eject}, \overline{T})}{\partial T_c} = (\ln(z_{eject} - T_c) - \ln(T_{inj} - T_c)) - (\overline{T} - T_c)(\frac{-1}{z_{eject} - T_c} + \frac{1}{T_{inj} - T_c}) \tag{2-48}$$

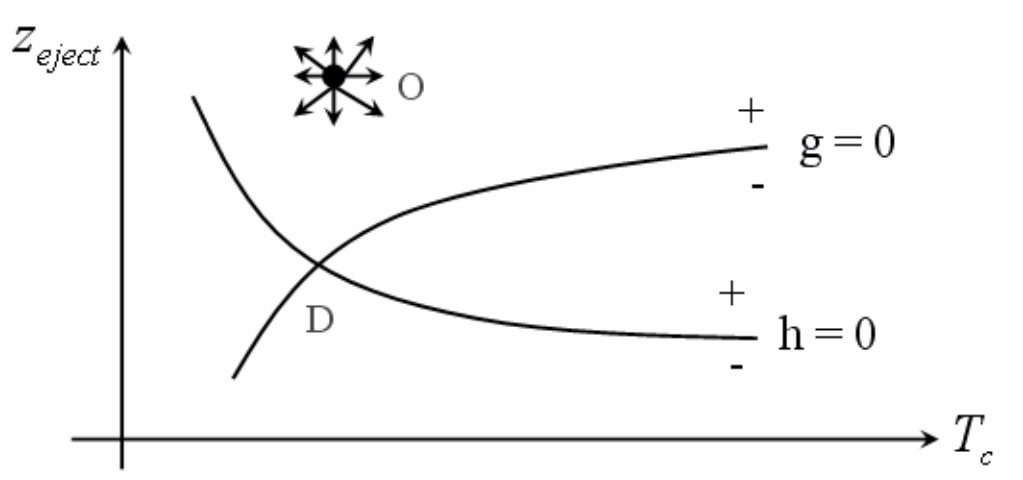
$$\frac{\partial h(T_c, z_{eject}, \overline{T}, \underline{\overline{T}}_n)}{\partial z_{eject}} = \frac{-(-\underline{\lambda}^T \underline{1}\overline{T} + \underline{\lambda}^T \underline{\overline{T}}_n)(t_{eject} - t_0)(\frac{-1}{z_{eject} - T_c})}{(\ln(z_{eject} - T_c) - \ln(T_{inj} - T_c))^2} \tag{2-49}$$

$$\frac{\partial g(T_c, z_{eject}, \overline{T})}{\partial z_{eject}} = \frac{z_{eject} - \overline{T}}{z_{eject} - T_c} \tag{2-50}$$

In this method, the two unknown ΔT_c and Δz_{eject} are solved then T_c and z_{eject} can be updated. However, one problem with this method is that the convergence is slow. Since for each part voxel at each iteration, the *h* and *g* functions need to be solved, the overall convergence is much slower than that in asymptotic model.

Another way is the direct search method. Suppose the *h* and *g* relation are illustrated using Fig 2-12. For the *g* curve, g = 0 only happens on the curve and the upper side is for g > 0 and lower side is for g < 0. The situation is similar for h curve. The intersection point D of the g curve and the h curve is the solution. At one iteration step, our starting point is O and the target is D. The search is started with a small step in each of 8

directions at O, with 45^{o} angle between each two neighboring directions. By taking the minimum value of $|h| + |g|$ for each step, the next point can be found since at point D, the value of $|h| + |g|$ is zero. The search is repeated at the new point and this procedure is continued until the $|h| + |g|$ of the new point is close enough to zero.



Section I-Figure 2.12: Direct search to solve *h* and *g* functions. The starting point is at O and the target is D. Marching is made from O to D by choosing a direction at each step.

The above method uses a fixed step size. By experience, it was found that the number of steps for searching varies with the convergence of global computation. For example, if we choose a fixed step size of 1 ^{o}C, early in the global computation, it usually takes ~6 steps of marching to reach D. As the computation advances, the number of steps to reach D decreases to 3 or 4. When the computation is almost done, there is little marching from O. This observation indicates that at the beginning, the starting point O is far away from D but O is closer and closer to D during the computation though D is not a fixed point.

Based on this observation, the direct approach can be further improved using fewer search directions and changing step size. The procedure is as follows:

1) Choose a relatively large step size at the beginning, say, 2 ^{o}C;

2) Change the step size to 1 ^{o}C, 0.5 ^{o}C and 0.2 ^{o}C when 20%, 50% and 80% of computation is done (this can be measured approximately through the convergence);

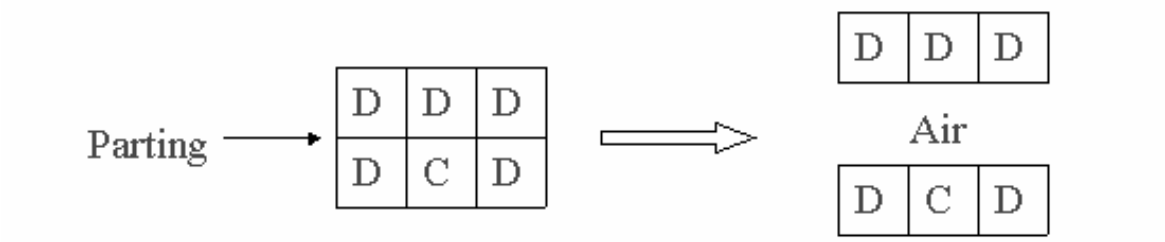
3) Not searching the 8 directions every time except the first time. In other words, the first step is taken by searching the 8 directions. The sequential search is only within last direction and the two neighboring directions. For example, in the first time, we search 8 directions and take the 5th direction. Next time, we only search the 4th, 5th and 6th directions and take the best one.

The direct search method is much faster than Newton-Raphson method and the total run time is only 10 ~ 20% more than that of the pure asymptotic model.

2.7 Composite Heat Transfer at Interface

A typical die casting cycle consists of 6 steps: die close, injection and solidification, die open, ejection, spray and idle. An interface voxel, i.e., a voxel on the parting plane or cavity surface, has different neighbors at different steps during a cycle (Fig. 2.13). The middle die voxel in the upper row of the figure, located on the parting plane surface, has a casting voxel neighbor when the die is closed and has an air voxel neighbor when the die is open. Similar differences occur for any voxel on the parting surface or cavity

surface. Thus, the heat balance equation needs to be modified for all interface voxels to account for the differences in neighbors during the cycle.



Section I-Figure 2.13: A voxel at interface has different neighbors at different step during a cycle, (D depicts die, C casting)

Considering the composite heat over the cycle from both types of neighbors, it can be seen that the total heat is composed of the heat across the interface when the die is closed plus when open, see Eq. (2-51). It was approximated with two different heat transfer coefficients and different neighbor average temperatures and the appropriate average temperatures Eq. (2-52). Dividing Eq. (2-52) by the cycle time gives the heat rate Eq. (2-53) and rearranging terms gives the expression in terms of the average voxel temperature, average part voxel temperature and air temperature Eq. (2-54). Each term is weighted by the fraction of the cycle that it applies. Eq. (2-54) is used in the heat balance equation Eq. (2-10) for all interface voxels. Essentially, a time weight factor is applied for interface voxels instead of the simpler original Eq. (2-10) that is based on fixed conditions between neighbors.

$$Q_{iface} = \left[\int_{t_{closed}} q_{iface,closed}(t)dt + \int_{t_{open}} q_{iface,open}(t)dt \right] * A \quad (2\text{-}51)$$

$$Q_{iface} = h_c * A * (\overline{T}_{i,j,k} - \overline{T}_{casting}) * t_{closed} + h_o * A * (\overline{T}_{i,j,k} - \overline{T}_{air}) * t_{open} \quad (2\text{-}52)$$

$$Q'_{iface} = h_c * A * (\overline{T}_{i,j,k} - \overline{T}_{casting}) * \frac{t_{closed}}{t_{cycle}} + h_o * A * (\overline{T}_{i,j,k} - \overline{T}_{air}) * \frac{t_{open}}{t_{cycle}} \quad (2\text{-}53)$$

$$Q'_{iface} = (h_p * \frac{t_{closed}}{t_{cycle}} + h_o * \frac{t_{open}}{t_{cycle}}) A * \overline{T}_{i,j,k} - h_p * \frac{t_{closed}}{t_{cycle}} * A * \overline{T}_{part} - h_o * \frac{t_{open}}{t_{cycle}} * A * \overline{T}_{air} \quad (2\text{-}54)$$

In these equations:

h_c, h_o -- heat transfer coefficients between die and casting, die and air

$t_{cycle}, t_{open}, t_{close}$ -- cycle time, die-open time and die-close time

Q_{iface}, Q'_{iface} -- heat and heat rate on interface

Not that the step between equation (2-51) and (2-52) is not rigorously correct because $\overline{T}_{i,j,k}$ is determined by the total cycle, whereas the average value for each of the two integrals could be different. The approximation is not unreasonable however.

To obtain the equation in Eq. (2-10) form, let us consider a die voxel with 6 faces which is located at interface. Let us further suppose that face 1 is the interface and the time periods to contact air and part voxel are t_{open} and t_{closed}. The heat balance equation can be written as:

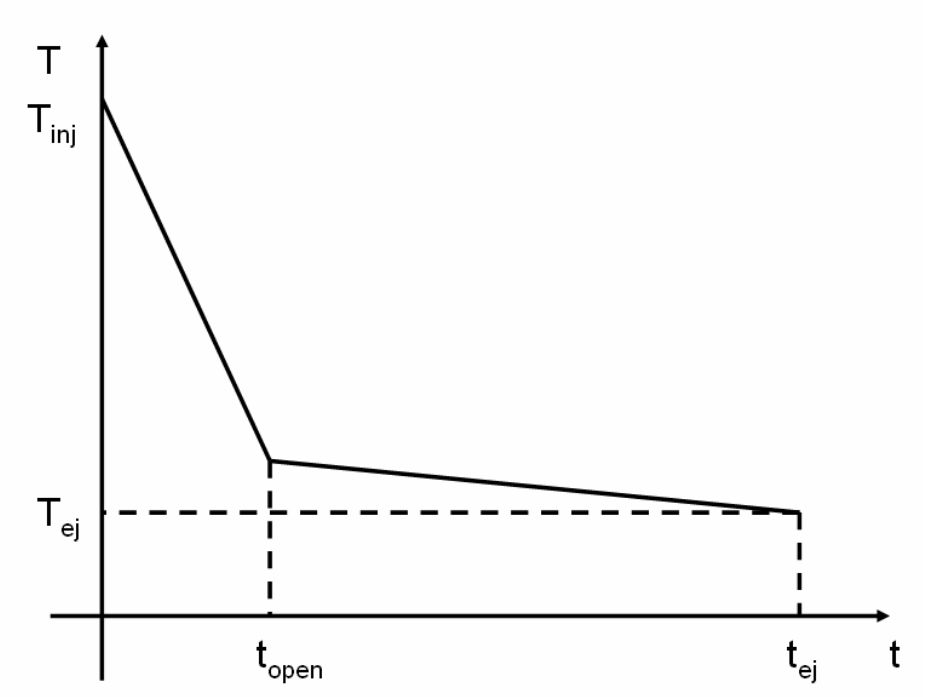
$$\frac{\overline{T}-\overline{T_1}}{R_1}t_{closed}+\frac{\overline{T}-\overline{T}_{air}}{R_{air}}t_{open}+\frac{\overline{T}-\overline{T_2}}{R_2}t_{cycle}+\frac{\overline{T}-\overline{T_3}}{R_3}t_{cycle}+\frac{\overline{T}-\overline{T_4}}{R_4}t_{cycle}$$
$$+\frac{\overline{T}-\overline{T_5}}{R_5}t_{cycle}+\frac{\overline{T}-\overline{T_6}}{R_6}t_{cycle}=qt_{cycle} \quad (2\text{-}55)$$

$$\overline{T}\left(\frac{t_{closed}}{R_1}+\frac{t_{air}}{R_{air}}+\frac{t_{cycle}}{R_2}+\frac{t_{cycle}}{R_3}+\frac{t_{cycle}}{R_4}+\frac{t_{cycle}}{R_5}+\frac{t_{cycle}}{R_6}\right)-\frac{t_{closed}}{R_1}\overline{T_1}-\frac{t_{cycle}}{R_2}\overline{T_2}$$
$$-\frac{t_{cycle}}{R_3}\overline{T_3}-\frac{t_{cycle}}{R_4}\overline{T_4}-\frac{t_{cycle}}{R_5}\overline{T_5}-\frac{t_{cycle}}{R_6}\overline{T_6}=qt_{cycle}+\frac{t_{open}}{R_{air}}\overline{T}_{air} \quad (2\text{-}56)$$

Thus the equation can be written in the form of Eq. (2-10). If there are more faces on interface (contacting air or part), the heat balance equation needs to change accordingly.

2.8 Average Temperature at Different Stages

In Eq. (2-56), temperature $\overline{T}$ is the average temperature of an interface voxel over the whole cycle. This temperature is used to calculate the heat transfer with other interface voxels in different cycle stages. However, careful inspection found there was error by using this global time average temperature.



Section I-Figure 2.14: Calculating average temperature of an interface part voxel at different cycle stage. There are two stages, contacting die surface and contacting air.

Suppose there is a part voxel at interface (Fig. 2.14), during time period 0 ~ t_{open}, the heat transfer happens between part voxel and die voxel. During time period t_{open} ~ t_{ej}, the heat transfer happens between part voxel and air voxel. In Eq. (2-56), the global equilibrium temperature $\overline{T}$ is used to calculate the heat transfer in both periods. This is not correct because neither the time average temperature $\overline{T}_{part\_closed}$ during the period 0 ~ t_{open} is the global temperature $\overline{T}$ nor the time average temperature $\overline{T}_{part\_open}$ during the period t_{open} ~ t_{ej} is $\overline{T}$. In fact, it can be seen from Fig. 2.14 that the three average temperatures are different.

An observation from Fig. 2.14 is that $\overline{T}_{part\_open}$ is very close to T_{ej}. Based on this, an approximation can be done to reduce the error. Let us assume that $\overline{T}_{part\_open} = T_{ej}$, then:

$$\overline{T} = \frac{t_{open} * \overline{T}_{part\_closed} + (t_{ej} - t_{open}) * \overline{T}_{part\_open}}{t_{ej}} \tag{2-57}$$

$$\overline{T} = \frac{t_{open} * \overline{T}_{part\_closed} + (t_{ej} - t_{open}) * \overline{T}_{ej}}{t_{ej}} \tag{2-58}$$

$$\overline{T}_{part\_closed} = \frac{\overline{T} * t_{ej} - (t_{ej} - t_{open}) * \overline{T}_{ej}}{t_{open}} \tag{2-59}$$

Since $\overline{T}$ is known during computation, T_{ej} can be calculated by asymptotic-surrogate model and $\overline{T}_{part\_closed}$ can be calculated. By this improvement, the error mentioned earlier can be corrected partially.

2.9 Cooling Line and Spray

Cooling lines and spray are two important ways for changing heat removal conditions and improving the design. Two issues are considered: 1) How to define the geometry; and 2) How to model their effects for heat transfer.

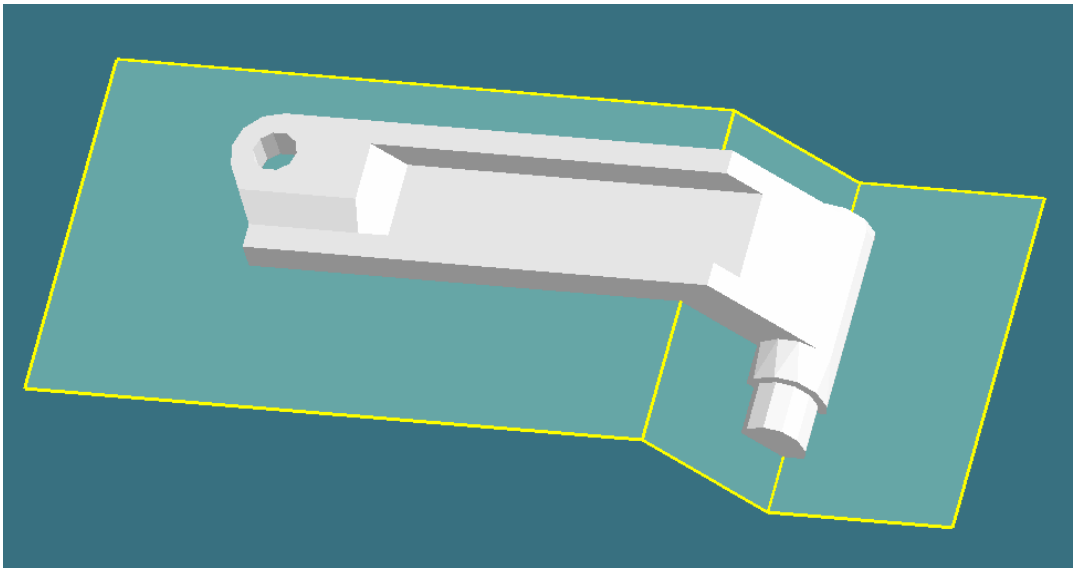
In this method, the opening direction and parting plane are defined before cooling line definition since a cooling line cannot go through parting plane or run into the part. One or more sketching planes are then defined on which cooling lines can be located. The

diameter of the lines and the thermal data of the media can also be specified for the cooling lines (Chen, 2002).

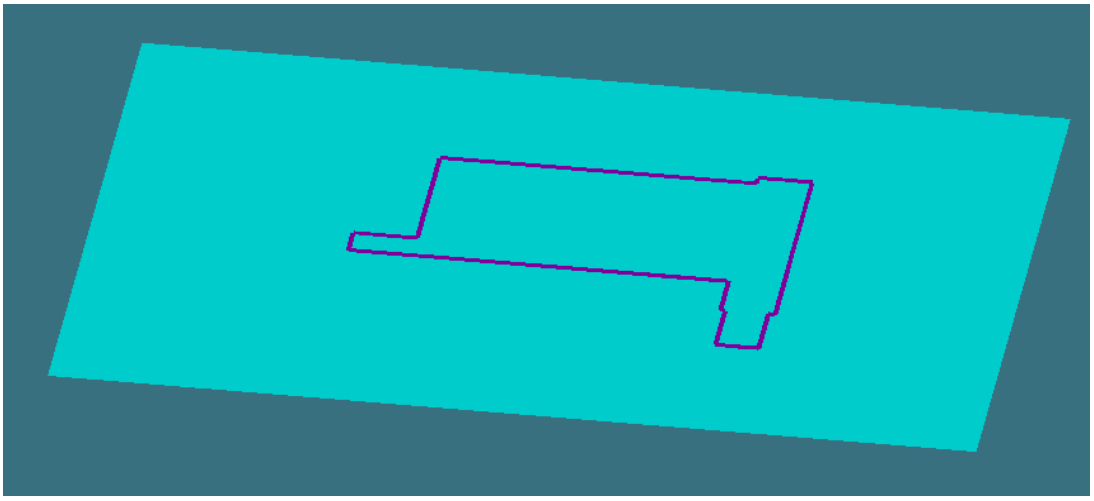
The definition for spray is relatively simple since spray is applied only on the parting surface and die cavity. Rectangular areas on the parting surface can be defined to represent spray areas. The procedure to define a spray area is:

1) Define the die configuration including parting plane (may be a stepped parting plane);
2) Construct a sketch plane perpendicular to the opening direction and at the middle of insert box;
3) Calculate the intersection contour of the part with parting plane;
4) Project the contour on the sketch plane for drawing reference;
5) Define rectangle spray areas on the sketch plane.

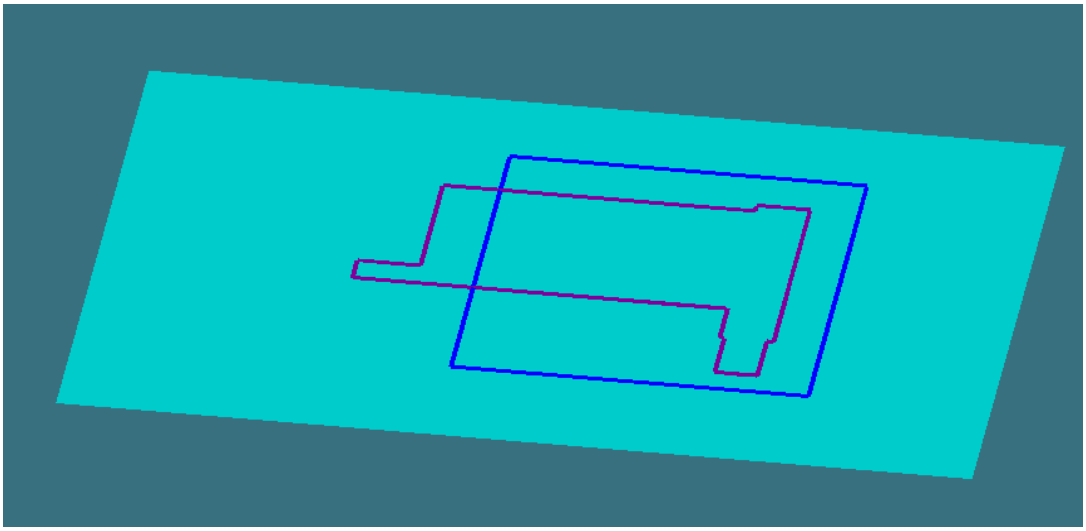
The drawing procedure can be illustrated by Fig. 2.15 ~ 2.17.



Section I-Figure 2.15: A part with stepped parting plane



Section I-Figure 2.16: Define a sketch plane and the intersection contour of part with parting.



Section I-Figure 2.17: Add a spray area at sketch plane by define a rectangle region (in blue color).

Like other components (die or part), cooling lines are represented by voxels. Thus the heat transfer between cooling line and other components (die or insert) is modeled using the same finite difference equation with the assumption that the media temperature in the cooling line is constant.

Modeling actual spray effects is complicated but it is modeled approximately by simply increasing the heat transfer coefficient between the sprayed region and air. In the calculation, if the die surface is in spray area, its coefficient is increased to a certain value that is user selectable.

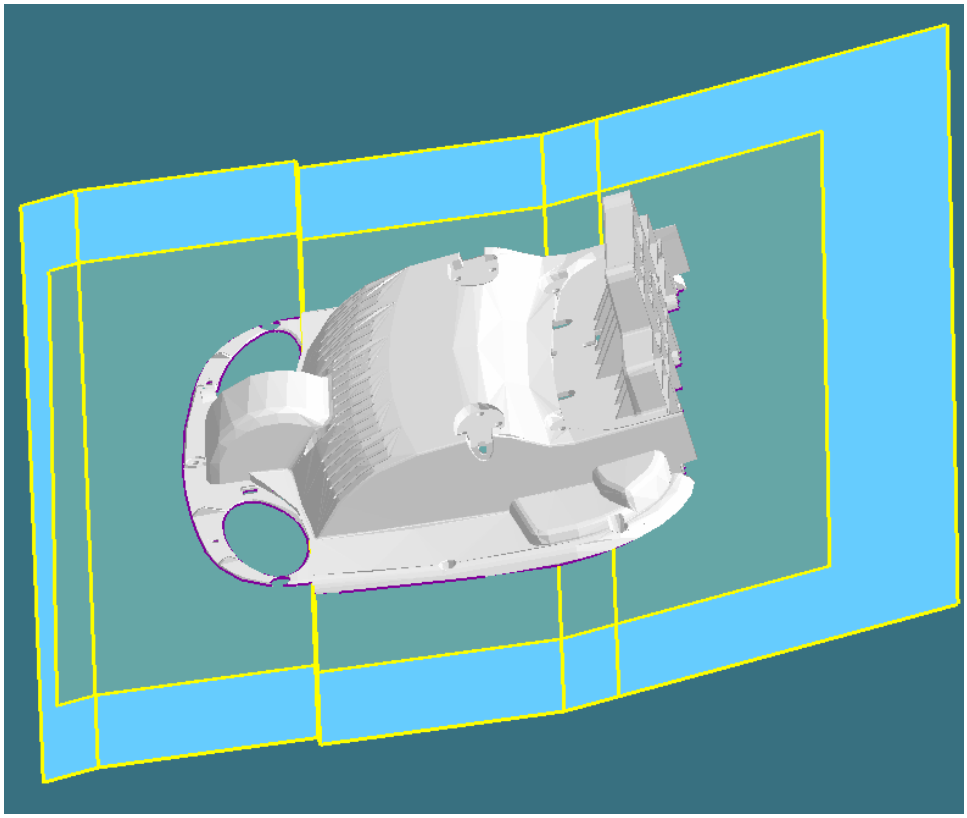
2.10 Handling Parting Surface

A parting surface is the contact surface between ejector die and cover die. According to the surface complexity, the parting surface can be classified into two types, a simple parting plane and stepped parting surface. A simple parting plane is a single plane that

splits the cover half and ejector half and the whole surface is at the same plane. A stepped parting plane actually contains multiple planes connecting each other to form a complex surface. Fig. 2.18 shows an example of stepped parting plane.

The parting plane plays an important role in equilibrium temperature calculation since it determines the heat transfer condition at interface, i.e. heat transfer between steel-steel and steel-air. Detailed treatment for the calculation was discussed earlier. Therefore, an important work is to split the die shoe into two pieces, cover die and ejector die in the voxel model. In the current algorithm, only the part model is created by discretizing the CAD model into voxel model. The die voxel model is then constructed based on the part voxel model. Therefore, a question to be answered is how to split die into two halves if the user has defined a parting plane as Fig. 2.18.

Cai (2002) developed an algorithm to easily define stepped parting planes and internal parting plane. The output of the complex parting plane contains some loops, which are actually intersection vertexes of planes with the die box or part, with an indicator for external plane or internal plane. His technique can be used to split the die into two pieces.



Section I-Figure 2.18: An example of complex stepped parting plane. This stepped parting plane contains 6 separate planes.

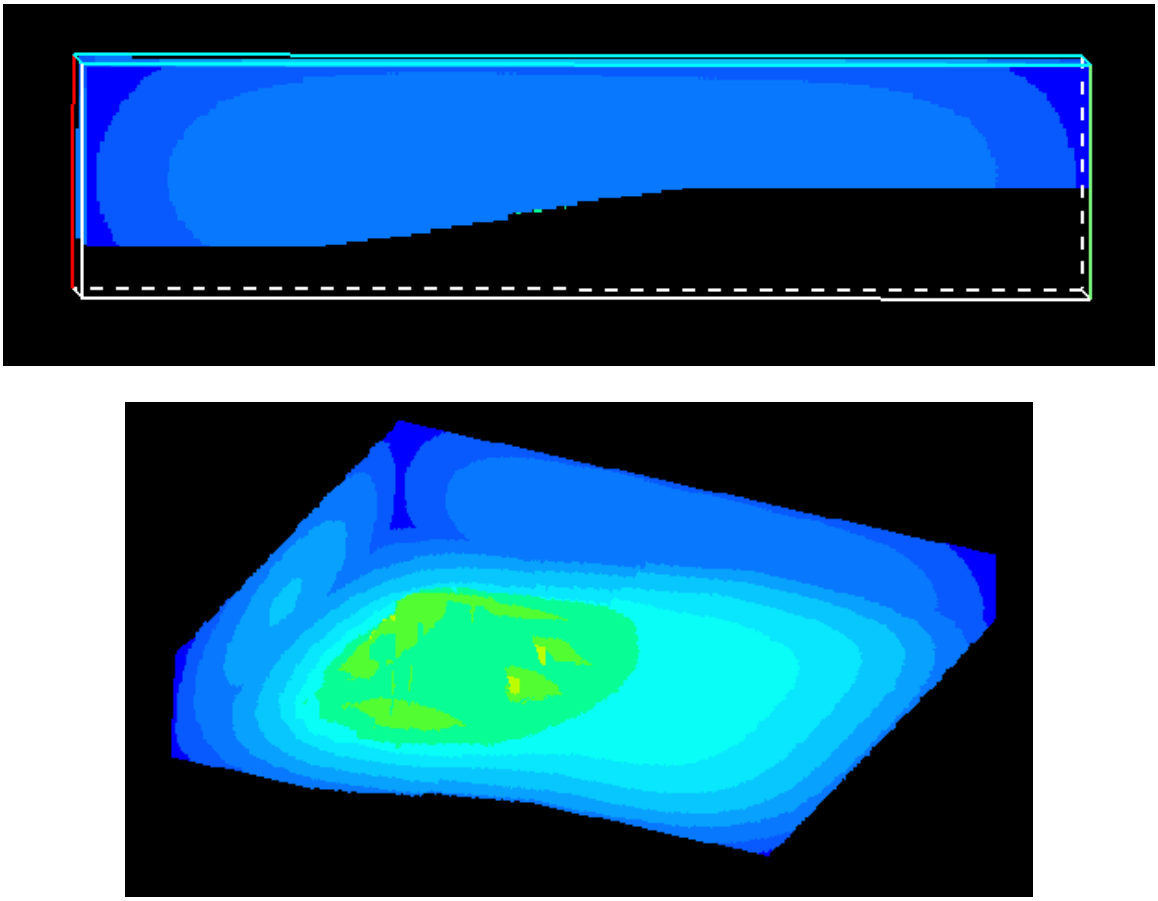
If the parting surface is a simple parting plane, the splitting procedure is:

1) Rotate the part model and parting plane so that Z direction is the opening direction;
2) For each voxel at X-Y bottom plane, march along Z direction to the top;
3) If no part voxel is encountered, die is split at simple parting plane;
4) If there is part voxel encountered, die is split the first time a part voxel is encountered.

For a stepped parting plane, it becomes a bit more complicated since the first task is to search which loop that each X-Y bottom voxel corresponds to. The procedure to split the die based on the loops information is then:

1) Rotate part so that Z direction is opening direction;
2) Project the loops to the bottom plane of die box;
3) Triangulate projected loops (polygons) which may be convex or concave polygons;
4) For each voxel at the bottom plane, calculate which triangle that the voxel center point is in. Thus the loop that the voxel belongs to is known since each triangle is associated with a loop.
5) For this voxel, calculate the height of the parting plane (since its loop is known from 3);
6) Split die box for this voxel using simple parting plane technique;
7) This sequence is used on external planes then on internal planes. External parting surfaces represent parting surfaces where the two die halves meet outside the part. Internal parting surfaces represent the parting surfaces that are inside the part.

By this method, the two pieces of die (cover and ejector) can be separated and its interface can be modeled. The cover die after splitting for example in Fig. 2.18 is shown in Fig. 2.19.



Section I-Figure 2.19: Voxel model of cover die after spitting.

2.11 Computational Efficiency

2.11.1 Two-step method

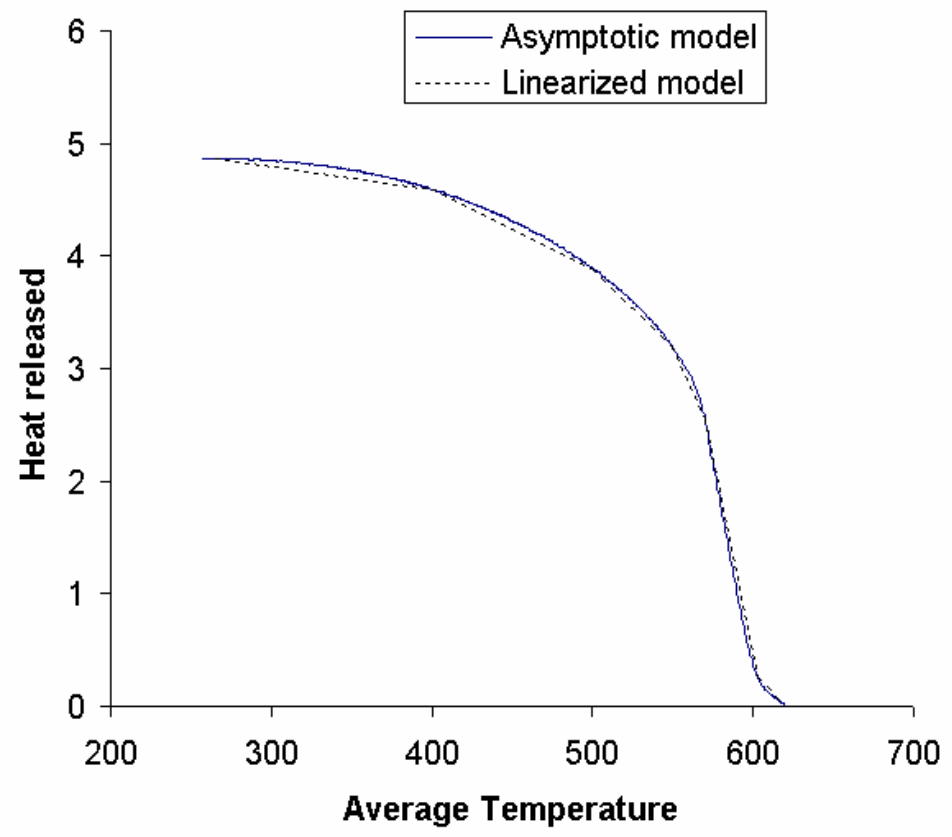
The SOR solver requires an initial value for iteration. The closer this initial value is to the final result, the faster the computation converges. In addition, designers are usually concerned with part and insert temperature more than die shoe temperature. Thus in a two-step method, more effort is put on initial temperature of insert and part to improve the computation efficiency. In the first step, a coarse mesh and loose convergence

criterion are applied to the whole domain (die, insert and part). The result of first step is used as the initial value for the second step computation (by interpolation) where the mesh is refined and a tight convergence criterion is applied. In the second step, the computation is limited to the insert and part. Since the initial value is close to the final result, the computation converges quickly. Due to the application of a refined mesh and high convergence criterion, the accuracy for insert and part temperature can be ensured. Experiments showed that similar results could be obtained while the run time was reduce up to 40%.

2.11.2 Heat source linearization for asymptotic model

Due to the relationship between temperature and released heat for part voxels, the source term of a part voxel changes during the calculation. This change, which is non-linear (Fig. 2.20), causes two problems and increases run time. First, the source term needs to be computed for each part voxel at each iteration; and second, the changing source term slows convergence. To alleviate these problems, a linearized model is applied, in which a few line segments are used to approximate the heat released function. In Fig. 2.20, each segment represents the linearized relationship between temperature and heat. The source term then can be written as: $S = a*T + b$. The first term $a*T$ can be moved to left hand side in Eq. (2-10). The term b is left on the right hand side of Eq. (2-10) but it does not change in this interval. There is no need to re-compute the source term if the temperature stays in the same interval. The calculation also converges faster due to the more stable source term. Experiments showed that the run time could be reduced about 20% compared to the nonlinear representation for pure asymptotic model. For the combination

model (asymptotic and surrogate), the run time is reduced about 5% since asymptotic model only takes effect in the early 20% computation.



Section I-Figure 2.20: Asymptotic model and its linearized model for relationship between temperature of part voxel and heat it releases.

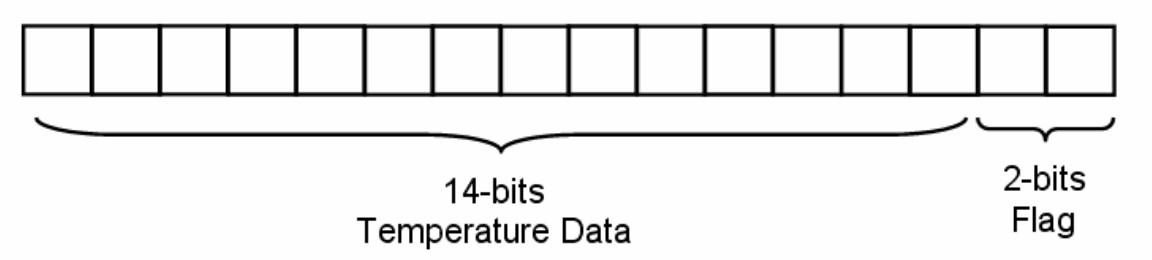
2.11.3 Symmetric part

Some parts are symmetric then only half or even quarter of them needs to be computed. Like transient solvers, for these cases, an isothermal layer is set at its central line and only one half is computed. The other half can be mirrored after the computation is finished. The run time is reduced proportionally.

2.12 Data Storage

The result output is volume data, which actually contains three sets of data corresponding to temperatures of three components (part, cover insert and ejector insert). The volume data can be rendered by CastView post-processor where ray tracing is applied. The three sets of data can be output in three buffers to the post-processor for display. However, this will cause memory waste because some voxels in each buffer are not used.

A better way is to store the whole data in a single buffer but this requires some modification of the current post-processor. The original ray tracer in CastView renders the whole volume data at once. Since the output data from equilibrium temperature analysis is an array of integer data, containing temperature of three components (cover, ejector and part), if only the result of a component is desired, a filter between temperature volume data and ray tracer is necessary.



Section I-Figure 2.21: Store three data sets in a single buffer by using lower 2 bits as flag to distinguish die, cover insert and ejector insert. The 14 higher bits are used to store temperature data.

Since the temperature range for die casting can not be outside of 0 ~ 1000 and an integer type (16-bits) can stores a value between 0 ~ 65536, the lower 2 bits can be used as flags. Fourteen bits can still store a value between 0 ~ 16384, sufficient for temperature

representation. The setting for the lower two bits is: 00 -- part voxel; 01 -- cover insert voxel and 10 -- ejector insert voxel. Thus, before the rendering, the filter fetches the appropriate data from volume data and sends it to the ray tracer.

2.13 Conclusions

1) An algorithm based on the Finite Difference Method for equilibrium temperature calculation for die casting process has been developed to speed up part and die design at early design stage and help optimize cooling, heating and cycle design.

2) The algorithm is built on the assumption that the heat released from part equals to the heat absorbed by the environment during a cycle after the production reaches the steady state. The solver SOR was chosen to solve the linear equation system.

3) A few models have been tried to compute the heat released from part voxels. The combination of asymptotic model and surrogate model are chosen because they can compute the ejection from average temperature.

4) Heat transfer at the interface, cooling line and spray effect are addressed in the algorithm.

5) Special attention is paid to computational efficiency and data storage to save run time and memory

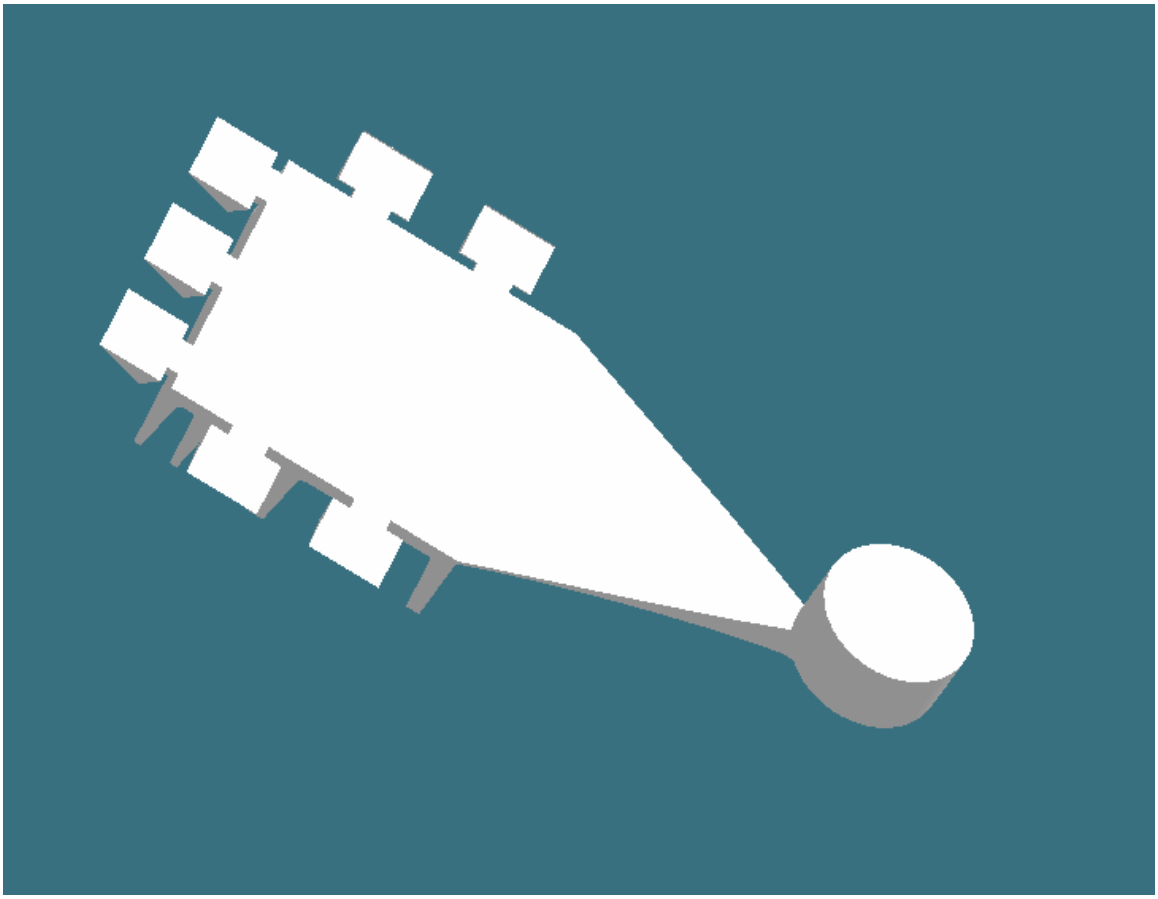
3. EXAMPLES AND VERIFICATION OF EQUILIBRIUM TEMPERATURE ANALYSIS FOR DIE AND PART

3.1 Implementation

The program for equilibrium temperature analysis is being integrated into the *CastView* software using Visual C++ on the MS Windows platform. The examples in this chapter were run on a 2.4 GHz Pentium IV PC with 1 GB memory. The typical run time was less than 1 minute with resolution of 200 voxel at maximum dimension along X, Y and Z directions.

3.2 Case 1: Flat Part

The first part to be discussed is a flat part (shown in Fig. 3.1). For comparison, the transient temperature simulation of 100 cycles was run on Abaqus using the same parameters. It is assumed that at cycle 100, the production has reached the steady state and the time average temperature of die is the equilibrium temperature.



Section I-Figure 3.1:Geometry of flat part, rendering in CastView.

The two sets of analysis parameters on Abaqus and CastView are the same and is list below, which comes from CastView data file. The meanings of most parameters are clear by their names. The parameters in Advanced section are for SOR solver and convergence tolerance.

General:

cycle_time: 51 sec
die_close_to_end_of_fill: 1 sec
end_of_fill_to_die_open: 9 sec
time_to_eject: 7 sec

spray_time: 0 sec

room_temp: 30 C

die_con_file: fp-part.cvdcg

cooling_file(0-no__1-yes): 1 fp-part.cvcln

spray_file(0-no__1-yes): 0

Casting:

material: Al380

density: 2.76 g/cm^3

specific_heat: 0.963 J/g-K

conductivity: 109 W/m-K

latent_heat: 389 J/g

injection_temp: 620 C

liquidus_temp: 598 C

solidus_temp: 538 C

Die:

holder_block_material: H13

insert_material: ST4140

holder_block_conductivity: 40 W/m-K

insert_conductivity: 29 W/m-K

heat_transfer_coefficient_steel_to_steel: 5000 W/m^2-K

heat_transfer_coefficient_steel_to_air: 16 W/m^2-K

heat_transfer_coefficient_part_to_cavity: 5000 W/m^2-K

Others:

symmetry(0-none__1-x__2-y__3-xy): 1

part_temp(0-ejection__1-average): 0

output_die_temp(0-no__1-yes): 0

top_platen(0-no__1-yes): 0 0 C
bottom_platen(0-no__1-yes): 0 0 C
left_platen(0-no__1-yes): 0 0 C
right_platen(0-no__1-yes): 0 0 C
front_platen(0-no__1-yes): 0 0 C
back_platen(0-no__1-yes): 0 0 C

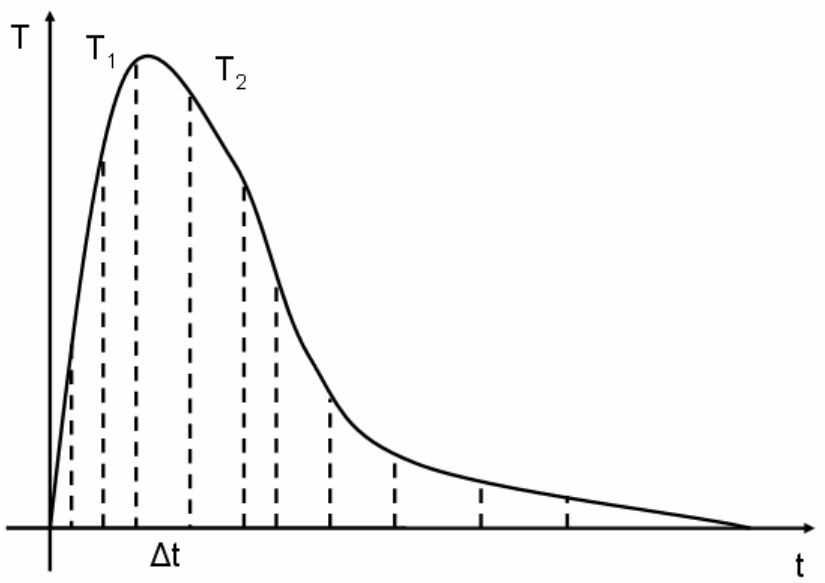
Advanced:

first_step: 0.01
second_step: 0.001
SOR_Omega: 1.9
asymptotic_temp: 150 C

Since Abaqus provides the option to output temperature change into a text file, the time average temperature, i.e. the equilibrium temperature can be obtained by integrating the temperature and dividing by the cycle time. The calculation is shown by Fig. 3.2. The time average temperature at each nodal point is then:

$$\overline{T} = \frac{\sum (T_1 + T_2) * \Delta t}{2t_{cycle}}$$

Therefore, the Abaqus results (equilibrium temperature of the die and part) can be compared to CastView. Further, the comparison can be performed not only by temperature pattern but also the temperature values since the result data from Abaqus is available.

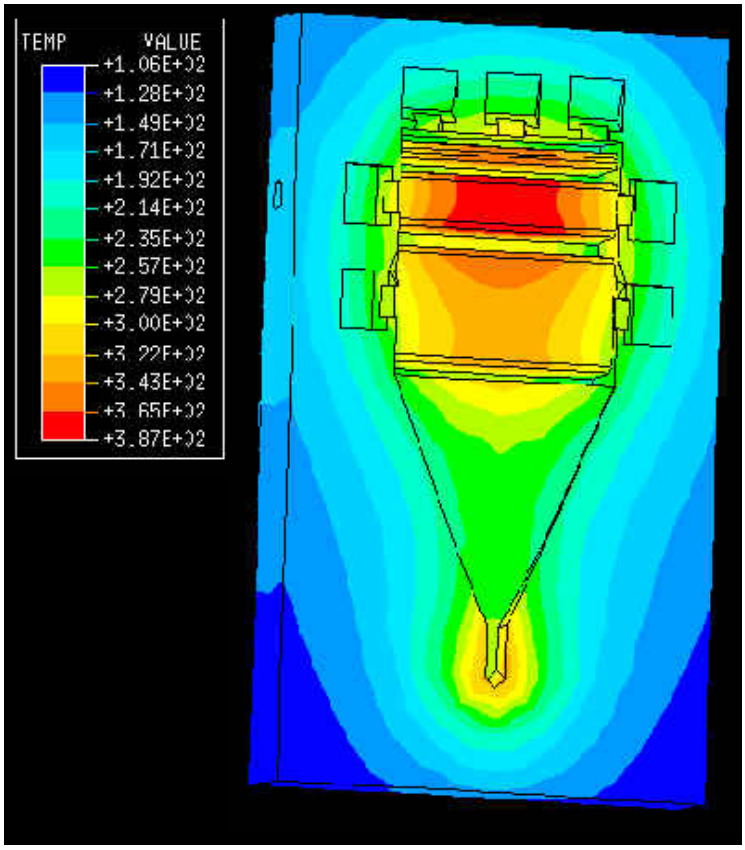


Section I-Figure 3.2: Calculate the time average temperature over a cycle from Abaqus result

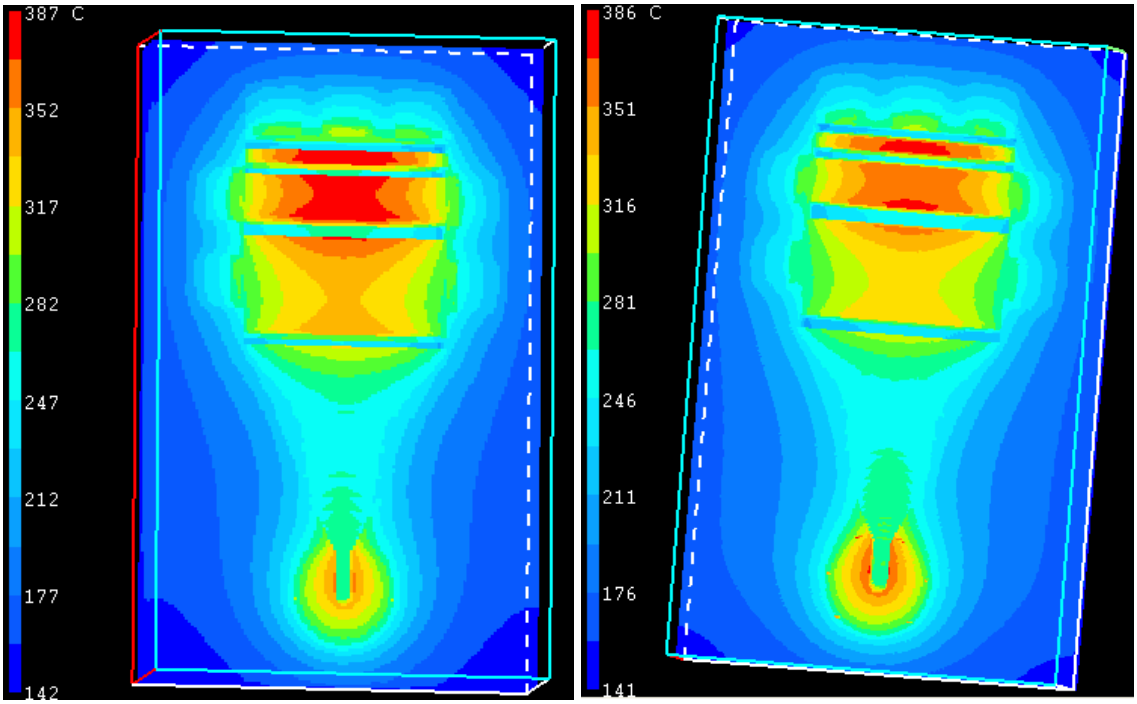
The analysis of flat part was run on CastView with two different voxel resolutions, 200 voxels and 300 voxels. The voxel resolution is referring to the voxel number on the maximum dimension among x, y and z direction of the part bounding box. For example, the dimension of xyz bounding box of the flat part is 152.4mm×275.2mm×56.8mm. The y dimension is larger than the other two. For the 200 voxels case, there are 200 voxels on this dimension. The voxel size is then 275.2/200 = 1.376 mm. The dimension of the whole die box is 300mm×400mm×500mm. The total voxel number in the computational domain is then (300/1.376) ×(400/1.376) ×(500/1.376) = 218×290×363 = 22 948 860. The voxel size for the 300 voxels case is 275.2/300 = 0.917 mm. The total voxel number is then (300/0.917) ×(400/0.917) ×(500/0.917) = 327×436×545 = 77 701 740. The run time for the 200 voxels case is 1 minute 4 seconds and the run time for the 300 voxels

case is 4 minutes 12 seconds. The run time to finish 100 cycles on Abaqus on a similar PC was about 7 days.

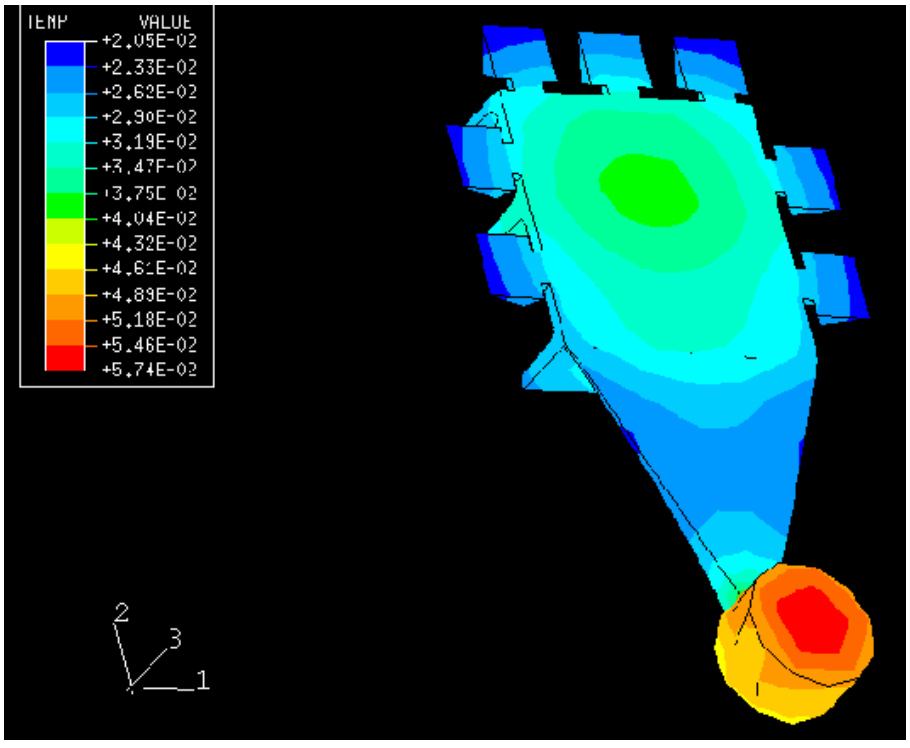
Fig. 3.3 shows the equilibrium temperature of ejector die from Abaqus. As described earlier, this is from the average temperature and is read into Abaqus for display. It can be clearly seen that the hot spot is at the middle rib and the temperature range is 106 ^{o}C ~ 378 ^{o}C. The results of two resolutions from CastView are shown in Fig. 3.4. The temperature range of the 200 voxel resolution is 140 ^{o}C ~ 387 ^{o}C and the range of the 300 voxel resolution is 141 ^{o}C ~ 386 ^{o}C. This may indicate that the 200 voxel resolution is enough for this part since to increase the voxel number does not change the results much. It can also be seen that the three temperature patterns are very similar and the hot spot in three results is at the middle rib.



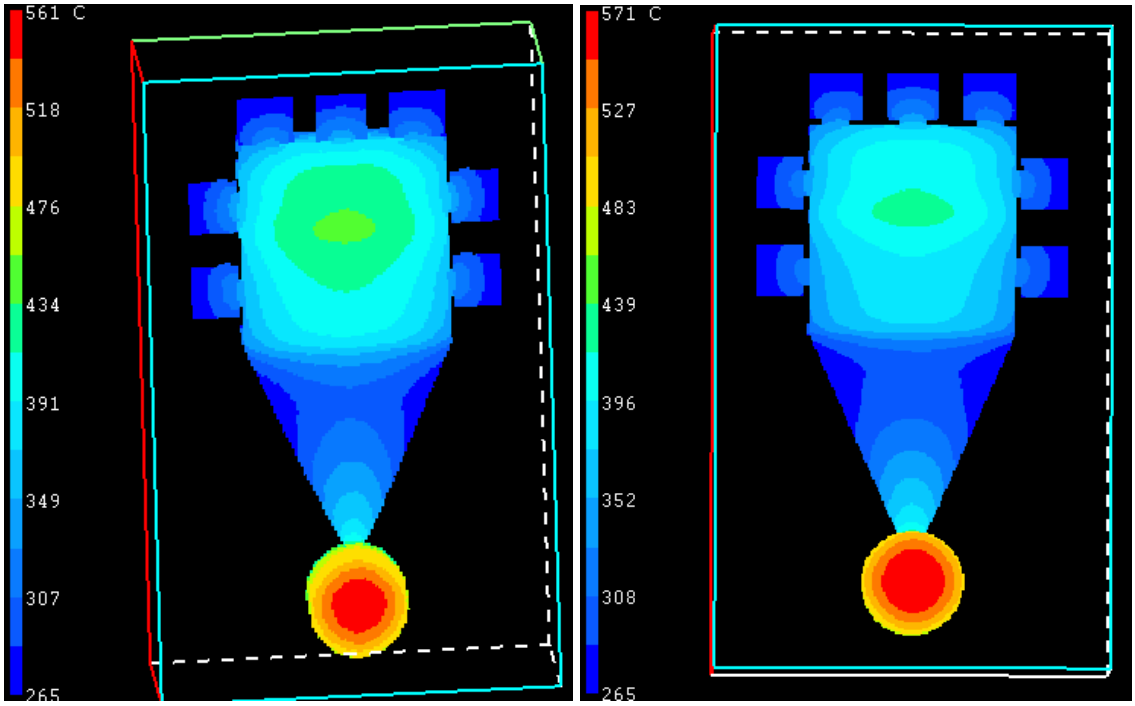
Section I-Figure 3.3: Equilibrium temperature of ejector insert from Abaqus



Section I-Figure 3.4: Equilibrium temperature of ejector insert from CastView. Left: 200 voxel resolution; Right: 300 voxel resolution.



Section I-Figure 3.5: Part ejection temperature from Abaqus



Section I-Figure 3.6: Part ejection temperature from CastView using combined asymptotic and surrogate model (Left: 200 voxel resolution; Right: 300 voxel resolution).

The results for part ejection temperature from both CastView and Abaqus are compared, which are listed in Fig. 3.5 and Fig. 3.6. The calculation from CastView is obtained using combined asymptotic and surrogate model. Unlike die average temperature, the part ejection temperature can be obtained from Abaqus at the cycle 100 without further treatment. It can be seen that the temperature range for Abaqus is 205 ^{o}C ~ 574 ^{o}C. The temperature range of 200 voxel resolution for CastView is 265 ^{o}C ~ 561 ^{o}C and that of 300 voxel resolution is 265 ^{o}C ~ 571 ^{o}C. It can be seen again that the temperature values from two voxel resolutions are not much different. The lower bound from CastView is slightly higher than that from Abaqus with similar pattern in both cases.

A deeper comparison between both cases is the temperature value comparison. Since Abaqus can output temperature into text file, it is possible to compare the temperature value at the same location. However, there is a little work to be done for such comparison since Abaqus applies Finite Element Method and CastView applies Finite Difference Method and one mesh needs to be transformed into another mesh. Here we transform FEM mesh into FDM mesh (Fig. 3.7). The procedure is as follows;

1) FEM mesh and FDM mesh are aligned at low left corner of bounding box, which is the origin;

2) For center of each voxel, calculate the coordinate values. The temperature of this center from FDM is known from CastView results as T;

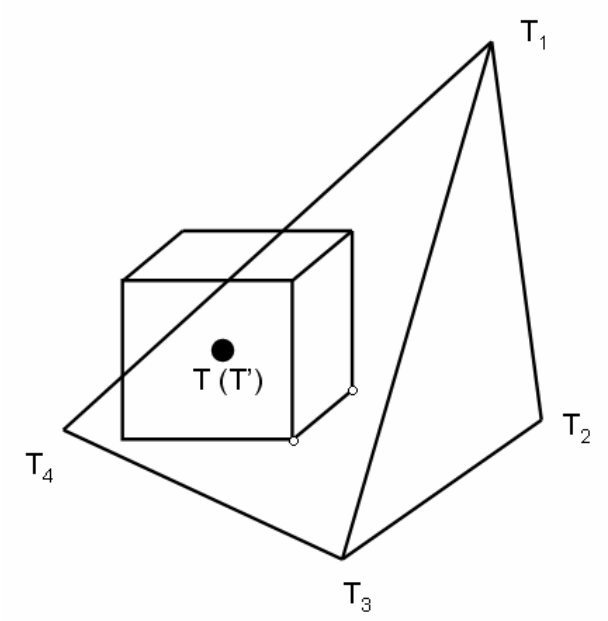
3) Calculate which FEM element this voxel center is in;

4) Calculate the temperature T' at this point from FEM data by interpolation since the temperatures at the 4 vertices are known as T_1, T_2, T_3 and T_4;

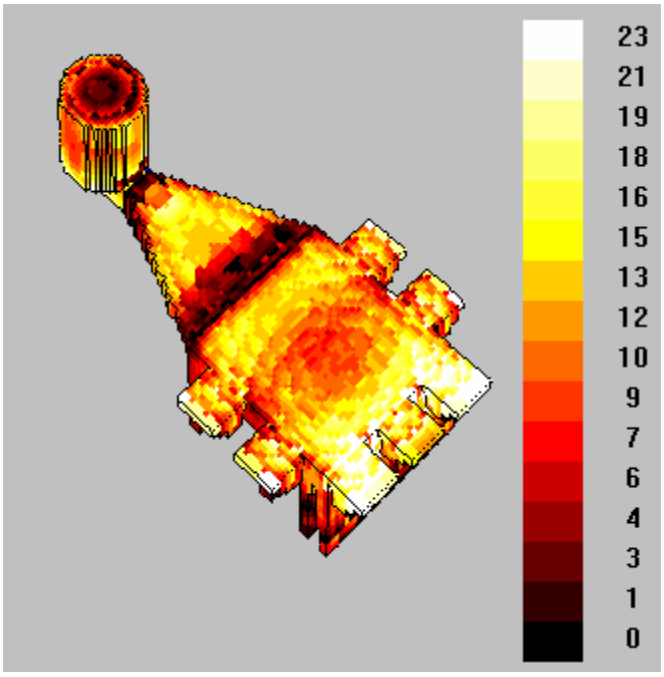
5) Compare the difference between T and T'.

Such comparison on part ejection temperature and die average temperature are shown in Fig. 3.8 and Fig. 3.9 in difference percentage. Since the results of 200 voxel resolution and 300 voxel resolution are not different much, the comparison was done on 200 voxel resolution. The range of difference for part is 0% ~ 23% and for die is 0% ~ 28%. For part, the maximum difference happens at the overflow tip and for die, the maximum

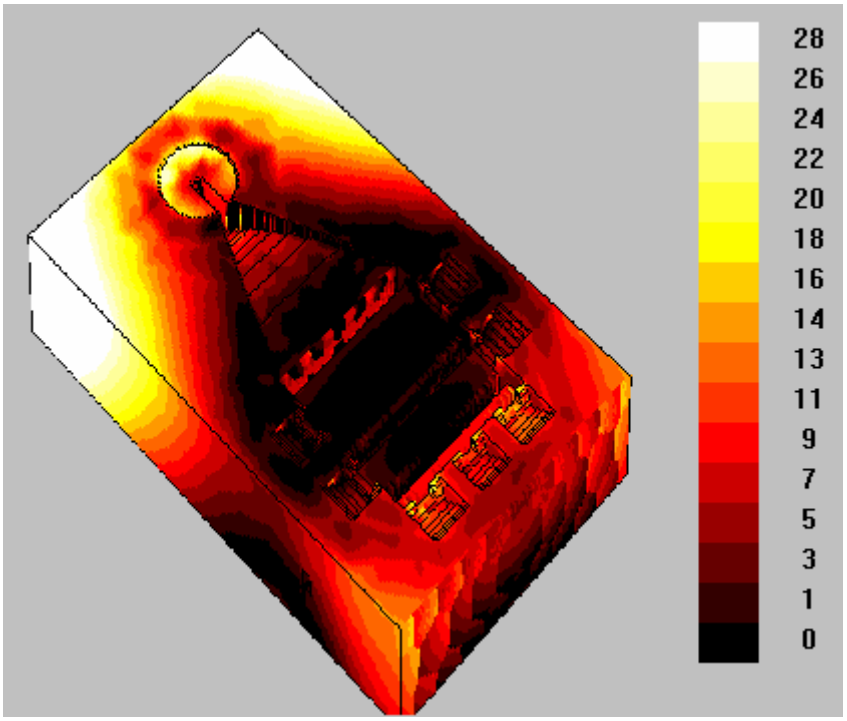
difference happens at the corner of the box. It can be expected that the difference between two cases is partially due to error of interpolation.



Section I-Figure 3.7: Comparison of FDM data and FEM data by interpolation



Section I-Figure 3.8: Part ejection temperature difference in percentage of Abaqus result and CastView result (200 voxel resolution).



Section I-Figure 3.9: Die equilibrium temperature difference in percentage of Abaqus result and CastView result (200 voxel resolution).

3.3 Case 2: Zinc Part

The die configuration is a little special for this part. There are four cavities in this die and no insert and the cavities are in the die directly. Another characteristic is there are six cooling lines in the die and a chiller is used in cooling lines, which helps maintain the cooling line temperature at ~10 ^{o}C. The melting point of zinc alloy is low (~380 ^{o}C) compared to other alloys. Since the dimension is small (only 90 mm on largest dimension for multipart cavity), the cycle time is only 9.35 seconds. The complete calculation parameters are listed below.

General:

cycle_time: 9.35 sec
die_close_to_end_of_fill: 0.23 sec
end_of_fill_to_die_open: 6.08 sec
time_to_eject: 0.2 sec
spray_time: 1 sec
room_temp: 10 C
die_con_file: Casting.cvdcg
cooling_file(0-no__1-yes): 1 Casting.cvcln
spray_file(0-no__1-yes): 1 casting.cvspr

Casting:

material: Zn
density: 6.6 g/cm^3
specific_heat: 0.419 J/g-K
conductivity: 109 W/m-K
latent_heat: 120 J/g
injection_temp: 417 C
liquidus_temp: 389 C
solidus_temp: 380 C

Die:

holder_block_material: H13
insert_material: H13
holder_block_conductivity: 29 W/m-K
insert_conductivity: 29 W/m-K
heat_transfer_coefficient_steel_to_steel: 5000 W/m^2-K
heat_transfer_coefficient_steel_to_air: 20 W/m^2-K
heat_transfer_coefficient_part_to_cavity: 5000 W/m^2-K

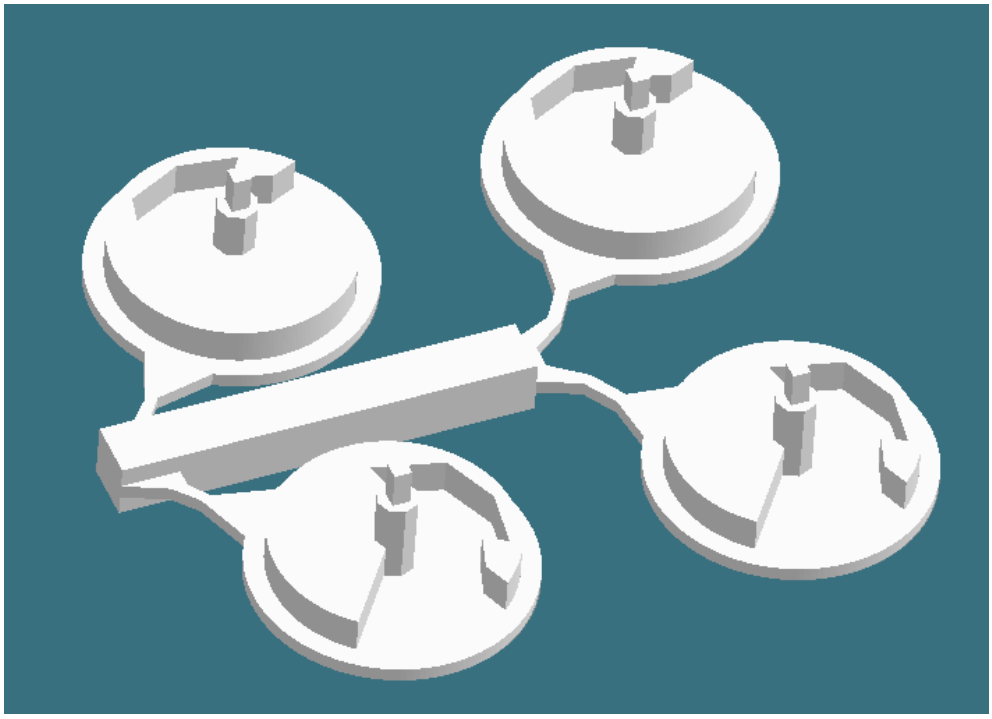
Others:

symmetry(0-none__1-x__2-y__3-xy): 0
part_temp(0-ejection__1-average): 0
output_die_temp(0-no__1-yes): 0
top_platen(0-no__1-yes): 0 0 C
bottom_platen(0-no__1-yes): 0 0 C
left_platen(0-no__1-yes): 0 0 C
right_platen(0-no__1-yes): 0 0 C
front_platen(0-no__1-yes): 0 0 C
back_platen(0-no__1-yes): 1 30 C

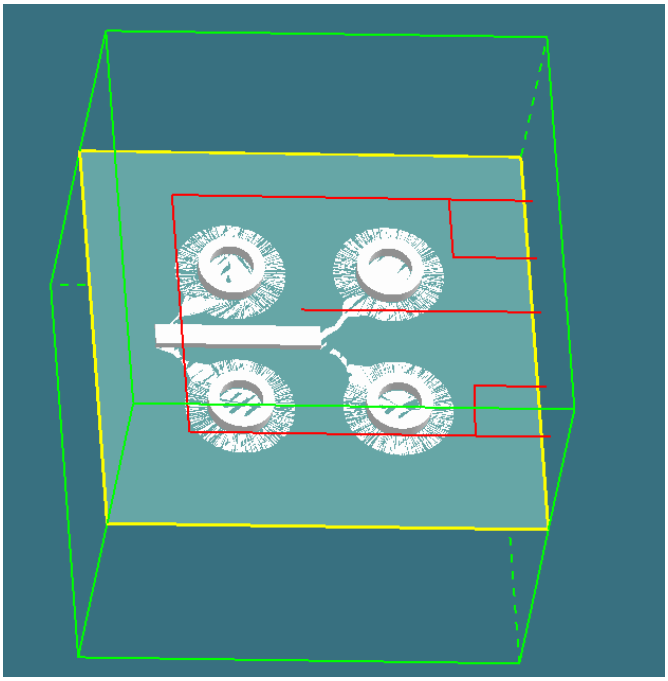
Advanced:

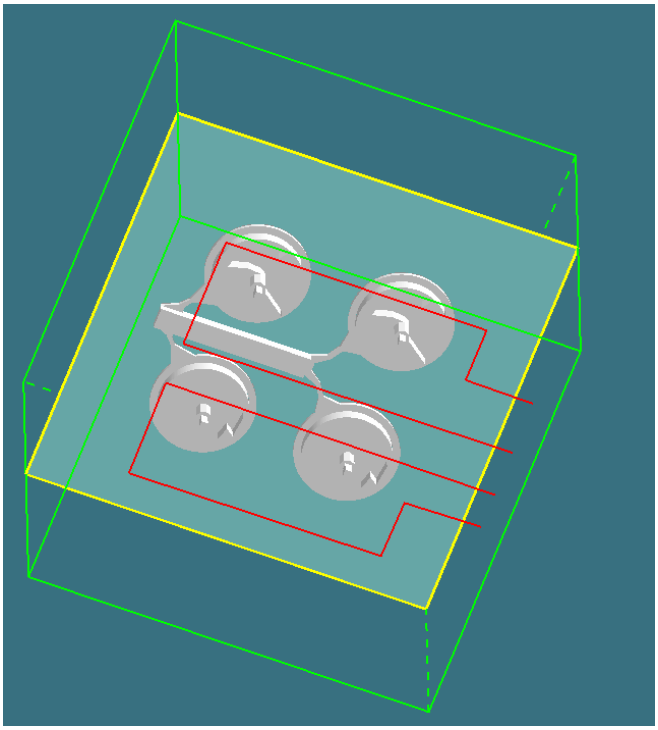
first_step: 0.01
second_step: 0.001

The 3D part geometry is shown in Fig. 3.10 and the cooling lines are shown in Fig. 3.11 (both sides). The spray areas are also shown in Fig. 3.12.

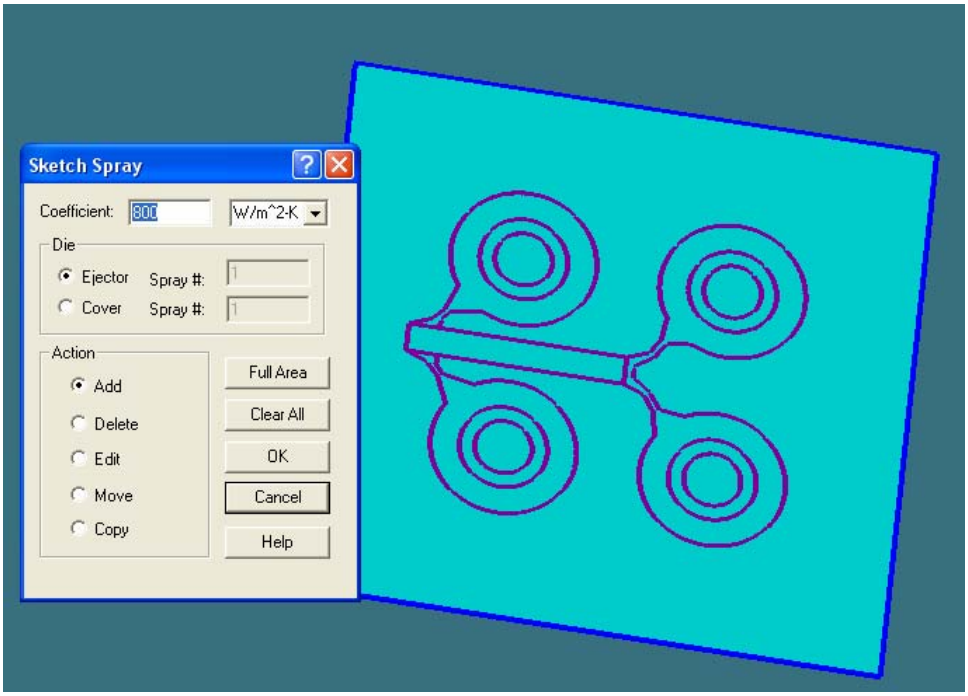


Section I-Figure 3.10: Geometry of the multipart cavity





Section I-Figure 3.11: Cooling line pattern at cover and ejector side (totally 6 cooling lines



Section I-Figure 3.12: Spray pattern (whole area on die surface for both sides)

The Abaqus results and CastView results are compared, where CastView results are computed from pure asymptotic model and combined asymptotic and surrogate model. The die equilibrium temperature from Abaqus is shown in Fig. 3.13 with the range of 17 ^{o}C ~ 247 ^{o}C. Again, the equilibrium temperature from Abaqus is computed by taking average of dynamic temperature over a cycle. The corresponding CastView result is shown in Fig. 3.14 and Fig. 3.15, with range of 34 ^{o}C ~ 260 ^{o}C for pure asymptotic model and range of 33 ^{o}C ~ 235 ^{o}C for mixed model. Both CastView results look like the pattern of Abaqus result. However, result from mixed model is better than that from pure asymptotic model since the temperature value is closer to Abaqus result.

The part ejection temperature from Abaqus is shown in Fig. 3.16, with range of 126 ^{o}C ~ 234 ^{o}C. The corresponding CastView results from both models are also shown in Fig. 3.17 and Fig. 3.18. The pure asymptotic result has the temperature range of 194 ^{o}C ~ 316 ^{o}C and the mixed model has the temperature range of 222 ^{o}C ~ 284 ^{o}C. It can be seen again that these results have similar patterns. However, compared to die temperature, the difference between CastView part result and Abaqus part result is larger. This may be because 1) The thermal diffusivity of die is less than that of part; 2) Die is surrounded by air with constant temperature. These reasons make part temperature more sensitive to errors or other factors. In addition, the geometry of die in CastView and Abaqus is different since the geometry of die for Abaqus was generated by CAD system and the detailed feature can be modeled. The geometry of die for CastView can only be generated

by CastView itself, which is actually a simple die box. The difference of die geometry should also have some impact for part temperature calculation.

3.4 Conclusions

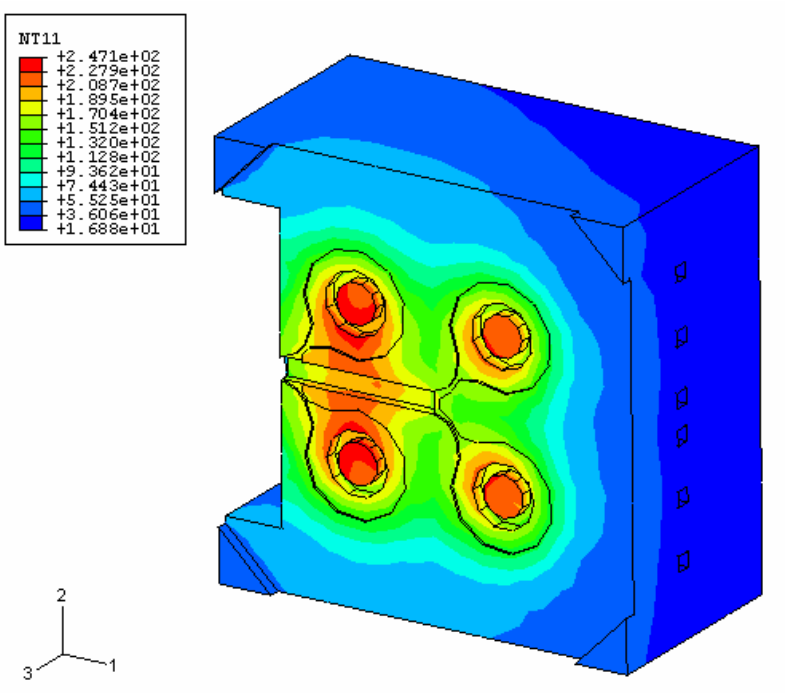
1) Two cases were selected for test and verification of the algorithm for equilibrium temperature analysis. The analyses were run on CastView and on numerical simulation software Abaqus using the same parameters.

2) In the flat part case, the simulation of 100 cycles was run on Abaqus to reach quasi steady state and the run time was a few days. The run time on CastView for equilibrium temperature was only a few minutes.

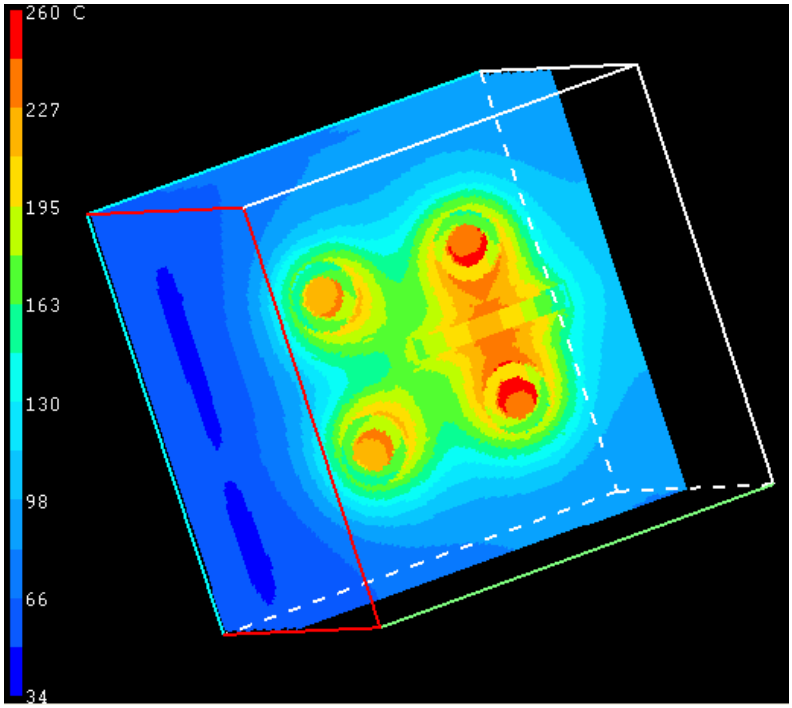
3) The temperature patterns of ejector insert and part from two packages are compared. For flat part case, the temperature values of two results are compared by interpolation since the numerical methods of two packages are different (one is FDM and the other is FEM).

4) In both cases, the temperature patterns of die and part from both packages are very similar and the hot spot is at the same location, which indicates our algorithm for equilibrium temperature is valid and efficient.

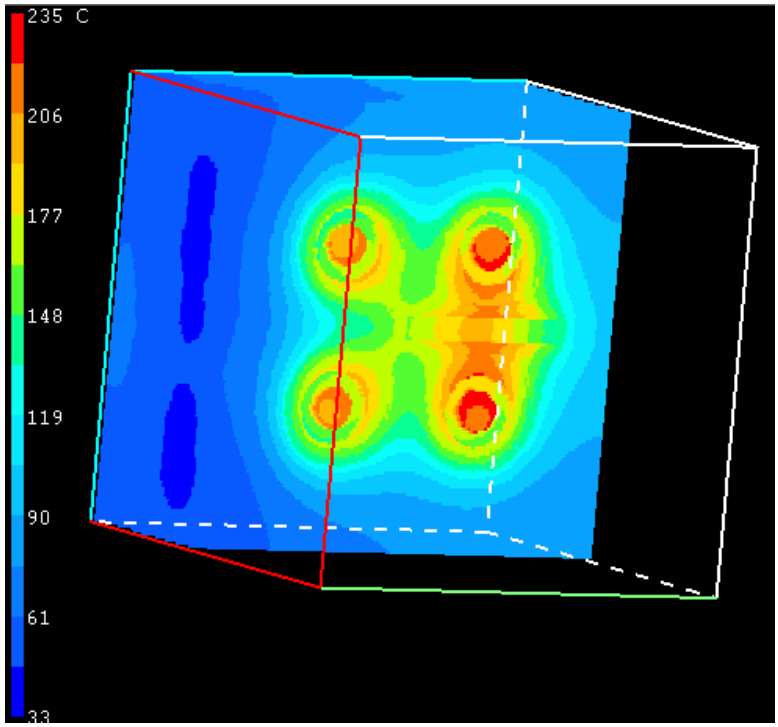
5) In the comparison of zinc part, the part results from CastView and Abaqus are more different that the flat part. This may be caused by a few reasons including the different die geometry in two computations.



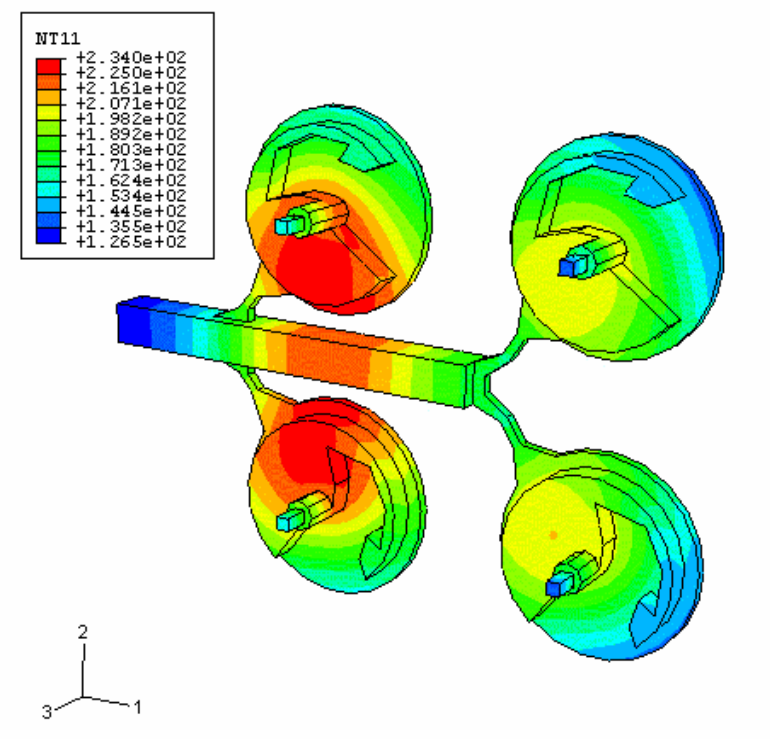
Section I-Figure 3.13: Die equilibrium temperature from Abaqus



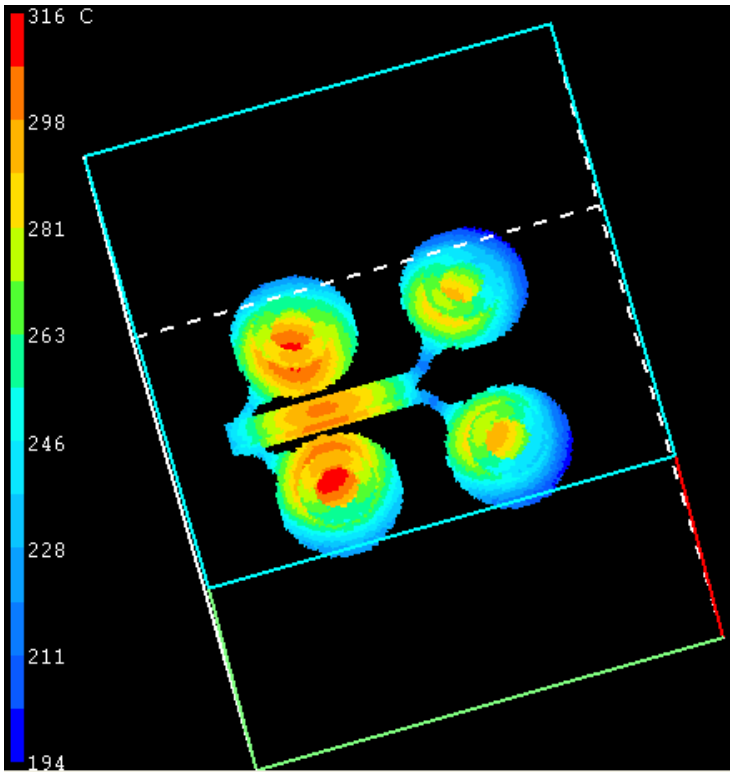
Section I-Figure 3.14: Die equilibrium temperature from CastView (Pure asymptotic model)



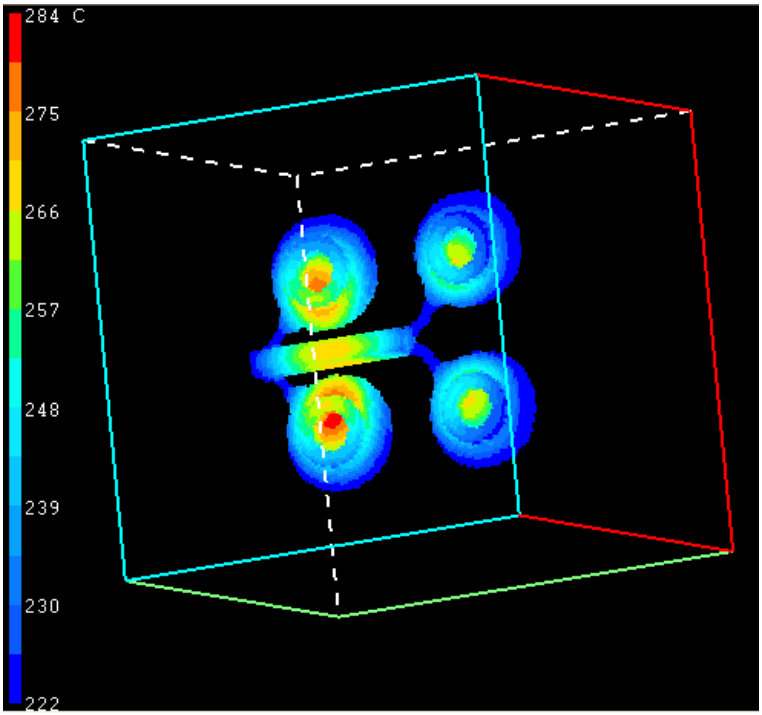
Section I-Figure 3.15: Die equilibrium temperature from CastView (Asymptotic + surrogate)



Section I-Figure 3.16: Part ejection temperature from Abaqus



Section I-Figure 3.17: Part ejection temperature from CastView (Pure asymptotic model)



Section I-Figure 3.18: Part ejection temperature from CastView (Asymptotic + surrogate)

4. IMPROVEMENT OF GEOMETRIC REASONING ALGORITHM FOR FILL PATTERN

4.1 Introduction

The original algorithm applied in CastView for fill pattern makes many assumptions and simplifications. The assumptions and simplifications are helpful to reduce calculation thus reduce computation time. However, they also cause some shortcomings. The major problem is sometimes the fill pattern predicted by CastView is not correct, compared to numerical simulation result by commercial software, especially for processes with complex cavity or multiple gates. The incorrectness may come from the assumptions and simplifications. Therefore, it is important to improve or replace the current algorithm for fill pattern reasoning. Today's computers allow using more complex calculation without much time penalty (however, to solve the conservation equations is still too time-consuming thus is not the direction of this research). This chapter describes an improved algorithm which overcomes some problems of the old algorithm but does not increase the computation time significantly

4.2 Shortcomings and Problems of Old Model

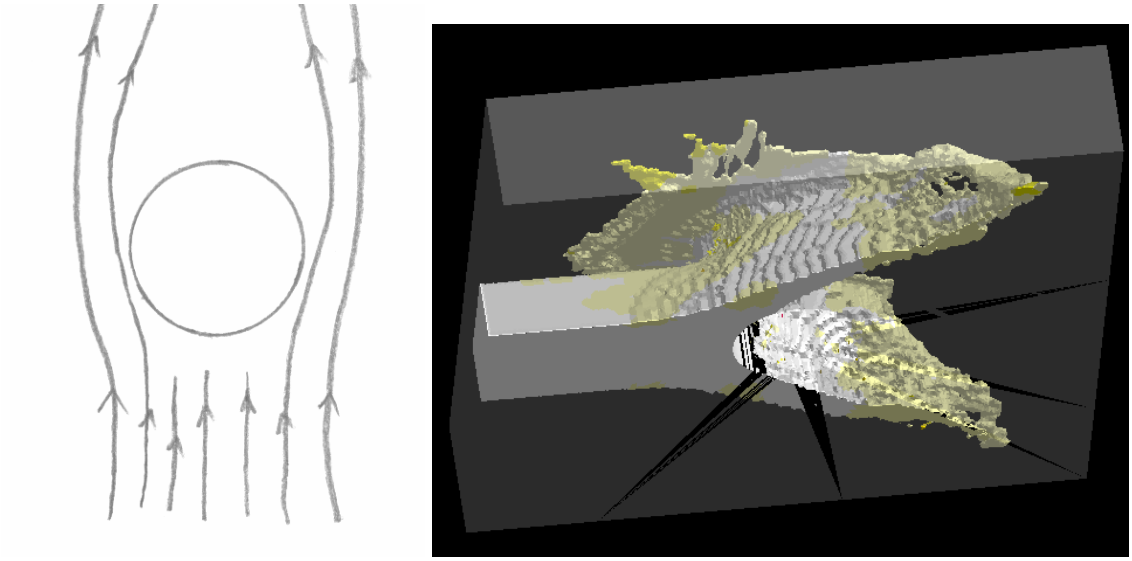
1) Metal speed not included

Velocity includes speed and direction. Generally speaking, the speed is determined by pressure gradient and other factors then speed determines the fill pattern. However, in the old model, speed is not considered and all moving voxels (flow front) therefore effectively have the same speed. As a result, some space may be filled (occupied) faster

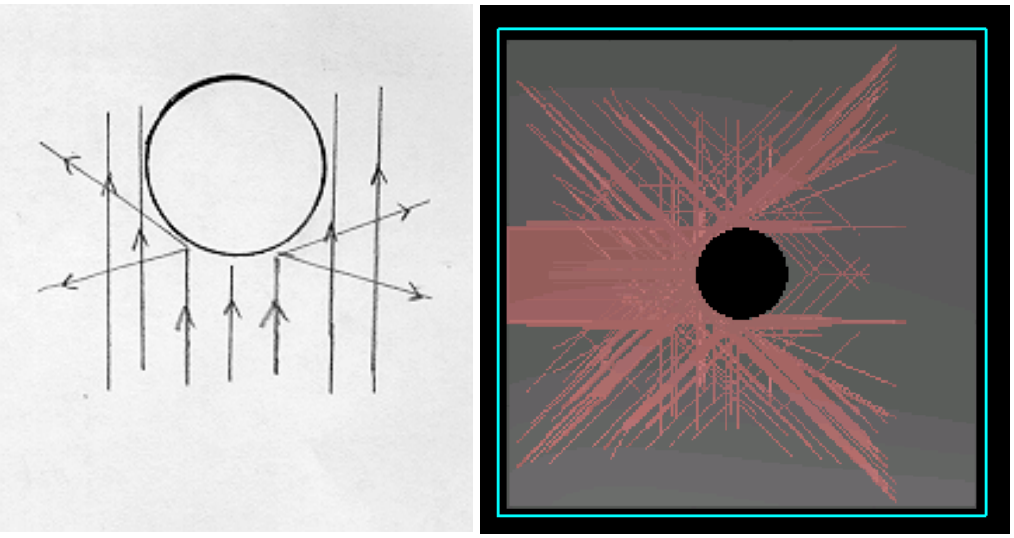
than it should be. These filled voxels then obstruct the flow channel for other moving voxels, which causes error for the whole fill pattern. This is a serious problem of the current model and needs to be corrected in new algorithm.

2) Flow obstruction calculation

In the old model, when there is an obstruction, the new flow front is calculated based on the available empty voxels. As a result, a flow will be reflected from a central obstruction, which is not correct (shown in Fig. 4.1 and Fig. 4.2). Fig. 4.1 shows the correct flow pattern when there is an obstruction. The left one is a sketch as demonstration and the right one is a numerical simulation for a simple part using MagmaSoft. Fig. 4.2 shows the result from old model. In the new method, this calculation is improved. The new flow vector is calculated not only based on the available empty voxels, but also on the original flow vector and the near flow vectors.



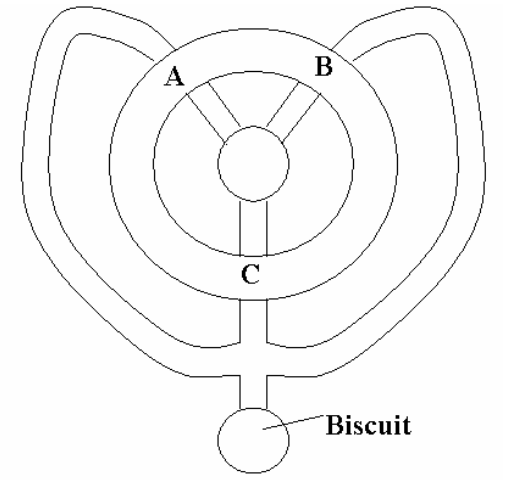
Section I-Figure 4.1: Actual flow pattern when there is an obstruction. Left: sketch; Right: numerical simulation from MagmaSoft.



Section I-Figure 4.2: Result calculated using old algorithm when there is an obstruction. Left: sketch; Right: CastView result from old algorithm.

3) Cavity with multiple gates

For some parts with thin wall or have special shape, multiple gates are usually applied to avoid filling problem. A hypothetical example is shown in Fig 4.3. Generally, an ideal engineering design for multiple gates and the resistance at the gates themselves should ensure the melt enters the cavity simultaneously from each. In this situation, cavity fill will start after the runner system pressurizes which should not happen until the runner system is nearly full. Even if melt reaches gate C before the other two, the cavity will not start to fill because the gate resistance is significantly higher than the runner resistance. However, the old model can not distinguish if this is a good design. Due to different fill distance of OA, OB and OC, the melt reaches B earlier than it reaches A and C. The melt will then fill the cavity at B before it reaches A and C even the thickness at C is thin enough, which will makes an incorrect prediction for the fill pattern.



Section I-Figure 4.3: Melt should reach gate A, B and C simultaneously for a well designed runner system with multiple gates

4) Flow resistance not considered.

Flow resistance is an important concept in qualitative reasoning for fill pattern since pressure is not calculated. Flow resistance is difference at different location due to different geometric characteristics. The flow resistance affects the flow speed then affects the whole fill pattern.

4.3 Improvements in New Algorithm

The basic idea is to correct the problems mentioned earlier but not to increase computation time or memory requirements significantly since the efficiency is a principal goal. The following improvements have been accomplished:

1) Speed calculation

In the fill pattern algorithm, a list is used to store the flow front voxels. The cavity voxels can be classified into three groups, filled voxels, empty voxels and flow front voxels. The initial flow front voxels are gate voxels with the initial incoming vector. In each

calculation step, the flow front voxels will fill some empty voxels and generate new flow front voxels. This calculation is continued until the whole cavity is filled.

A flow front voxel had a direction (vector) but no speed in the original algorithm. In another word, all flow front voxels had the same speed. In the new algorithm, speed is included (qualitatively) and each flow front voxel may have a different speed. Both speed and the direction of flow may change during cavity filling. There are three possibilities that cause speed change:

a) When the flow hits an obstruction. As the flow vector changes, the speed drops due to energy loss;

b) When a voxel is filled by more than one voxel. The speed of outgoing voxel is calculated based on the speed of incoming voxels;

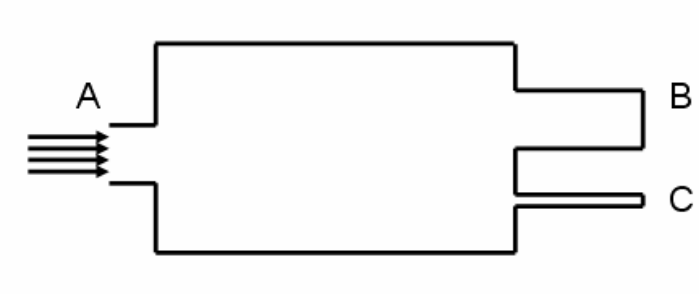
c) When flow resistance (discussed in detail later) changes due to local geometry the flow speed then may change.

The speed is also rounded to a small number of speed levels to save the computation and storage expense. This is reasonable since we are calculating the fill sequence and not the dynamic change with time. Furthermore, we can freely scale all the speeds of flow front voxels at the same time, which allows us to normalize the speeds to a fixed range. For example, if all the speeds are too small or too large, we can always normalize them to the range of 0 ~ 100.

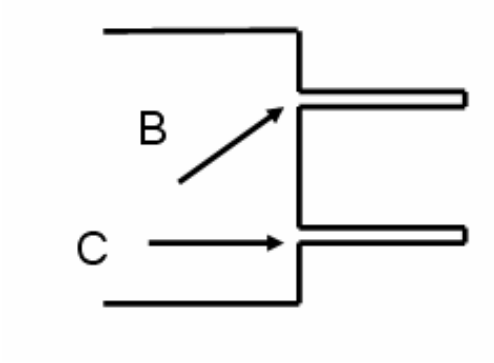
2) Flow resistance

A flow resistance for each flow front voxel is defined based on local geometric characteristics. One example of flow resistance is shown in Fig. 4.4. Flow is coming from A and will fill B earlier than fill C since flow resistance at C is larger than that at B. Flow speed at C is reduced due to larger flow resistance if the thickness of C is small enough. Apparently the flow resistance is related to cavity openness. The more open, the smaller the resistance. In addition, it is related to the flow vector. For example, in Fig. 4.5, flow B will have larger resistance than flow C when they are filling the two ribs because there is an angle between flow B and the rib, which makes it hard for flow B to fill the rib.

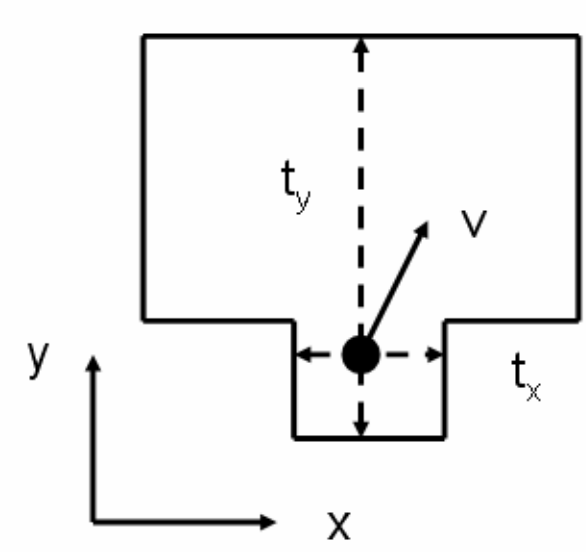
An initial definition of cavity openness might be part wall thickness. However, part wall thickness does not represent the cavity openness very well. An example is the part shown in Fig. 4.4 where the part is flat and the thickness in Z direction (the direction coming out from page) is the same. The wall thickness will be the same everywhere but the openness is different in different regions. For example, the openness at B is larger than that at C. However, the variation in openness of different parts of the same shape with differing thickness would be captured by wall thickness.



Section I-Figure 4.4: If flow coming from entrance A to fill ribs B and C. Flow will fill B earlier than fill C because B is more open and has small resistance.



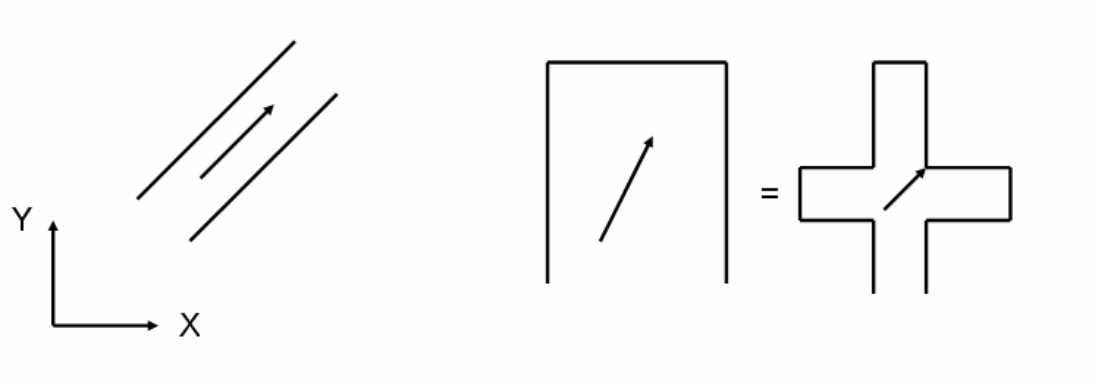
Section I-Figure 4.5: Flow C will fill rib earlier because flow B has larger resistance than C due to the angle between flow B and the rib.



Section I-Figure 4.6: Second method to calculate flow resistance based on thickness and speed components along x, y and z directions.

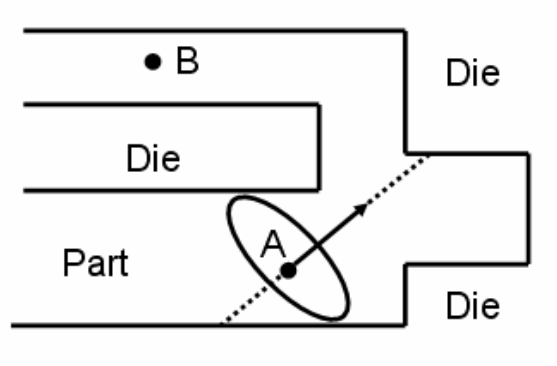
To calculate the flow resistance, we need to consider both the local openness and flow vector at the same time. The second attempt to calculate flow resistance is shown in Fig. 4.6. The thickness along x, y and z direction is calculated separately, denoting as t_x, t_y and t_z. The speed components along x, y and z direction are also calculated as v_x, v_y and v_z. The flow resistance is then defined as:

$$r = \frac{1}{|t_x v_x| + |t_y v_y| + |t_z v_z|} \tag{4-1}$$



Section I-Figure 4.7: New problems for second definition of flow resistance. In the left example, the flow resistance calculated using Eq (4-1) is larger than it should be. In the right example, both cases would have the same flow resistance, which is not correct.

Using this definition, the problem of the flat part with the first definition can be eliminated since the thickness at different location is distinguished. However, it would cause other problems, which are illustrated in Fig. 4.7. In the left example, if both the flow vector and part wall is inclined with respect to the coordinate axes, the calculation result from Eq. (4-1) is large because t_x and t_y are both small. However, the actual flow resistance is small because the wall open direction is the same of flow vector. In the right example, calculation results using Eq. (4-1) of both cases are the same because the thickness component at x and y direction are the same. However, it is apparent that the openness of two cases is different. Therefore, the second definition can not represent the flow resistance correctly.



Section I-Figure 4.8: Third definition for flow resistance. The resistance is calculated based on local geometry openness corresponding to flow vector.

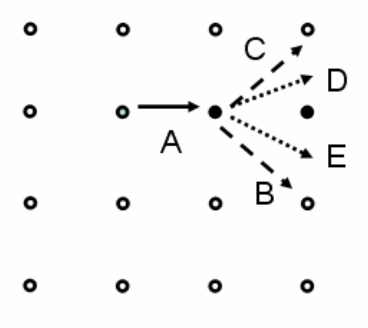
The third resistance thus is defined as inverse of the multiplication of part voxel number along flow vector and part voxel number at the plane perpendicular to flow vector. This can be shown by the example in Fig. 4.8 to calculate the flow resistance at flow front voxel A with a flow vector. A flow line is first constructed along flow vector (shown as the dash line in Fig. 4.8) and we count the number of part voxel along the local region at this flow line, denoting the number as *m*. Then a flow plane through voxel A but perpendicular to the flow line is constructed, shown as an ellipse in Fig. 4.8. We count the number of part voxel in local region at the flow plane, denoting as *n*. Note that voxel B is not counted because it is not in the local region even it is at the flow plane. A local region means there is no die voxel between the current voxel and the target voxel. Then the flow resistance is computed as $r = 1/(m \times n)$. By this definition, we can address both local openness and flow vector. The flow resistance then can be used to compute flow speed.

Actually, we only need the inverse of flow resistance, $m \times n$ during the calculation. We define the flow potential (not to be confused with potential energy) as $p = s \times m \times n$, where s is the speed. In each calculation step, we first calculate p for every flow front voxel then take the overall average potential $\overline{p}$ as the threshold. If the potential of a flow front is larger than the threshold, this voxel is said to have enough potential to move. If not, this voxel is held into the next step until it has larger potential than the threshold. In addition, since there are only limited flow vector in discrete space, we can pre-compute the locations of voxels for each flow line and each flow plane associated with each flow vector and store them in a table. When computing flow resistance, we only need to check directly if the voxels in the table are cavity voxels. Further, we only check voxels in the flow plane within a certain range, say, 10×10 voxels around the flow front voxel to save computation.

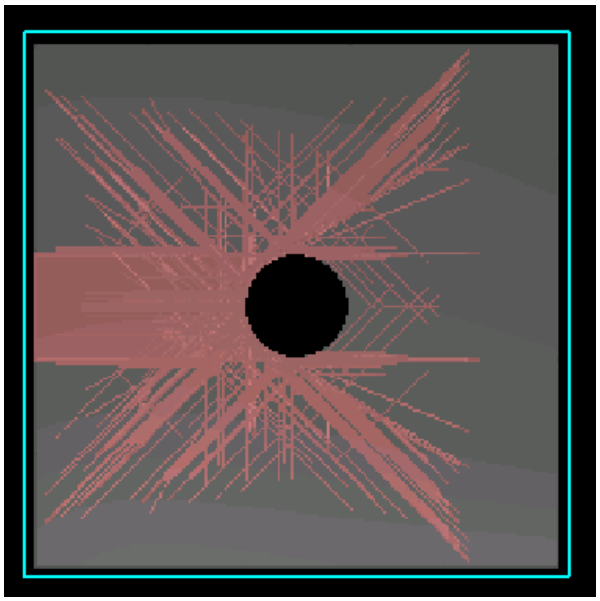
3) Vector change at obstruction

When a flow hits an obstruction, the flow vector is going to change. In the previous algorithm, the new flow vector was calculated by simply constructing a vector from the current flow front voxel to the empty voxel. With the improvement, this is changed to the sum of original vector and position vector (vector from current voxel to empty voxel). In Fig 5-9, solid dots represent obstructions and blank dot represent empty voxels. A flow front voxel has the vector A and hits an obstruction. In the old algorithm, there were two new vectors, B and C while in the new one, the two new vectors are D (sum of A and C) and E (sum of A and B). In this way, the new direction is not only related to position but also related to the original direction, which gives a better way to calculate new direction. This can be considered to include the effect of inertial term, which is relative to the

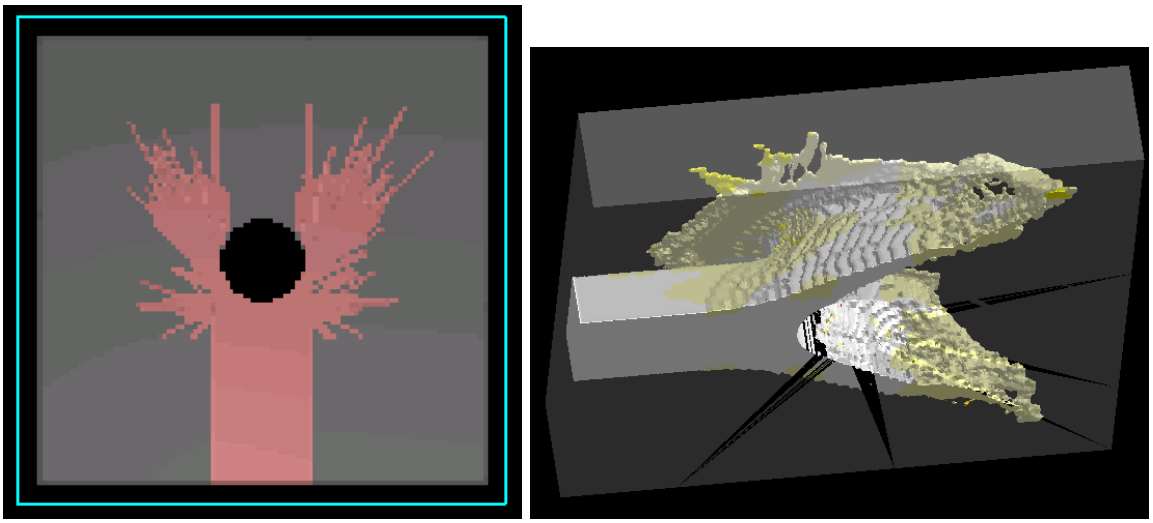
original direction. The calculation results using the old and new algorithms are shown in Fig. 4.10 ~ 4.11 using a plate part with a through hole, where the hole serves as the obstruction. When the flow hits the obstruction, it should disperse more like the pattern shown in Fig. 4.11 than Fig. 4.10.



Section I-Figure 4.9: New vector calculation, where A is oncoming vector, B and C are vectors based on available voxels, D and E are calculated outgoing vectors.



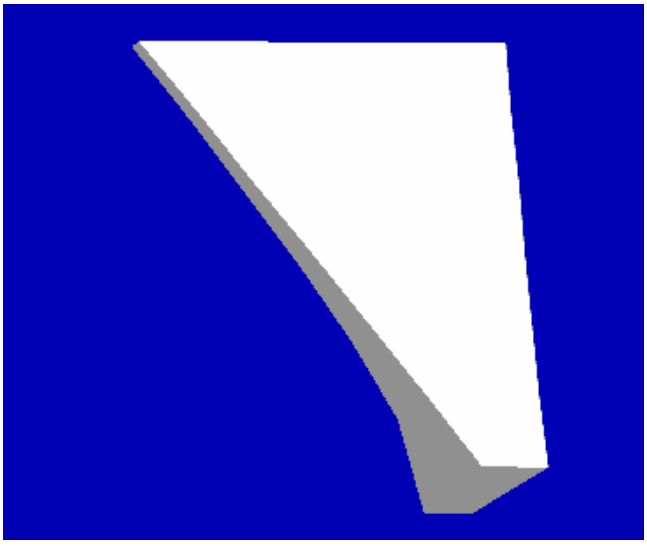
Section I-Figure 4.10: Flow pattern when hitting an obstruction when using old algorithm for outgoing vector calculation.



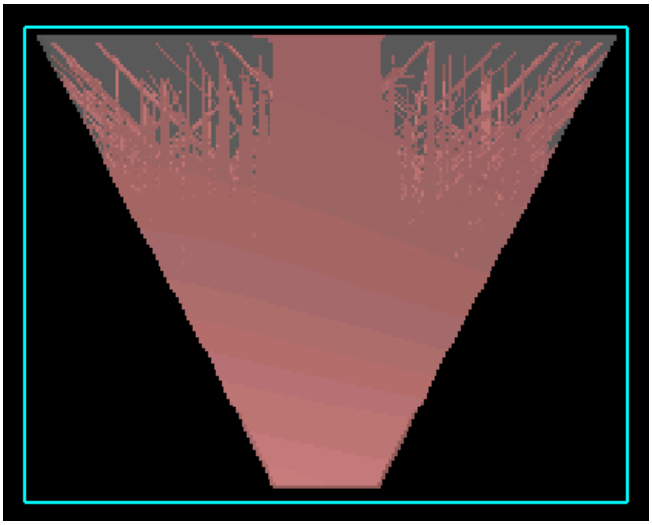
Section I-Figure 4.11: Flow pattern when hitting an obstruction. Left: using new algorithm for outgoing vector calculation; Right: MagmaSoft result.

There is a pre-computed angle table to search empty voxels when a flow hits obstructions. The table contains voxel locations of 13 different angles, which is between current flow vector and vector to empty voxels, ranging from 35^{o} to 180^{o}. The neighboring empty voxels can be found through this angle table. The angle search starts from smallest angle to largest angle. Whenever there is an available angle, which means there is an empty voxel, the search stops and takes this empty voxel to calculate the new vector in the old algorithm. In other words, the search only selected one angel as the output. As a result, some available vectors are ignored and the new vectors are not the complete possible vectors. Fig. 4.12 shows a fan gate to be filled. The flow should move smoothly from back to front and the flow front should be nearly a “flat line”, which is the design intent of the fan gate. However, as shown in Fig. 4.13, the result from original algorithm is not an anticipated.

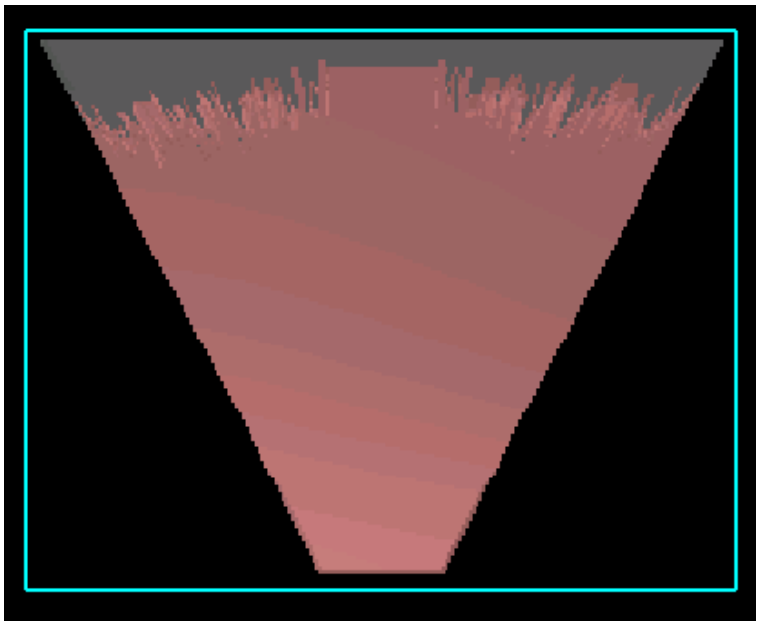
The reason for this is that only one angle was considered to calculate new vectors. As the flow advances, the thickness is smaller and smaller then the resistance becomes larger and larger. The flow would move to any possible direction. If there is one angle limitation, the flow front becomes irregular, as shown in Fig. 4.13. In the new algorithm, more available angles are provided to be searched. The results for the fan gate for 2, 3 angles and 4 angle searching are shown in Fig. 4.14 ~ 4.16. It can be seen that result from 2 angle searching is much better than one angle searching. Result of 3 angle searching is better than 2 angle searching while result of 4 angle searching is better than 3 angle searching. However, there is efficiency penalty to search more angles. Compared to the original run time, the run time on 2, 3 and 4 angel searching are about 25%, 40% and 50% more, respectively. For the tradeoff, we choose 2 angel searching since it is much better than one angle searching and only slightly increases the run time.



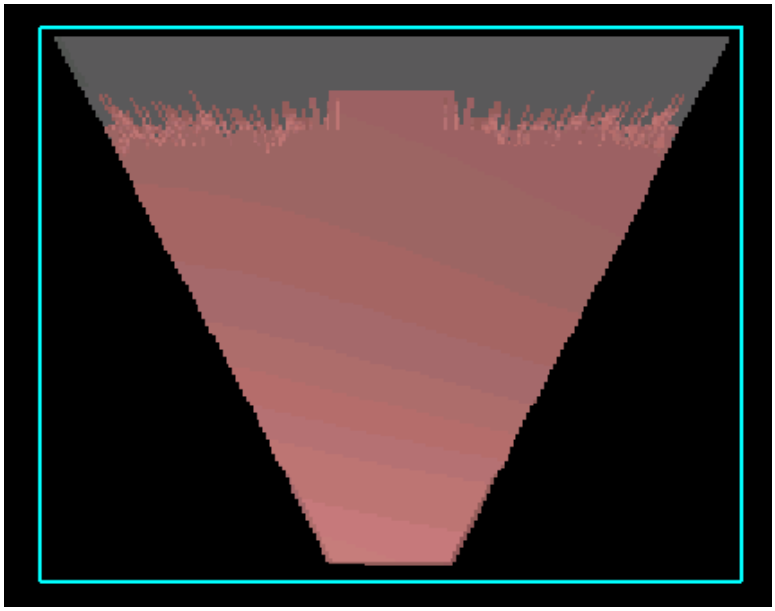
Section I-Figure 4.12: Cad Drawing of Fan Gate



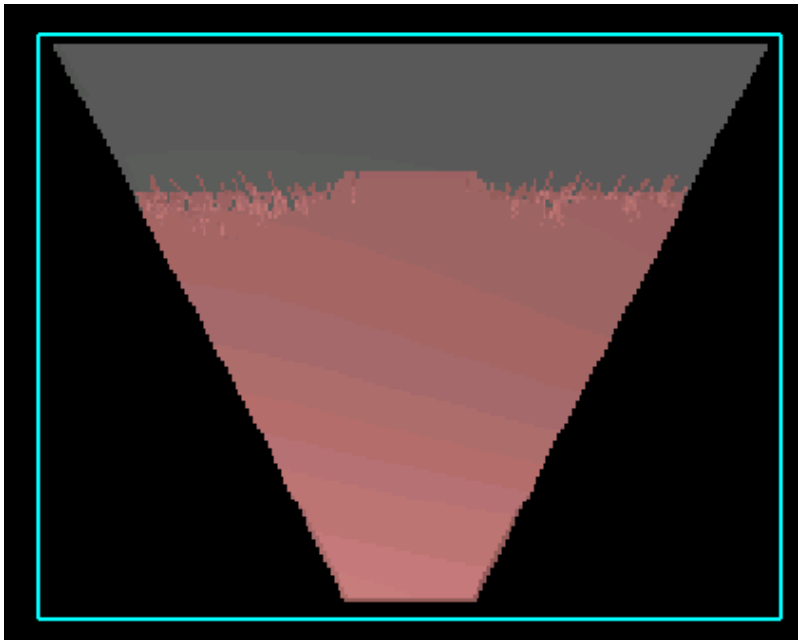
Section I-Figure 4.13: A fan gate to be filled and the result using original algorithm with only one angle search



Section I-Figure 4.14: Fill pattern result using two angle searching. The flow front is flatter than using one angle searching.



Section I-Figure 4.15: Result of three angle searching which is better then two angle searching



Section I-Figure 4.16: Result of four angle searching which is better than three angle searching

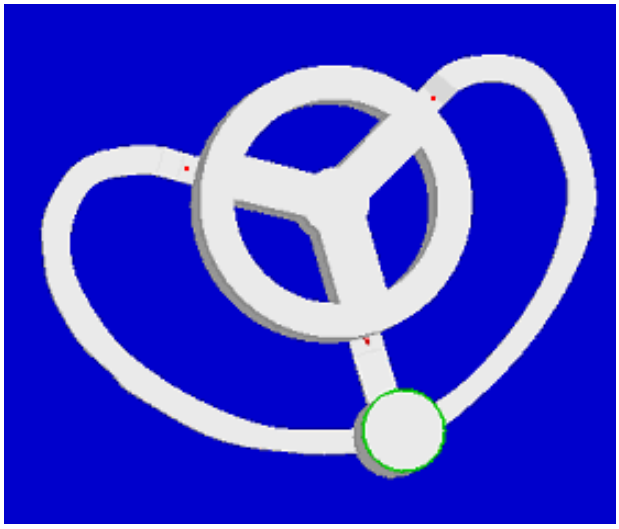
Flow speed is also affected when flow hits obstruction since there is energy loss. The change of speed is related to the angle between the original vector and the new vector. The larger the angle is, the more the speed decreases. For example, the speed loss for the 180^o angle should be larger than that for the 35^o angle. There are totally 13 angles in the angel table in ascending order and they are grouped based on the angle value. The reduced speed for each group is also in ascending order.

4) Multiple gates

If a part has multiple gates with different distances to the biscuit, the flow may or may not start to fill cavity simultaneously. If the runner system is well designed, the cavity fill will start from gates simultaneously due to the large resistance at gates. For the example shown in Fig. 4.3, cavity fill will start after the runner system pressurizes which should not happen until the runner system is nearly full if this is a properly designed runner system. Even if melt reaches gate C before the other two, the cavity will start to fill only to a small degree because the gate resistance is significantly higher than the runner resistance. However, in the original algorithm, due to different fill distance of OA, OB and OC, the melt reaches C earlier than it reaches A and B. The melt will then fill the cavity at C before it reaches A and B, which will makes an incorrect prediction for the fill pattern if this is a well designed runner system.

The new algorithm provides an option to allow the user to specify if he wants the cavity fill to start at multiple gates simultaneously. In the example of Fig. 4.17, three gates (identified with a red dot) and the biscuit (in green circle) are specified, which indicates this is a properly designed runner system. During the fill pattern calculation, the surface

analysis is done first to find the wall thickness information. For each gate point, the local region with the same thickness is found and is set to be a gate region. All gate voxels are assigned a very large flow resistance, which will force the flow to stop at gate region until the runner system is filled. The flow can start to fill cavity simultaneously from gate regions. Fig. 4.18 shows the result if the user does not specify the multiple gates while Fig. 4.19 shows the result if he does. It is clearly shown that in Fig 5-18, the three flows start to fill cavity almost simultaneously.



Section I-Figure 4.17: Three gates are specified with red spots. The green circle denotes the biscuit where the initial flow front starts from. By specifying these, the user wishes the cavity fill start at multiple gates simultaneously.

Section I-Figure 4.18: Result using the usual way. The cavity fill starts from central runner before other runners have not been filled.

Section I-Figure 4.19: Result if the user specifies the multiple gates. Cavity fill starts from three gates simultaneously after three runners are all filled.

5) Bias removal

The fill pattern for a symmetric part should also be symmetric (for example, the flat part with hole in Fig. 4.10). However, the analysis result from old algorithm was not symmetric. Careful inspection found that it was caused by the calculation sequence. As mentioned earlier, there is a list to store the flow front voxels. At each calculation step, the new flow front voxels are generated based on the sequence of old flow front voxels, i.e. from the front to the end. Since the number of voxel can be filled in a step is fixed due to mass conservation, there is fill competition for flow front voxels, which may generate fill bias. For example of Fig. 4.10, if in the initial flow front list, voxels at left side are the front and voxels at right side are the end, the voxels at left side will always be filled earlier than those at right side. As the calculation continues, there is more and more bias. The solution is to reverse the sequence at each step. At an odd step, the calculation is from the front of flow front list to its end while at an even step, it is from the end to the front. Thus the bias can be removed.

6) Influence of neighboring voxels

In the old algorithm, the calculation for each flow front voxel was independent of its neighboring flow front voxels. There was no influence between neighboring voxels. As a result, the flow lines were often dispersed. This differs from numerical simulation. Furthermore, occasionally there were small flows advancing too far compared to others, which should be incorrect. An example of old algorithm was shown in Fig. 4.10. Even after we apply the new calculation of vector (shown in Fig. 4.11), the flow lines are still too dispersed. In the new algorithm, the influence from neighboring voxels is addressed

by grouping flow front voxels and balancing their vector and speed, which helps reduce the dispersal. The flow front voxels are first grouped based on their neighbor relationship. In each group, the average vector and speed are calculated. Each flow front voxel is then slightly adjusted to the average vector and speed. This way does not eliminate the problem completely but can help balance the behavior of voxels within neighborhood.

7) Efficiency issue

Special attention was paid to efficiency for implementation of new algorithm. It is undesired that run time increases significantly due to the added computation. Some methods are applied to reduce computation: 1) Pre-computed tables of geometry information are used. 2) Real data are rounded to integers. 3) Variables are saved for later use. 4) Data structure and computation are optimized. With these treatments, the run time increase for the new algorithm is 20% ~ 50% compared to the old.

4.4 Conclusions

A new algorithm for visualization for fill pattern was designed based on the old algorithm. There are some improvements:

1) Speed calculation is included. The change of speed of flow front is affected by local flow resistance.

2) Flow resistance and flow potential are calculated during the analysis. Three definitions have been tried to model flow resistance. Flow potential determines whether a flow front has enough “potential” to move.

3) Vector change calculation at obstruction was redesigned. The simple examples of plate with hole and fan gate demonstrate the improvement.

4) The user is provided the option to specify multiple gates, which ensures cavity fill start at multiple gates simultaneously.

5) Fill pattern bias due to competition between flow fronts to fill empty voxels is removed by switching the sequence of flow front list.

6) Influence of neighboring voxels is considered. Special attention is paid to computational efficiency issue.

5. EXAMPLES AND VERIFICATION OF VISUALIZATION FOR FILL PATTERN

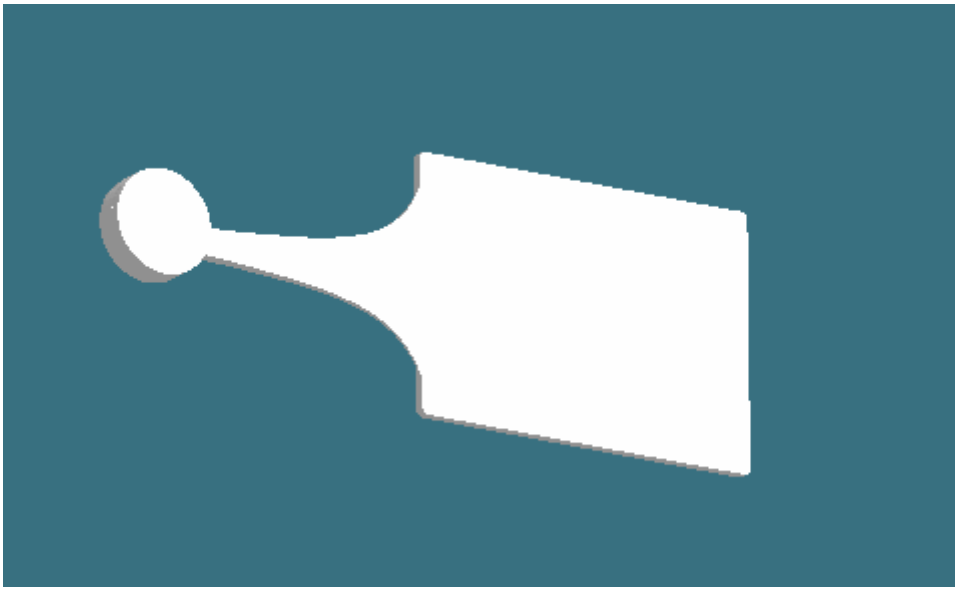
5.1 Implementation

The program for fill pattern visualization has been integrated into the *CastView* software using Visual C++ on MS Windows platform. The examples in this chapter were run on a computer – 2.4 GHz Pentium IV PC with 1 GB RAM. The typical run time of CastView is about 10 minutes with resolution of 200 voxel along the maximum dimension. Four cases are demonstrated in this chapter. Three of them are compared to numerical simulation, including a simple part, a medium complex part and a very complex part. The other is compared to water analog.

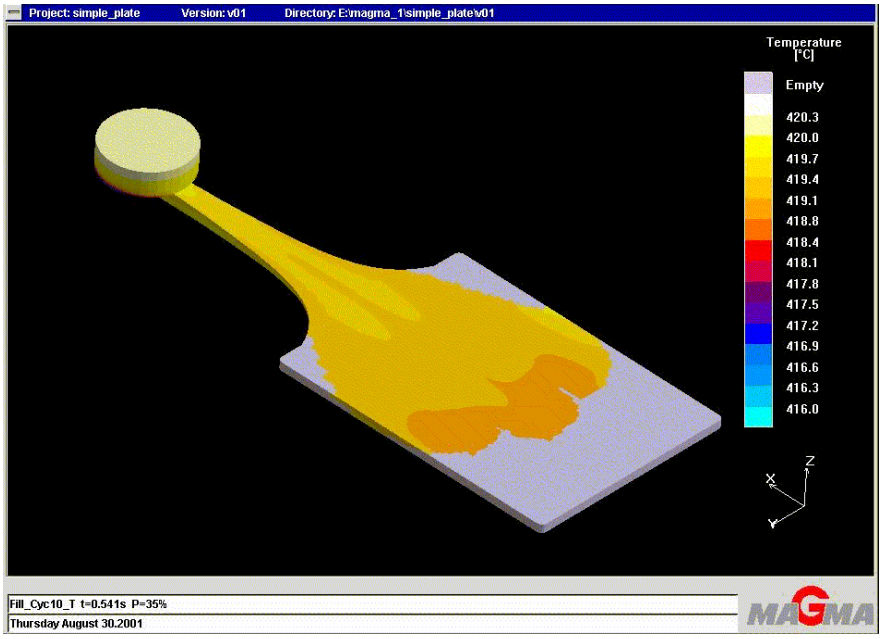
5.2 Case 1: Simple Plate Part

The comparison was done between numerical simulation and qualitative reasoning since the accuracy of numerical simulation is higher. The first part is a simple plate part. The numerical simulation was performed on MagmaSoft by Ms. Haijng Mao. The run time on MagmaSoft was ~2 hours and the run time on CastView was only ~ 2 minutes. The part geometry is shown in Fig. 5.1, in CastView rendering. The results from MagmaSoft are shown in Fig. 5.2 and Fig. 5.3 with the courtesy of Ms. Mao. From the MagmaSoft results, it can be seen that the flow fills the central region first and then the two far corners. The near two corners are the last regions to be filled. The visualization results from CastView using old algorithm are shown in Fig. 5.4 and Fig. 5.5 while the results using new algorithm are shown in Fig. 5.6 and 5.7. It can be seen that in the new results, the flow also fills the central region first and then far corners, finally near corners. Both

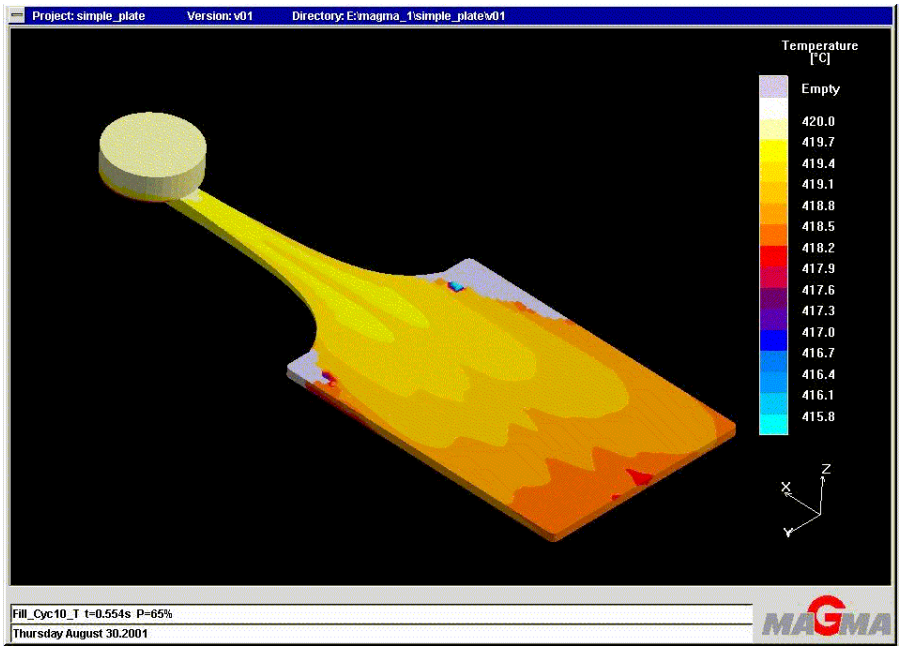
patterns of MagmaSoft result and new algorithm results are very similar. However, the old results are not similar to those of MagmaSoft. In the old results, the flow fills the near corners earlier and the last region to be filled is the central region, which is not correct. This comparison suggests the improvement.



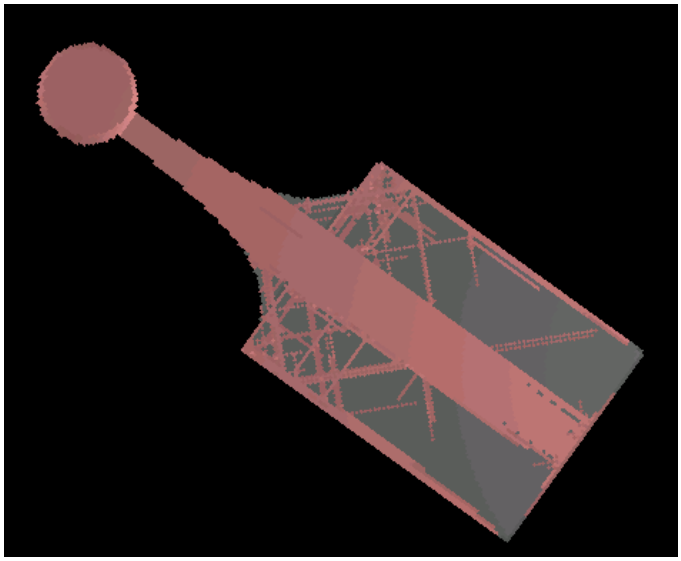
Section I-Figure 5.1: Simple plate part to be examined by MagmaSoft and CastView.



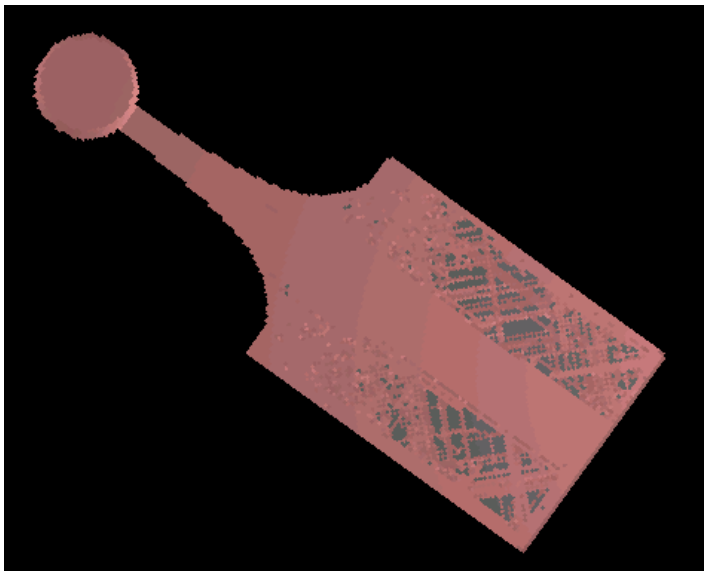
Section I-Figure 5.2: MagmaSoft result for simple plate part (with courtesy of Mao)



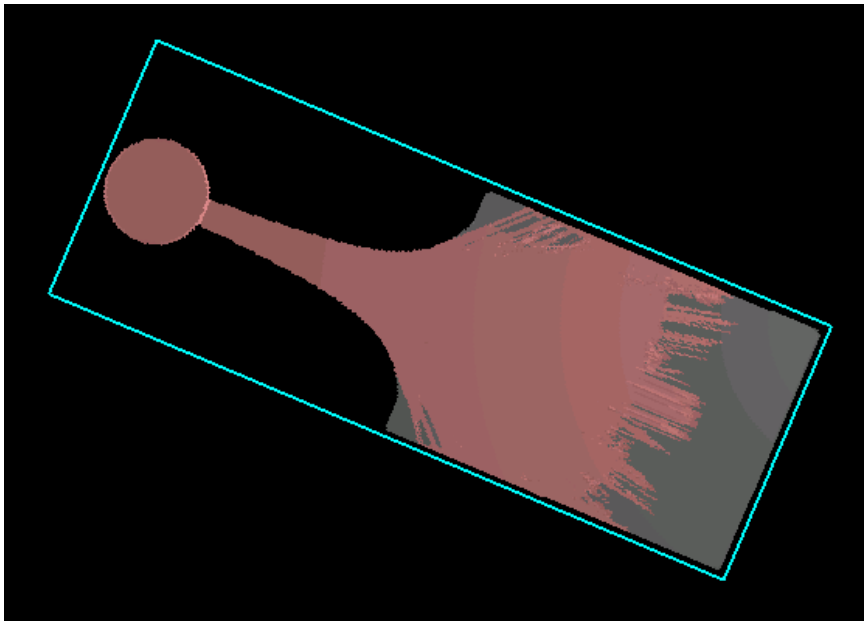
Section I-Figure 5.3: MagmaSoft result for simple plate part (with courtesy of Mao). The two near corners are the last regions to be filled.



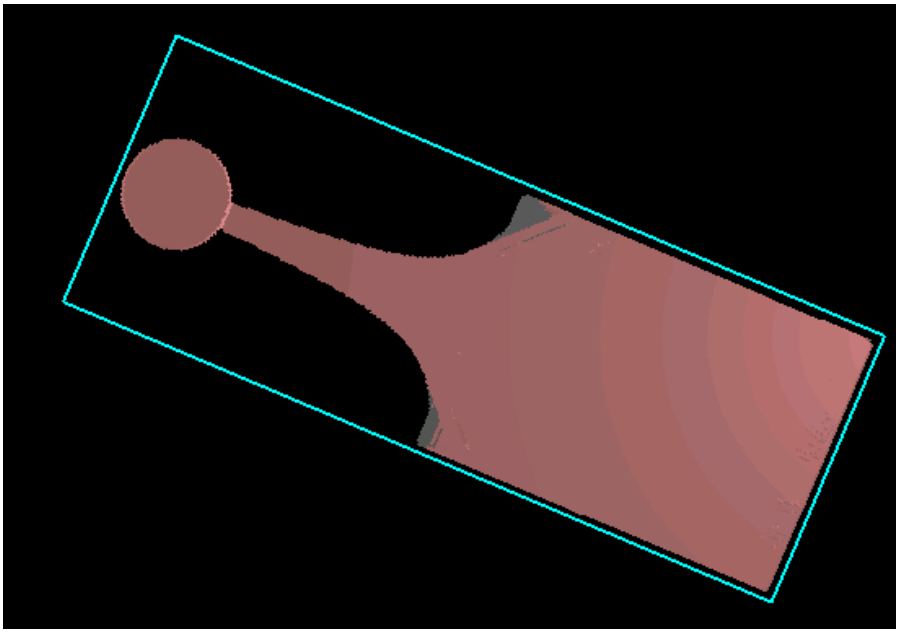
Section I-Figure 5.4: CastView result using old algorithm. The two near corners are filled too early.



Section I-Figure 5.5: CastView result using old algorithm. The last region to be filled is the central region, which is not correct.



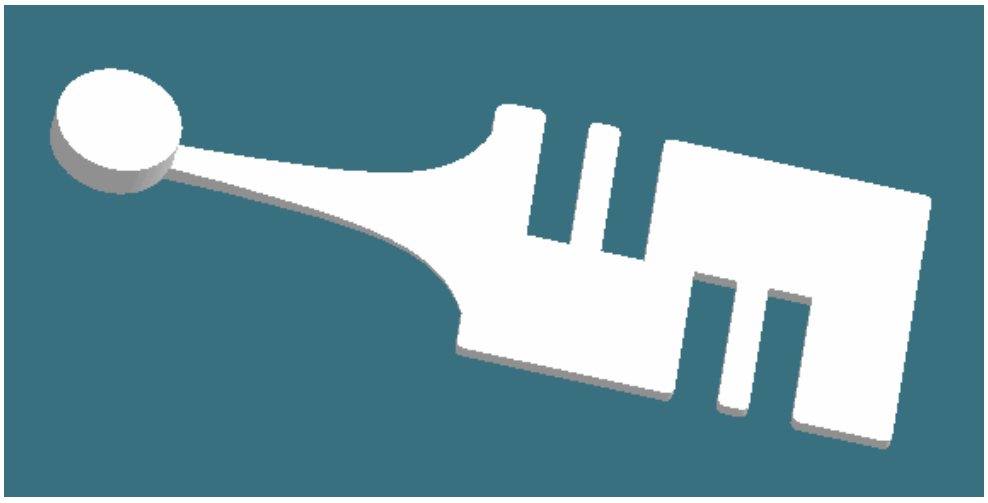
Section I-Figure 5.6: CastView result for simple plate part using new algorithm.



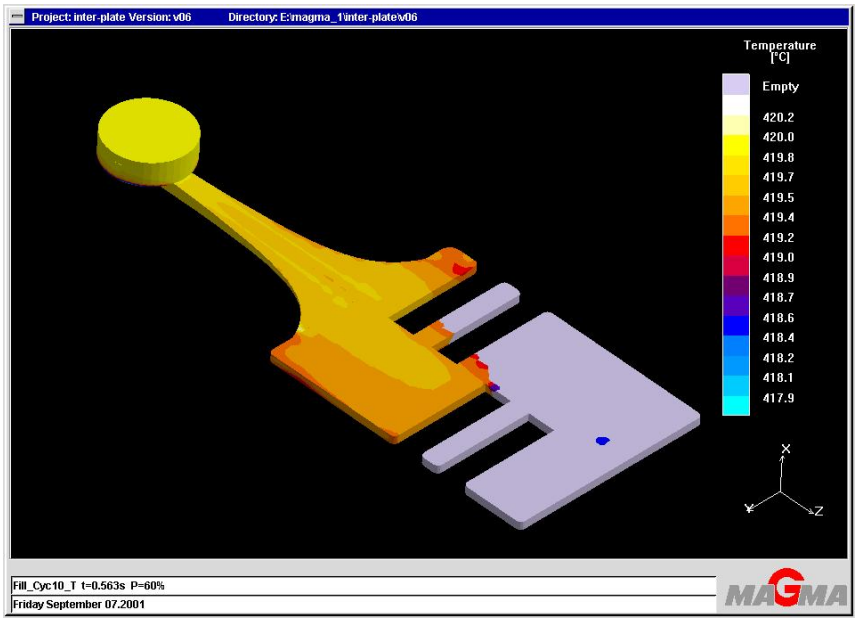
Section I-Figure 5.7: CastView result for simple plate part using new algorithm. The last region to be filled is the two near corners, which is similar to the result from MagmaSoft.

5.3 Case 2: Part With Fingers

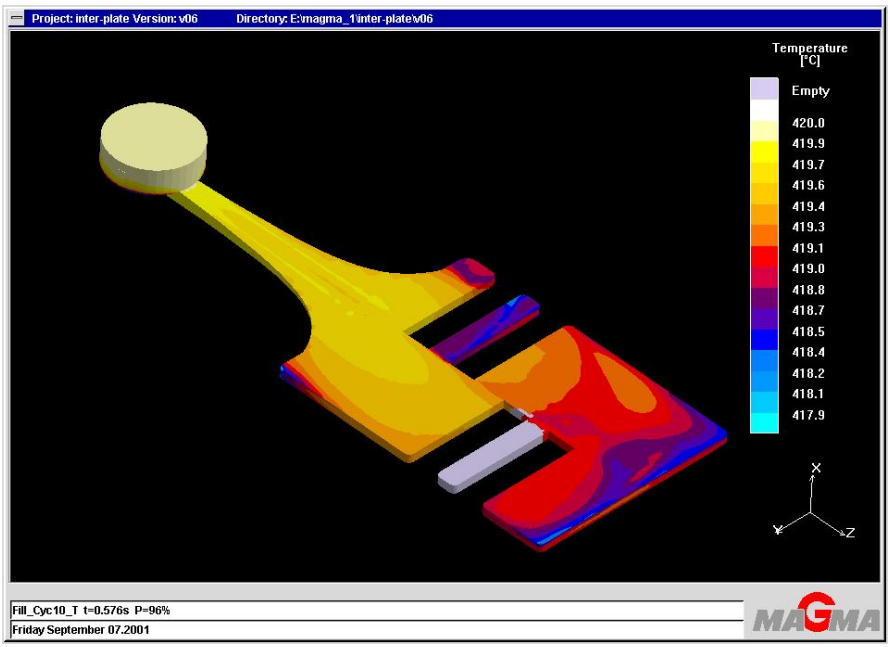
The second part to be compared is a part with two fingers at both sides. The 3D part geometry is shown in Fig. 5.8. Again the numerical simulation was done on MagmaSoft. The run time on MagmaSoft was ~3 hours and the run time on CastView was ~6 minutes. The results from MagmaSoft are shown in Fig. 5.9 and Fig. 5.10. It can be seen that the first finger is filled simultaneously with the second half of part and the second finger is the last region to be filled. The results from CastView using old algorithm are shown in Fig. 5.11 and Fig. 5.12 while the results using new algorithm are shown in Fig. 5.13 and 6-14. In the old results, the second half starts to be filled before the center of first half is filled, which is not similar to that of MagmaSoft. The results using new algorithm has similar pattern with that of numerical simulation. The first finger is not filled until the second half of part is filled. The second finger is the last region to be filled.



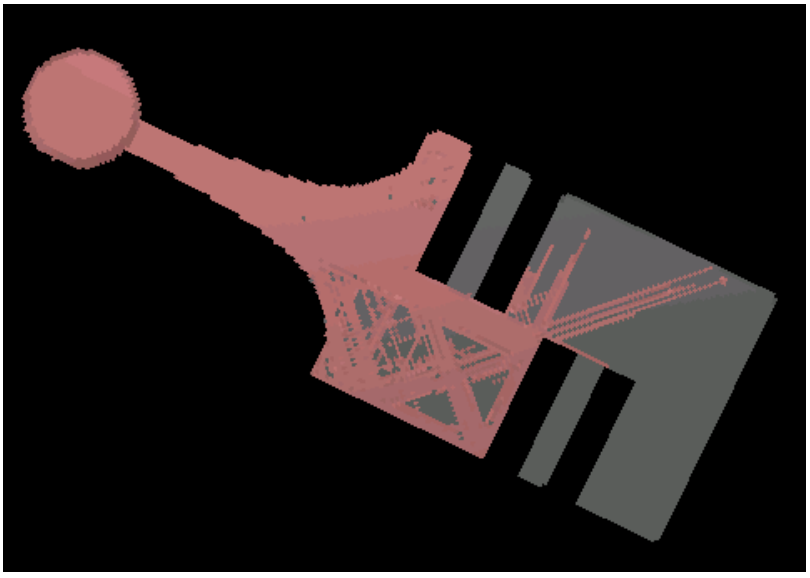
Section I-Figure 5.8: A part with two fingers at different sides



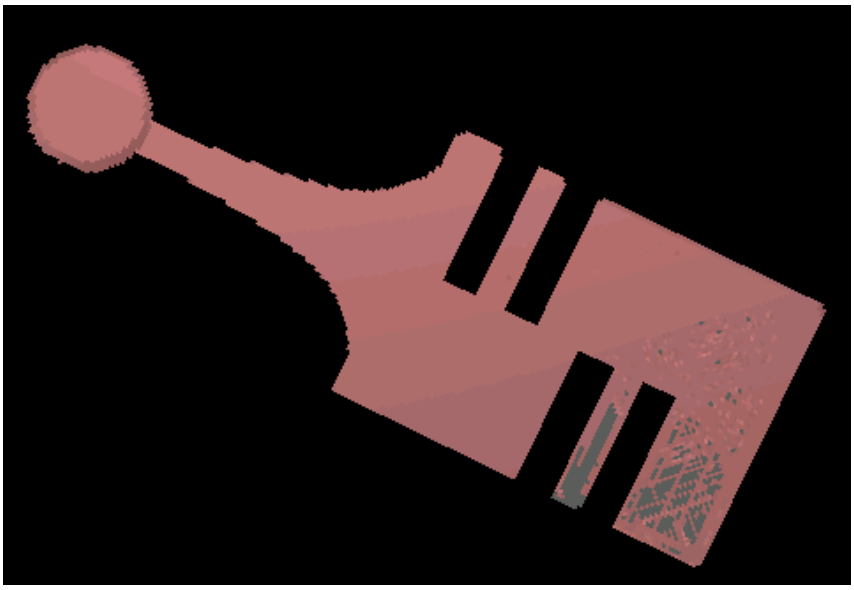
Section I-Figure 5.9: MagmaSoft result for finger part (courtesy of Mao).



Section I-Figure 5.10: MagmaSoft result for finger part (courtesy of Mao). The second finger is the last region to be filled.



Section I-Figure 5.11: CastView result for finger part using old algorithm.



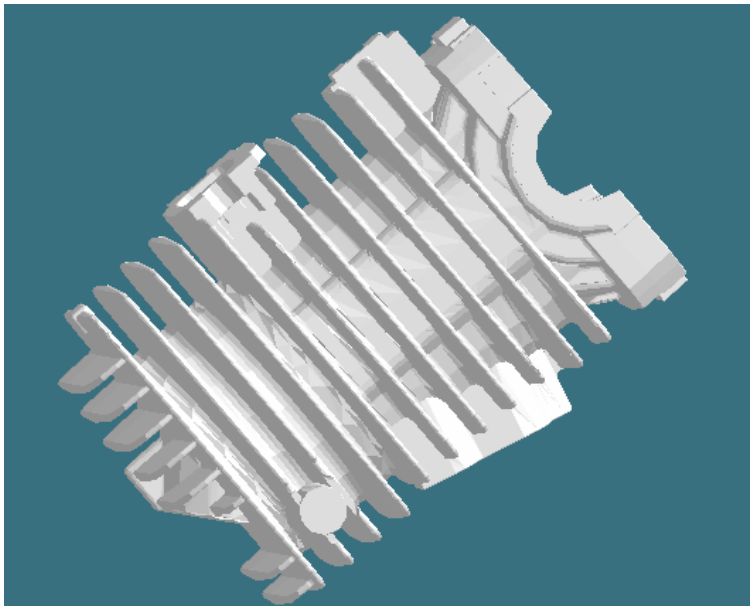
Section I-Figure 5.12: CastView result for finger part using old algorithm. The second finger is not the last region to be filled.

Section I-Figure 5.13: CastView result for finger part using new algorithm.

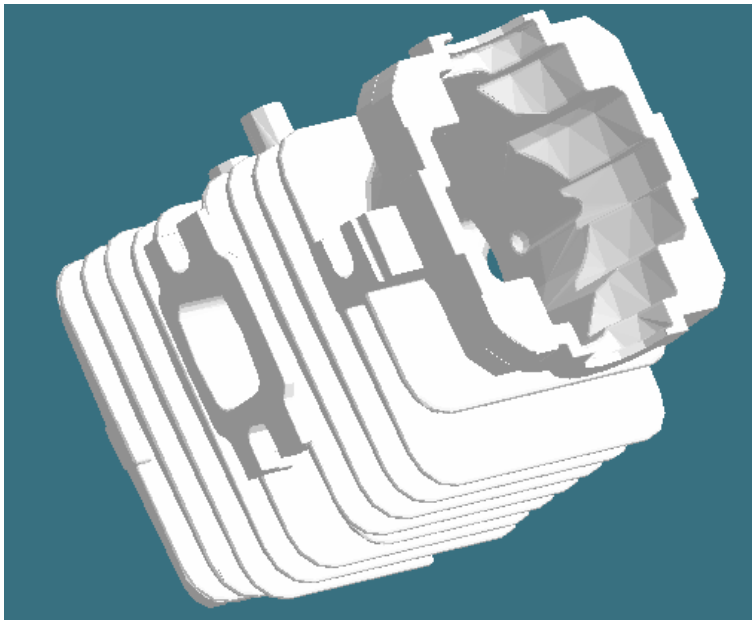
Section I-Figure 5.14: CastView result for finger part using new algorithm. The second finger is the last region to be filled.

5.4 Case 3: Transfer Case

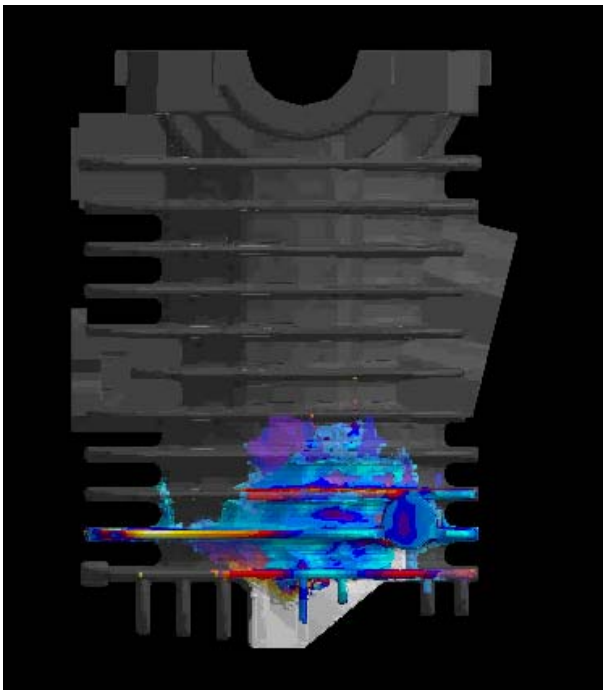
The third part for test is a relatively complex part with relatively thin wall. The 3D part geometry is shown in Fig. 5.15 and Fig. 5.16. The part is hollow with a number of very thin ribs while the part wall at bottom is thick, which may cause fill related problems in production. The results from MagmaSoft are shown in Fig. 5.17 ~ 5.19 (kindly provided by Dr. Walter Smith, DCD Technology Inc.). The results from CastView using old algorithm are shown in Fig. 5.20 ~ 5.22 while the results using new algorithm are shown in Fig. 5.23 ~ 5.25. It can be seen that the results using old algorithm does not have similar fill pattern with that from numerical simulation in that the last region to be filled is not the same. However, the results using new algorithm have similar pattern to those of MagmaSoft. The flow fills central region and ribs first then the upper left region is the last one to be filled. These results for the complex part show again the improvement for geometric reasoning algorithm for fill pattern.



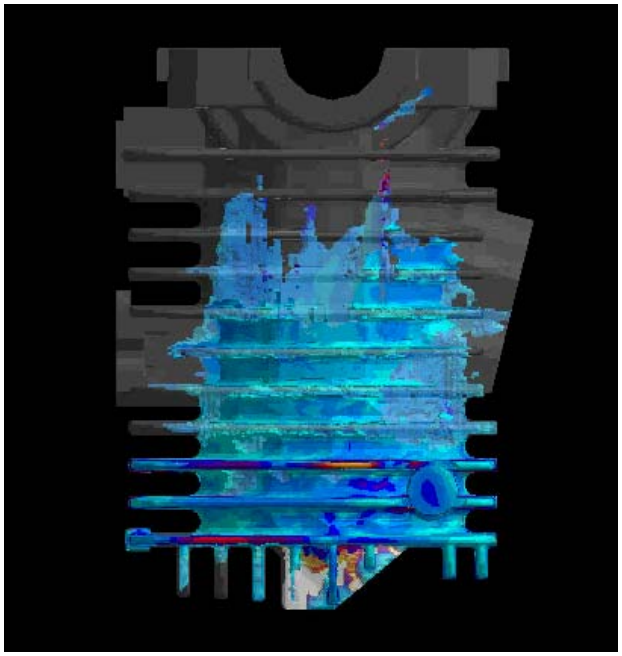
Section I-Figure 5.15: A very complex transfer case part for test. There are many thin ribs at around body surface.



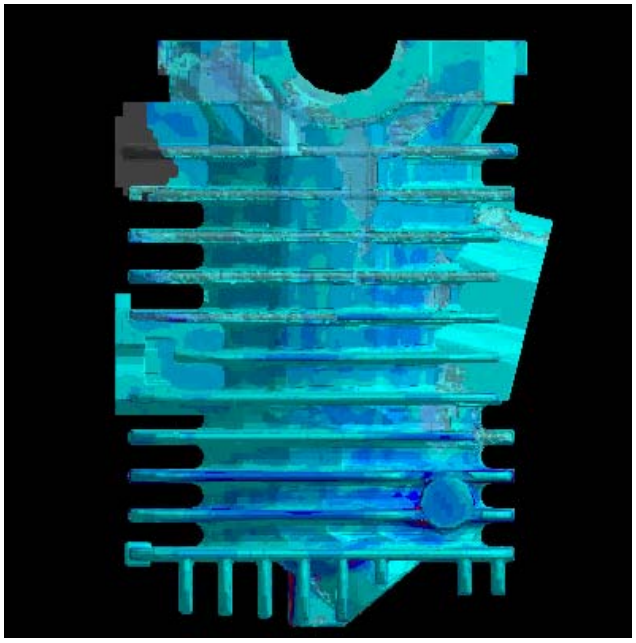
Section I-Figure 5.16: Another view of the transfer case part for test. The part is hallow and the wall thickness is uneven.



Section I-Figure 5.17: Result from MagmaSoft (courtesy of Dr. Smith)

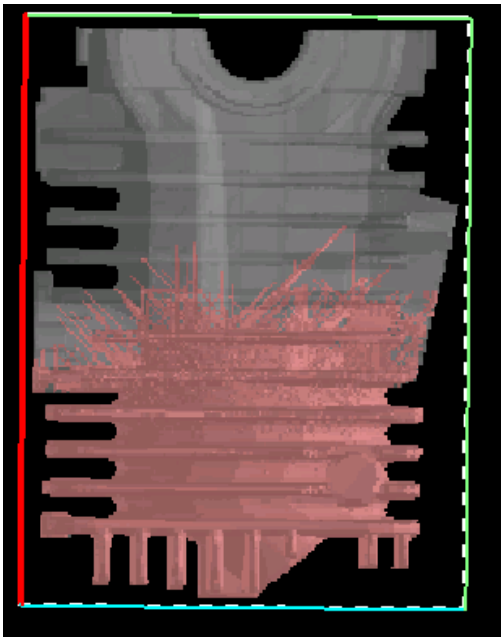


Section I-Figure 5.18: Result from MagmaSoft (courtesy of Dr. Smith)

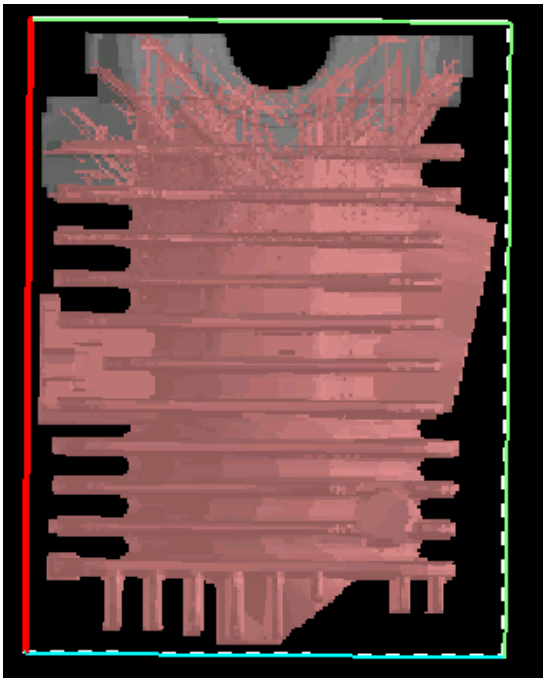


Section I-Figure 5.19: Result from MagmaSoft (courtesy of Dr. Smith)

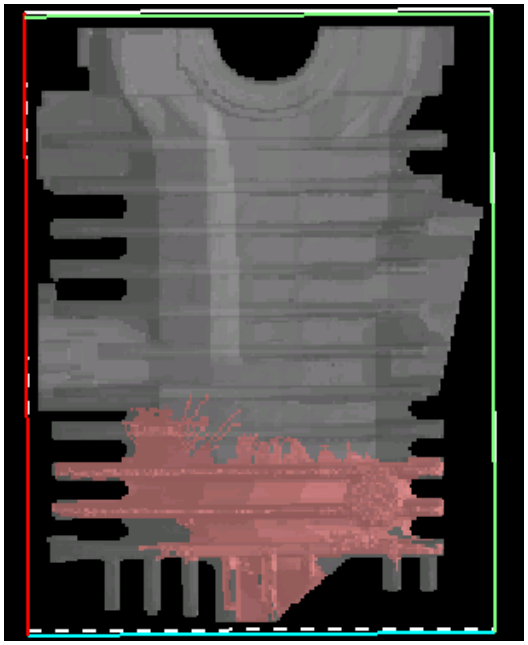
Section I-Figure 5.20: Result of transfer case from CastView using old algorithm.



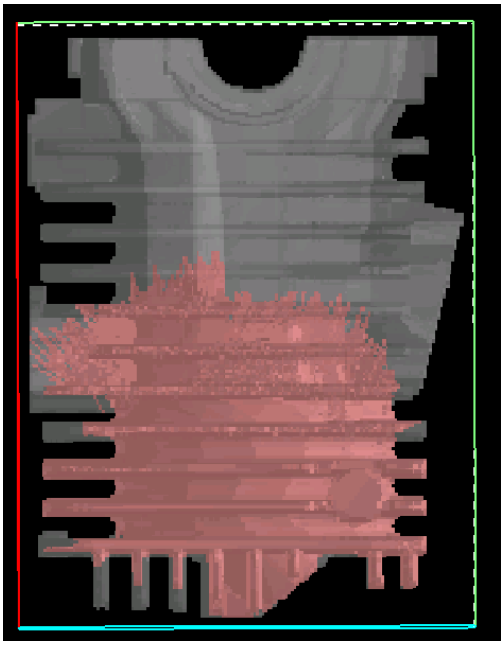
Section I-Figure 5.21: Result of transfer case from CastView using old algorithm.



Section I-Figure 5.22: Result of transfer case from CastView using old algorithm.



Section I-Figure 5.23: Result of transfer case from CastView using new algorithm.



Section I-Figure 5.24: Result of transfer case from CastView using new algorithm.

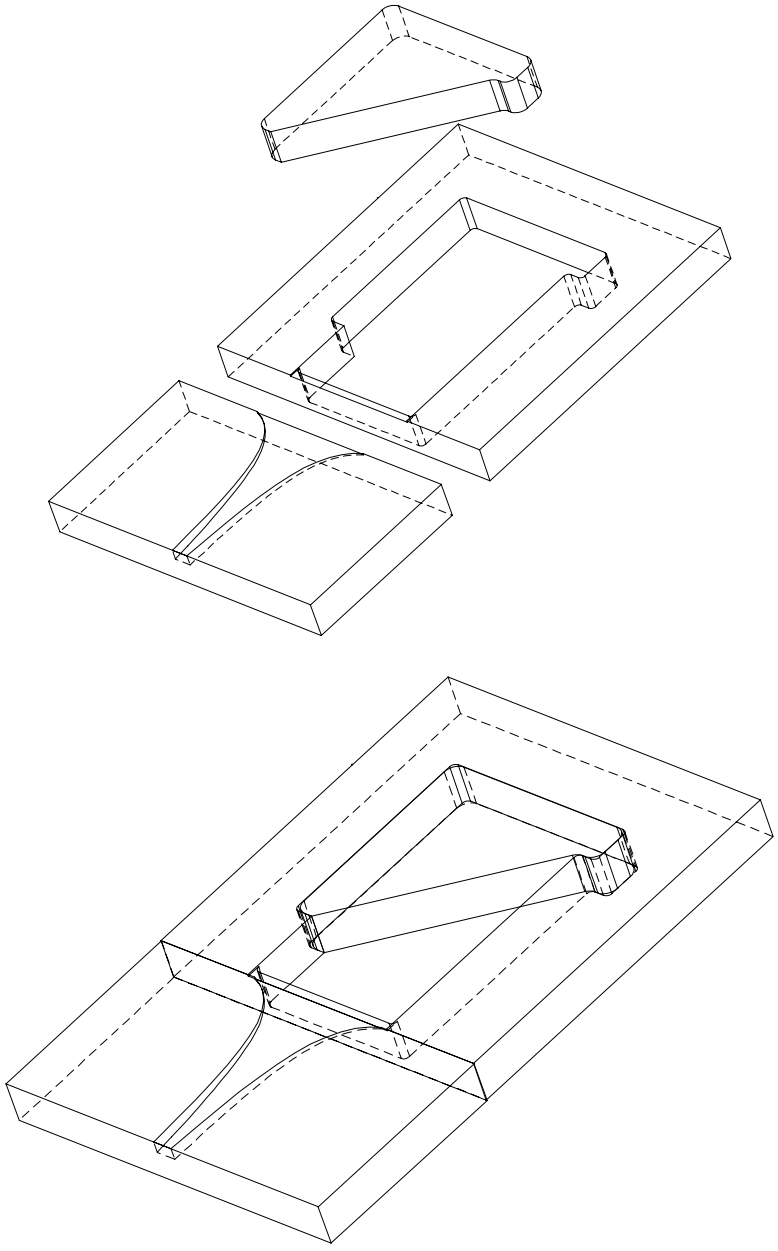
Section I-Figure 5.25: Result of transfer case from CastView using new algorithm.

5.5 Water Analog Study for Verification

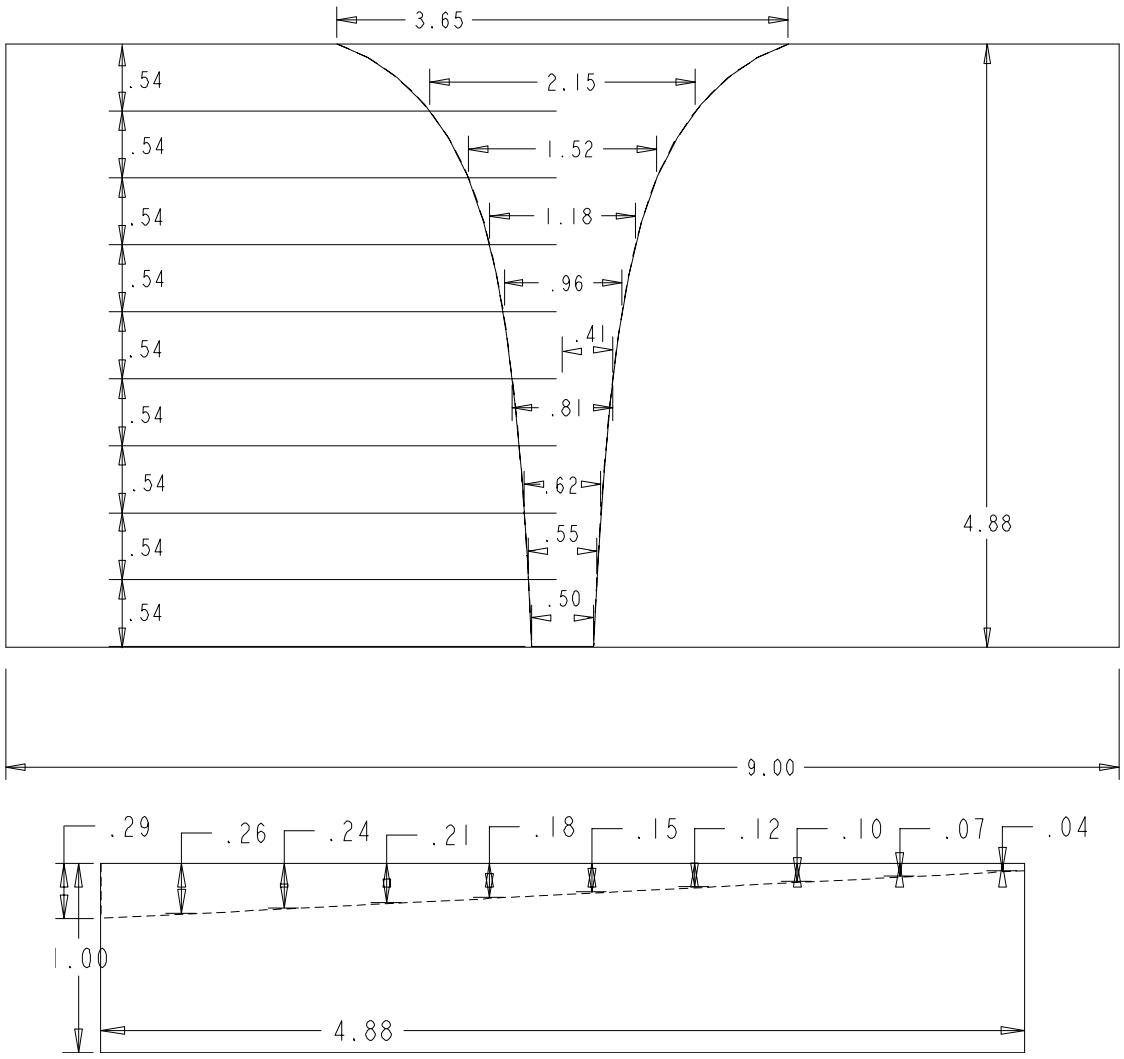
Since it is extremely difficult to run a real experiment on die casting process to obtain fill pattern result, water analog study is an important way to verify the fill pattern algorithm. Though the water analog model can not represent the heat transfer and solidification effects, it still has good similarity with the flow behavior of liquid metal in die cavity.

The water analog model results in this section are from Rebello's dissertation (Rebello 1997). Two cases of experiment with different cavity were run. One has an insert in the cavity and the other does not. The gate, cavity, insert and the assembly way are showed in Fig. 5.26. The dimensions of gate and runner are shown in Fig. 5.27 and those of cavity and insert are shown in Fig. 5.28.

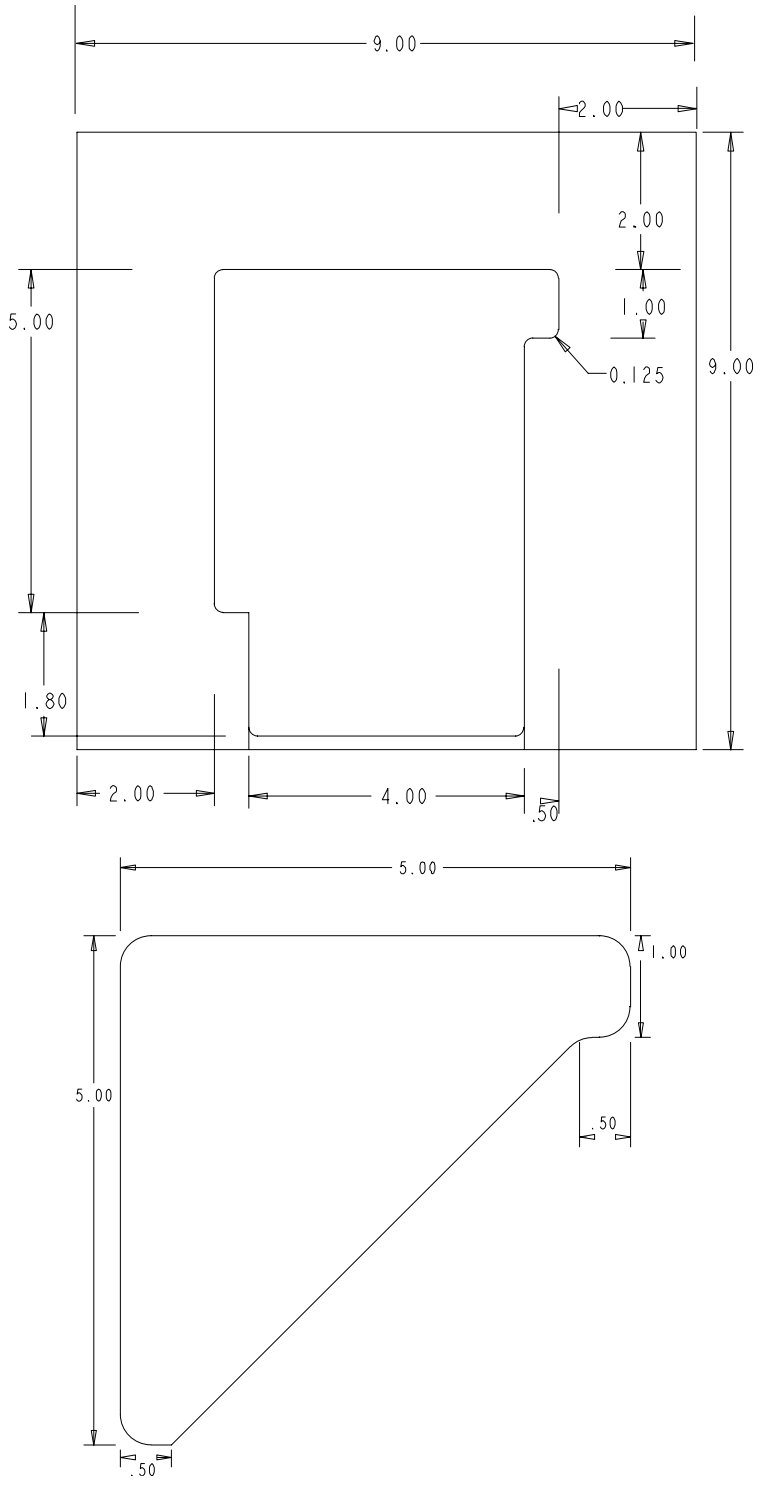
CAD model was built using Unigraphics NX and the STL model was exported for CastView analysis. Rendering of the two cavities (with and without insert) are shown in Fig. 5.29.



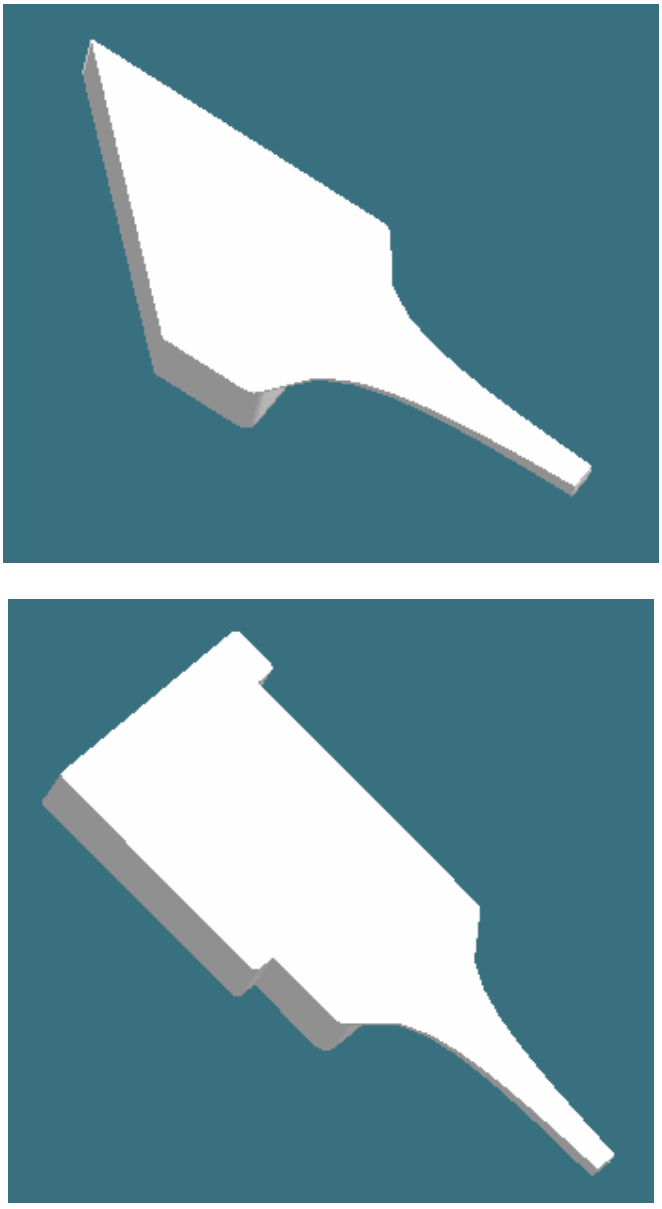
Section I-Figure 5.26: Illustration of the gate, cavity, insert and assembly way of water analog model. (Rebello, 1997)



Section I-Figure 5.27: Dimensions of gate and runner (in inch). Upper: top view; Lower: side view. (Rebello, 1997)



Section I-Figure 5.28: Dimensions of cavity, die and insert in inch. The depth is 1 inch. Upper: cavity and die; Lower: insert. (Rebello, 1997)

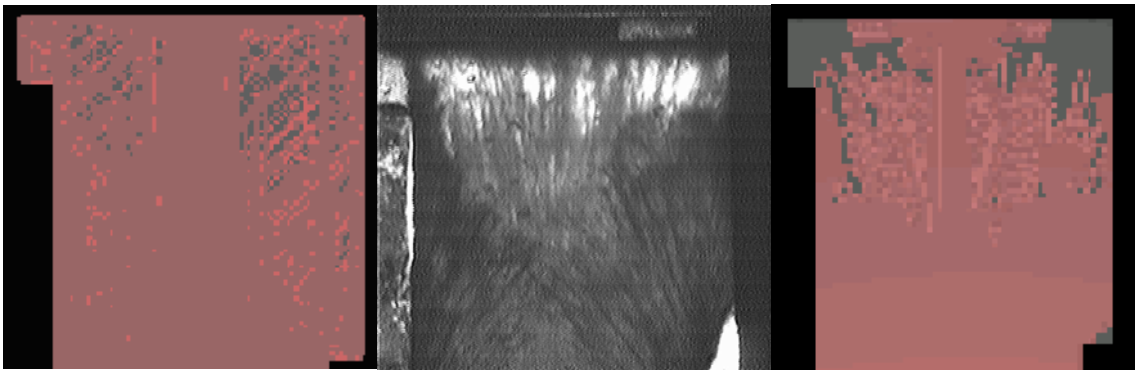


Section I-Figure 5.29: Two cavities for water analog experiment and fill pattern reasoning. Upper: with insert; Lower: without insert.

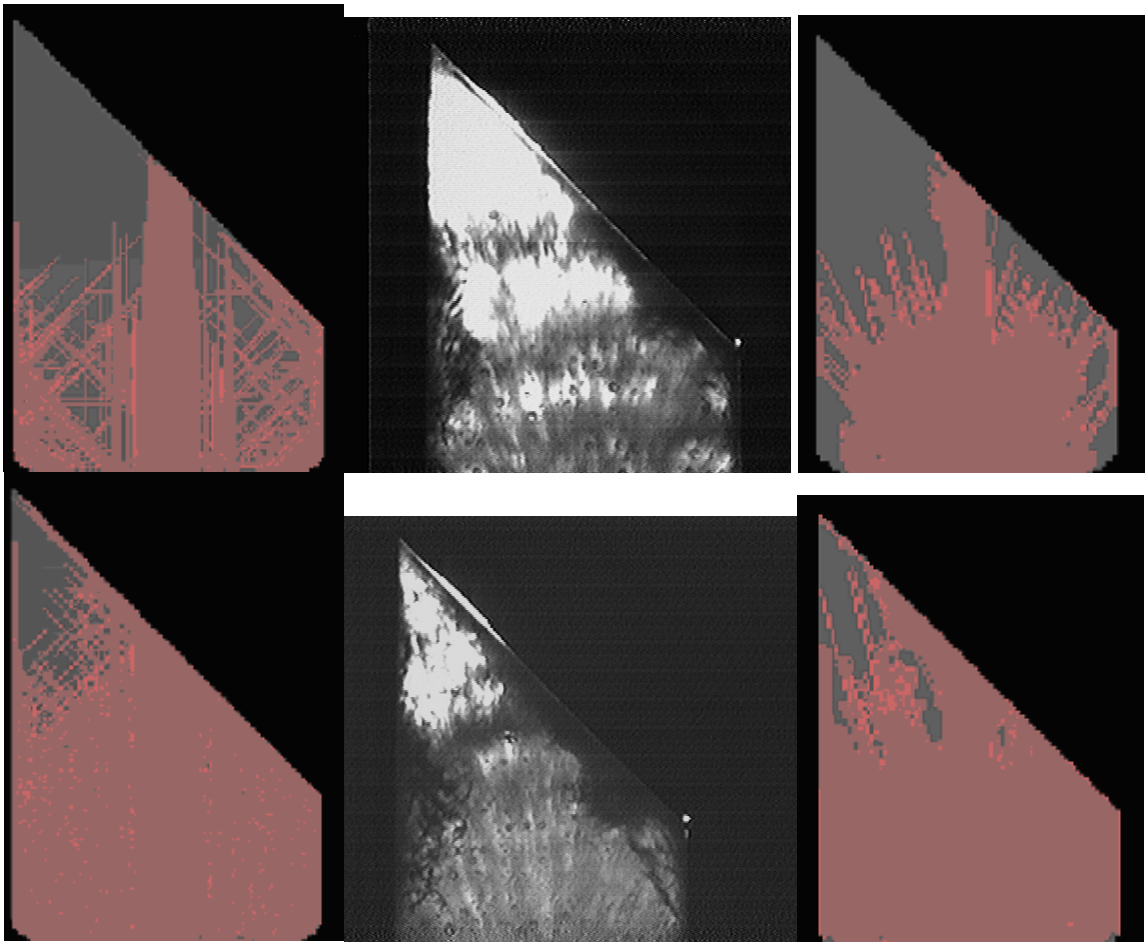
The experiment results and CastView results using old algorithm and new algorithm for the case without insert are listed in Fig. 5.30, (Left: old algorithm; Middle: water analog result from Rebello's dissertation; Right: new algorithm). It can be seen from the photo

taken from the experiment that the flow fills the central region first. Some flow is touching the end wall and starts to fill the end region with the region close to end wall unfilled. Both old result and new result follow this pattern but the new result is more similar to the experiment result. For example, the experiment result shows that the region close to gate is completely filled at this stage. The new result matches this pattern but the old result does not. The region close to gate has not been filled completely yet in old result. In addition, both experiment result and new result show that the small corner at the end has not been filled at this stage. However, the old result shows this corner has been filled, which is not correct.

The similar comparison is made on the case with insert, which is listed in Fig. 5.31. The fill patterns of three results are similar. Flow hits the 45^{o} angle wall then changes the direction. The small tip region is the last region to be filled. Comparing the old result and the new result, a slight difference is that in the new result, the flow fills the central region directly but in the old result, the flow fills the central region and the edge region at the same time. In addition, the flow lines in the old result are dispersed. Judging from the experiment result, the new result is slightly better than the old result.



Section I-Figure 5.30: Experiment result and CastView results using old algorithm and new algorithm for cavity without insert. Left: old result; Middle: experiment result (Rebello, 1997); Right: new result.



Section I-Figure 5.31: Experiment result and CastView results using old algorithm and new algorithm for cavity with insert. Left: old result; Middle: experiment result (Rebello, 1997); Right: new result.

5.6 Conclusions

1) Some cases are chosen for test and verification for the new fill pattern algorithm. These include three cases compared with numerical simulation results and two cases compared with water analog results.

2) The three parts compared to MagmaSoft simulation include a simple part, a medium complex part and a very complex part. The run time for a reasoning analysis is minutes and the run time for a numerical simulation is hours.

3) The results from new algorithm and results from numerical simulation have good agreement but there is some difference for the results from old algorithm. This shows the improvement of the new algorithm.

4) In the two cases compared to water analog results, one has an insert in the cavity and the other one does not. The three results from old algorithm, new algorithm and experiment are similar. However, the results from new algorithm are slightly better than those from old algorithm. The comparison shows the improvement of the new algorithm again.

6. CONCLUSIONS AND FUTURE WORK

Die casting is an important net shape manufacturing process. In this process, liquid metal is injected at high pressure and high speed into a metal die cavity, with subsequent solidification into useful shapes. The applications of die castings have increased significantly in the recent years, especially in the automobile, aerospace, electronics and medical apparatus industries. The die casting industry is now in fast development and will see its further expansion in the future.

The thermal characteristics and fill pattern are usually of concern to die casters. The temperature distributions of die and part are closely related to stress/strain, distortion, casting quality, die life, etc. In a good process design, the die temperature distribution should be as even as possible to reduce stress/strain and the part ejection temperature should be below the melting point of casting alloy for safe ejection. It is very desirable to have temperature pattern of die and ejection temperature of part when the process reaches the quasi steady state because they help the cooling/heating design and cycle design.

The typical way to obtain the thermal profile of die/part is numerical simulation for dynamic temperature change. This technique usually sets up equation system based on dynamic heat transfer and solidification then solves the equation system for the temperature change. However, this is very time consuming since to reach the quasi steady state, literally hundreds of cycles of simulation are needed. In addition, the information that numerical simulation provides is beyond that needed for cycle and die

cooling/heating design. Thus, numerical simulation is not suitable in the conceptual design stage.

Equilibrium temperature solution may be better than dynamic temperature solution for cycle and die cooling/heating design. The equilibrium temperature is defined as the time average temperature over a cycle of die and the ejection temperature of part after the process reaches the quasi steady state. To obtain the solution, the overall heat balance over the whole cycle is considered. The heat transfer concepts in real steady state are introduced into the quasi steady stage of die casting process. Since it does not solve the heat transfer in the cycles before the process reaches quasi steady state, this method is very quick and supports the interactive process design. This is particularly important in early design stage because there may be many alternative designs and the engineer wants to explore all designs using a quick computer tool.

Another concern of die casting engineers is the fill pattern of liquid metal in die cavity. The fill pattern also plays a very important role for casting quality. If the flow is not controlled well, it would cause many flow-related defects. One of the common defects is gas porosity, which usually happens at the last region to be filled. Thus, it is very important to evaluate the fill pattern of a process before the production.

Currently, the prevailing method to obtain the fill pattern is numerical simulation. This technique utilizes principles of heat transfer, solidification and fluid mechanics and is usually based on Finite Element Method (FEM), Finite Difference Method (FDM) or Boundary Element Method (BEM). Due to the complexity of the equation system to be

solved, numerical simulation is very time consuming. The computation time for a case using numerical simulation is usually hours or days. This is not suitable for quickly evaluating designs and limits the utility in the early stages of development.

An alternative method to obtain fill pattern is qualitative analysis which is based on geometric reasoning. In this method, the flow behavior is calculated using the cavity geometric information. The flow vector is recomputed when the flow hits obstruction. The mass conservation is considered within neighborhood. Some assumptions and simplifications are made to reduce the computation. Though the accuracy is not as high as numerical simulation, this method provides a quick and efficient way to predict the fill pattern in cavity.

The Ohio State University developed an algorithm based on geometric reasoning previously. This algorithm has proved a simple but efficient way for fill pattern. However, there are some shortcomings in this algorithm and occasionally the analysis results are not correct. Thus it is very necessary to correct the problems and improve the old algorithm.

In this research, an efficient algorithm to compute equilibrium temperature of die and ejection temperature of part has been developed. This algorithm was implemented in the program CastView based on Finite Difference Method (FDM). Compared to days of run time for numerical simulation for dynamic temperature solution, the typical run time using this algorithm for equilibrium temperature is only a few minutes. The results from

this algorithm have been compared to those from numerical simulation. Both results have good agreement, which indicate the validity and efficiency of the algorithm.

The old fill pattern algorithm for die casting process based on geometric reasoning has been redesigned. Many shortcomings in the old algorithm were fixed and improved. The new algorithm includes some other considerations which affect the flow behavior. The results produced by the new algorithm have been compared to those from numerical simulation and water analog model. The comparison clearly shows the improvement of the new algorithm.

The specific contributions made by this research are:

- Developed a mathematical algorithm to compute the equilibrium temperature of die and ejection temperature of part. The heat transfer concepts in steady state are introduced to quasi steady state to calculate the heat balance over a cycle.
- Special attention was paid to the calculation of heat released from the part. Several models were tried but finally the combined asymptotic and surrogate model was chosen.
- Composite heat transfer at interface, average temperature at different stages, cooling line effect, spray effect and die splitting at parting surface were addressed. Computational efficiency was considered by applying some special methods to reduce the computation.
- Implemented the algorithm in the program CastView, providing a quick tool to evaluate the cycle and die cooling/heating design. Compared to days of run time

using numerical simulation for dynamic solution, the run time of CastView for equilibrium temperature is only a few minutes.

- Developed a method to transfer date between FDM model and FEM model, thus related the equilibrium temperature data with numerical simulation temperature data.
- Redesigned the old fill pattern algorithm for die casting processes. Flow speed, flow resistance, flow potential and influence within neighboring flows were included. The method to compute new vector when flow hits an obstruction and the searching way for available vectors were improved.
- Implemented the new algorithm for fill pattern in the program CastView, providing a quick and efficient tool to evaluate fill pattern in cavity. The run time for a typical case on CastView is only ~10 minutes while that on numerical simulation packages is hours or days.
- Relating the fill pattern results from geometric reasoning with numerical simulation results.
- Relating the fill pattern results from geometric reasoning with water analog studies.
- Determined the dominant terms on flow behavior for die casting process, gravity casting process and squeeze casting process from Navier-Stokes equations.
- Developed fill pattern algorithms for gravity casting process and squeeze casting process by modifying the algorithm for die casting process.

- Implemented the algorithms in the program CastView. The typical run time is only a few minutes.
- Relating the fill pattern results of gravity casting and squeeze casting using geometric reasoning with the information in literature resource.

The tool for equilibrium temperature can quickly compute the overall temperature distribution of die and ejection temperature of part. This helps the user evaluate the effect of cooling and heating and verify the cycle design. Due to the efficiency of this tool, the user can explore numerous alternative designs, which is very difficult for numerical simulation packages.

The tool for fill pattern analysis provides an alternative means other than numerical simulation. The advantage of geometric reasoning is the computational efficiency. It can produce a fill pattern prediction in a few minutes. This is particularly useful in the early design stage. Compared with the old fill pattern algorithm, the new algorithm improves the calculation and considers more factors. The analysis results are more accurate but the computation time is only increased slightly.

Based on the analysis of Navier-Stokes equations and geometric reasoning technique, the fill pattern algorithms for gravity casting and squeeze casting can provide quick evaluation for fill patterns of these two processes.

The limitation of the analysis for equilibrium temperature is that we applied the steady state on the quasi steady state. This is to simplify the computation but also introduces some problems. The heat transfer is computed based on the time average temperature.

However, the actual heat transfer happens on the dynamic changing temperature. The discussion in Section 2.8 makes some remedy but it does not eliminate the problem.

The limitation for fill pattern analysis is that this method is solely based on geometric reasoning thus there is no or only little physics behind it. There is no strict consideration of mass conservation, momentum conservation and energy conservation. There is no calculation for heat transfer and solidification during flow analysis. This lack has inevitable effect on the result accuracy.

Extensions to the research on computing equilibrium temperature include the improvement of the current method to calculate heat released from part and better way to address the average temperatures in different cycle stages. This would improve the accuracy of the calculation, especially for the ejection temperature of the part.

Future work for fill pattern reasoning includes better definitions for flow resistance and flow potential. For example, flow resistance calculation should address the threshold of wall thickness which can affect the resistance. In the example of Fig. 5.4, if the thickness of C is large enough, it would not have resistance for the flow. Future work also includes the development of functions to define gate, biscuit and runner system. The fill pattern analysis will be performed on the whole cavity including part, gate, runner system and biscuit.

REFERENCES (Section I):

1. Allchin, T., Quality assurance with modern CNC die casting system, *Die Casting Engineer*, 1990, March/April, p34-38.

2. Barone, M. R. and Caulk, D. A., dieCas--Thermal analysis software for die casting: modeling approach, *Proceeding of 17th International Die Casting Congress and Exposition*, 1993.

3. Barone, M. R. and Caulk, D. A., Analysis of liquid metal flow in die casting, *International Journal of Engineering Science*, 2000, V38, p1279-1302.

4. Barton, Charles, The beginnings of die casting, *Die Casting Engineer*, 1991, September/October, p24-31.

5. Booth, Stuarte E. and Allsop, Derek F., Thermal control and design of dies, *Proceedings of 11th International Die Casting Congress and Exposition*, 1981, p1-6.

6. Cai, Li, Dexel-based Geometric reasoning and visualization for die configuration, PhD dissertation, The Ohio State University, 2002.

7. Chen, Haiqing; Li, Huaji and Cao, Yang, *Numerical Simulation for Casting Solidification*, First Edition, 1990, Chongqing University Press.

8. Chen, Xiaorui, Graphical user interface for cooling line functions and surface rendering, Thesis for Master's degree, The Ohio State University, 2002.

9. Elfakharany, Essameldin F., Qualitative reasoning for filling pattern in high-pressure die-casting and gravity-driven casting, PhD dissertation, The Ohio State University, 1999.

10. Ferziger, Joel H. and Peric, Milovan, Computational methods for fluid dynamics, Second Edition, 1999, Springer.

11. Granchi, M. and Vettori, E., Computer thermal analysis of die casting dies, *Proceeding of 12th International Die Casting Congress and Exposition*, 1983, p1-13.

12. Gu, JinSheng, Survey on die casting industry in China, Casting Committee of China Foundry Association, 1995, p12.

13. Ha, Joseph; Cleary, Paul; Alguine, Vladimir and Nguyen, Thang, Simulation of die filling in gravity die casting using SPH and MagmaSoft, *Proceedings of 2nd International Conference on CFD in the Minerales and Process Industries,* 1999.

14. Heine, R. W. and Uicker, J. J., Geometric modeling of mold aggregate, superheat, edge effects, feeding distance, chills, and solidification macrostructure, *AFS Transaction*, 1984, v92, p135-147.

15. Kami Buchholz, Magmisium on the move, *Automotive Engineering International Online*, 2001, http://www.sae.org/automag/material/07-2001/.

16. Kaye, Alan and Street, Arthur, Die casting metallurgy, Butterworths Scientific& Co. Ltd., London, 1982, p11.

17. Khayat, Roger E., A three-dimensional boundary element approach to confined free-surface flow as applied to die casting, *Engineering Analysis with Boundary Element*, 1998, v22, p83-102.

18. Kim, Chung-Whee and Ruhlandt, Gerald K, Computer modeling of thermal stress in die casting dies, *Proceedings of 13th International Die Casting Congress and Exposition*, 1985.

19. Kramer, Deborah A., Magnesium in 1997, *Magnesium Statistics and Information*, USGS, http://minerals.usgs.gov/minerals/pubs/commodity/magnesium/400497.pdf, Aug 1, 2004.

20. Lee, S. L. and Sheu, S. R. New numerical formulation for incompressible viscous free surface flow without smearing the free surface. *International Journal of Heat and Mass Transfer,* 2001, v44, n10, p1837-1848.

21. Lu, H. Y. and Lee, W. B., A simplified approach for the simulation of metal flow in a cylindrical sleeve die casting cavity, *Journal of Materials Processing Technology*, 1999, v91, p116-120.

22. Ma, Y., Qualitative geometric reasoning for thermal design Evaluation of die casting dies, *Ph.D. Dissertation*, 2000, The Ohio State University.

23. Miller, R. Allen, Lu, Shao-chiung and Alexander B. Rebello, Simple visualization techniques for die casting part and die design, Final report, DOE project # DE-FC07-95ID13362, 1998.

24. Nakamura, Shoichiro, Numerical method, Class notes, The Ohio State University, 2001.

25. Rebello, A. B., Visualization of the filling of die-casting dies, PhD dissertation, The Ohio State University, 1997.

26. Rosbrook, Christopher; Kind, Ralf; Maus, Wolfgang and Seefeld, Rudolf, Simultaneous advances in casting processes and casting process simulation create development synergy, *Die Casting Engineer*. 2000, January/February, p48-52.

27. Rosindale, I. and Davey, D., Steady state thermal model for the hot chamber Injection system in the pressure die casting process, *Journal of Materials Processing Technology*, 1998, v82, p27-45.

28. Sulaiman, S., Hamouda, A. M. S., etc., Simulation of metal filling progress during the casting process, *Journal of Materials processing Technology*, 2000, v100, p224-229.

29. Tuten, J. M. and Kaiser, W. D., Development of programs to use with programmable calculators for die-cooling-system design, *Proceedings of 10th International Die Casting Congress and Exposition*, 1979, p1-4.

30. Wallace, John F.; Chang, Qingming and Schwam, David, Process Control in Squeeze Casting, *Die Casting Engineer*, 2000, November/December, p42-48.

31. Wang, Junqing, The development survey of casting process CAD/CAE in the world, *Proceeding of 8th Annual Conference of Casting Association of China*, 1992, p45-58.

32. Zhang, Weishan, Study on numerical simulation of thermal field and mold filling of die casting process, *Ph.D. Dissertation* (Chinese), Tsinghua University, China, 1996.

33. http://www.egr.msu.edu/~pkwon/msm481/casting.ppt, Aug. 9, 2002.

34. http://www.hks.com/, Aug. 9, 2002.

35. http://www.magmasoft.com/, Aug. 9, 2002.

36. http://www.quantum-online.com/QDC-A2Large.htm, Aug. 9, 2002.

37. http://www.oit.doe.gov/metalcast/factsheets/vistool.pdf, Aug. 9, 2002.

38. http://www.ues-software.com/subhtm/products/procast.htm, Aug. 9, 2002.

SECTION II:

GRAPHICAL USER INTERFACE FOR COOLING LINE FUNCTIONS AND SURFACE RENDERING[4]

[4] Based on the work of Xiaorui Chen , "Graphical User Interface For Cooling Line Functions And Surface Rendering," MS Thesis, Mechanical Engineering, The Ohio State University, 2003.

1. INTRODUCTION

Die casting is a precision, high volume production process in which molten metal is forced, at high pressure and velocity, into a reusable die that has a cavity in the desired shape of the casting part. Upon solidification, facilitated by the cooling system in which the coolant flows through the die sections, die clamping is relaxed and the casting part is ejected.

Because of its capability to produce parts with thin walled, complex geometry, close tolerances and high production rates, die casting has been applied in many industries, especially in the automobile industry. However, die casting is such a complex process involving many factors, including parting line location, cooling system design, gate and runner design, vent location, and die layout design as well as part design, that die casting is unnecessarily expensive and the scrap rate is often high because of its incompatibility with the part design.

To meet the high demand of today's industry in terms of short time-to-market, high quality and low cost, Concurrent Engineering (CE) has emerged as a new design philosophy. In Concurrent Engineering, both the actual and simulated processes of designing and developing a product are performed simultaneously. All aspects, including final cost, manufacturability, safety, packaging, and recyclability, of a product are considered concurrently, with the design modified as necessary to make sure that the product is useful at all stages of its lifecycle. The practice of considering manufacturing needs at design time is known as Design For Manufacturing (DFM).

DFM is one of the most important aspects of CE. The primary objective of DFM is to produce a design at a competitive cost by improving its manufacturability without affecting its functional and performance objectives. Problems that might arise during manufacturing are anticipated and dealt with at design stage, where is the least expensive changes in a product's lifecycle.

Manufacturability evaluation plays an important role in DFM. Currently, two types of manufacturability evaluation techniques may be used to assist in the design of quality die casting parts. The first one is knowledge-based system that is limited in scope since they cannot reliably handle the range of complex part geometries found in die casting. The second one is based on finite element, finite difference, or boundary element method. These tools need very long run time, sometimes even days. Furthermore, an understanding of the assumptions behind the model is critical to correctly interpret the results.

To remedy the above problems, Lu and Miller proposed a qualitative reasoning approach based on a volumetric part representation. In contrast to assessment using conventional tools, this approach provides a more consistent and robust evaluation of the part geometry. It identifies the underlying geometric characteristics, which actually affect part quality or increase manufacturing difficulties, using volume-based geometric reasoning. The result can be easily interpreted by designers via volume visualization techniques.

By using this approach, a platform-independent die-castability evaluation tool, CastView, has been developed. It can be used by designers with little or no experience with process

simulation tools, quickly answer die-castability questions at the early design stage, and be easily integrated with virtually any CAD system.

1.1 Die Casting Dies

Die casting dies are made of alloy tool steels in at least two sections, called cover die half and ejector die half. The two halves separate at the parting surface, to allow for the removal of the casting part. The cover die half is mounted on the side toward the molten metal injection system, while the ejector die half is attached on the moveable platen of the machine.

The cover die half usually contains the sprue or shot hole through which molten metal enters the die. The ejector half is designed to contain the ejector mechanism and, in most cases, the runners (passage ways) and gates (inlets) that route molten metal to the cavity (or cavities) of the die. The pair of opposite directions along which the two die halves separate are the die opening directions.

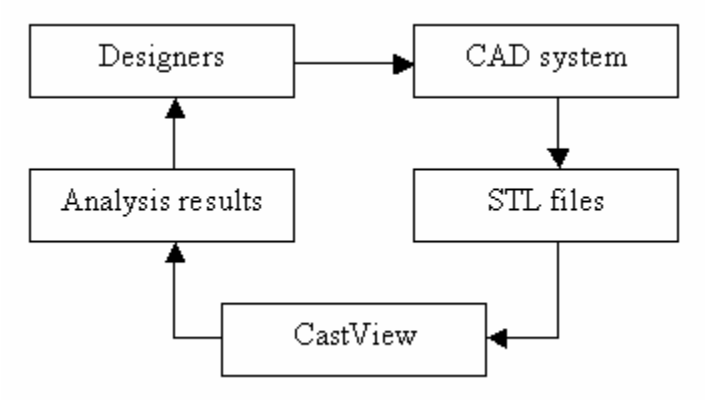
Die cores, either fixed or movable, may be placed in either die half. These allow holes to be cast in various directions. Various inserts may be pre-positioned in the dies to become integral casting features. The dies also have internal cooling conduits through which liquid is circulated.

The die cavity into which the casting part is formed is machined into both halves of the die block or into inserts that are secured in the die blocks. After the metal solidifies and the die opens, ejector pins push the casting from the ejector die half. Before closing for

the next injection cycle, the dies are subjected to an air blast to be cleaned, and then given a lubricating spray.

1.4 Research Objective

Figure 1.1 shows the current integrated system of CAD and CastView. The link between a CAD system and CastView is the STL file, which is generated by the CAD system and imported into CastView. Upon an analysis request, the voxel model is created from the STL model and the analysis is then performed based on the voxel model [16]. According to the analysis result, the designers may go back to modify the original design until the desirable result is obtained.



Section II-Figure 1.1: The integrated system of CAD and CastView

One goal of this research is to design the Graphical User Interface for the definition of the die configuration and the cooling lines, and implement related functions. Die configuration and cooling line locations play an important role in the temperature distribution in the die casting dies and part. Incorrect die configuration and cooling lines

locations may result in problems related with thermal distribution and filling pattern, such as cold shuts and non-uniform shrinkage.

One task of the thermal analysis is to assist the designers in specifying die configuration and cooling line placements. Since CastView is an interactive system, the die configuration and cooling lines can be designed on the STL model, and may be modified if the results of the thermal analysis are undesirable. Thus, the problem that may arise in manufacturing process is eliminated at the design stage, the time-to-market and the cost are greatly reduced, and the product quality is significantly improved.

Another goal in this research is to map the wall thickness obtained during the thin section analysis back to the part surface so that a comparison of all the wall thickness values may be performed, and wall thickness data output to the quantitative evaluation system.

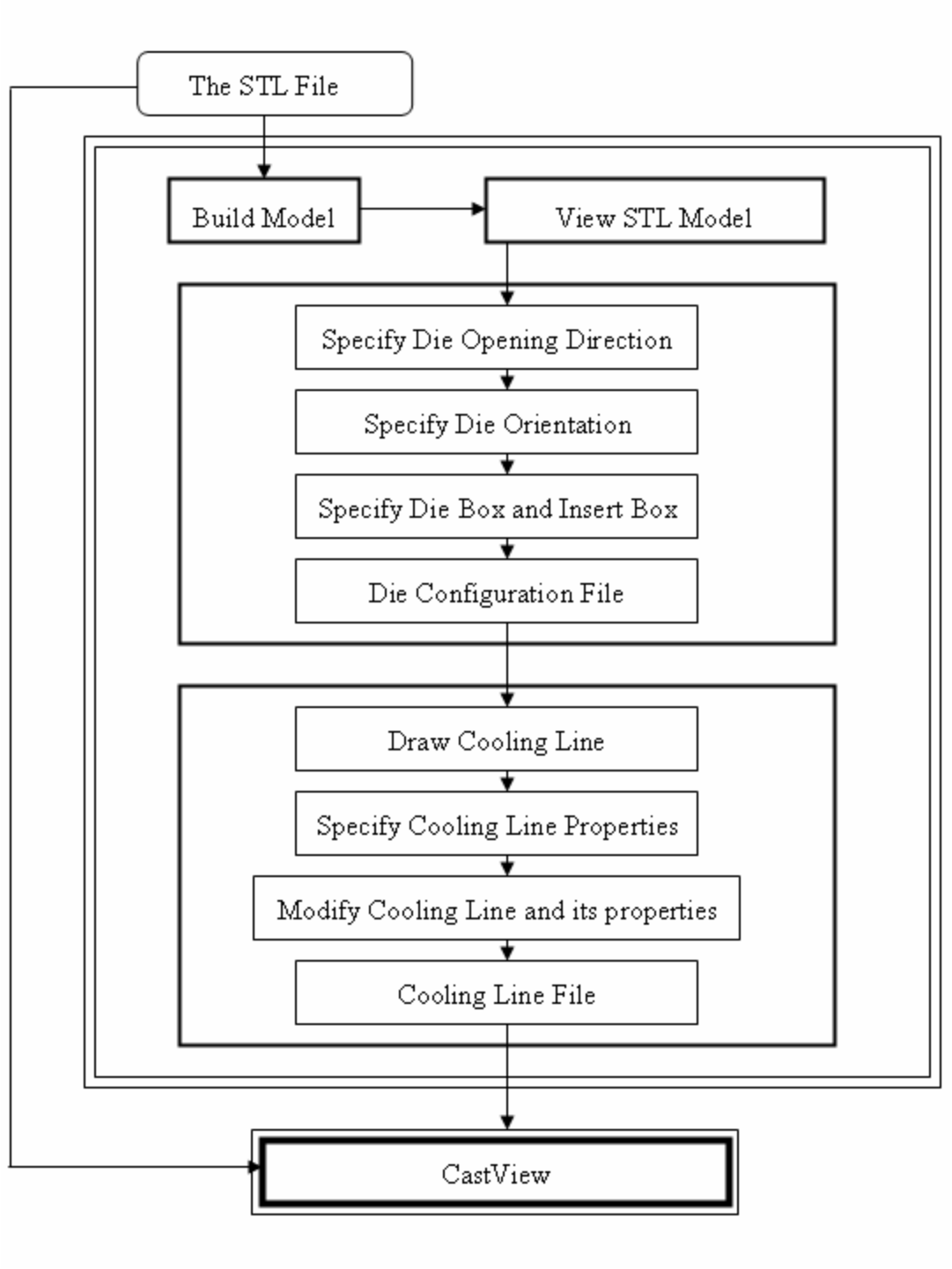
2. RESEARCH APPROACH

2.1 System Structure

The system structure is shown in Figure 2.1. There are four main modules contained in the system: model building, view module, die configuration module, and cooling line sketching module.

The model of the part is built right after the STL file is imported. Once it is displayed on the screen, users can work with it interactively, such as switching display modes, rotation, translation, and zoom.

Before cooling lines are constructed, the die configuration must be defined. Die configuration includes information about die opening direction, die orientation, die box, and insert box. In CastView, the die box refers to the die shoe, the insert box is actually the die in which the casting is formed, and the die opening direction points toward the cover side of the die while the die orientation can be a direction pointing to the operator or upward. The die configuration is calculated using part information and the user input information, and can be saved in the die configuration file.



Section II-Figure 2.1: The System Structure

After the die configuration is defined, users can draw or modify the cooling lines using three methods: directly input coordinates of the cooling line nodes, orthogonal sketch and free sketch. Once the cooling lines are finished, they can be saved in the cooling line file. The die configuration file and the cooling line file are used as an input for the thermal analysis by the system.

2.1.1 Model Building

The model is built using the information from the imported STL file. Then the rendering algorithm is applied to display the model on the screen. In CastView, the rendering is implemented using OpenGL.

2.1.2 View Module

After the model is displayed on the screen, users can manipulate it interactively. The following functions are available.

- Display the coordinate axes
- Display the STL model
- Display the die box and the insert box
- Display the die opening direction and the die orientation vectors
- Display the parting surface
- Display the specified cooling line(s) and/or sketch planes
- Implement view operations such as rotation, translation, and zoom.
- Switch between different display modes

2.1.3 Die Configuration Module

This module provides the following functions to help the user to define or modify the die configuration: die opening direction, die orientation, die box, and insert box.

- Enable the user to define or modify the die opening direction
- Enable the user to define or modify the die orientation
- Enable the user to define or modify the die box and the insert box

Die configuration gives a constraint for the cooling lines. That is, the cooling lines must lie inside the die box.

2.1.4 Cooling Line Sketching Module

The following functions are implemented in the cooling line sketching module to help the user to define the cooling line placements and properties.

- Enable the user to sketch cooling lines using three sketching methods
- Allows the user to define the properties for the specified cooling line
- Allows the user to modify a previously specified cooling line
- Allows the user to delete the specified cooling line
- Allows the user to save cooling line information in the cooling line file
- Allows the uses to load the cooling line file

Cooling lines are forced to fall inside the die box. Cooling lines and die configuration data can be written into their corresponding files, which provide data for thermal analysis.

2.2 STL Model vs. Voxel Model

The STL ("Stereo Lithography") model approximates the surfaces of a solid model with a set of triangles. The surface is tessellated or broken down logically into a series of small triangles that are called facets, each of which is described by a normal and three points representing the vertices of the triangle.

The .STL file format has become the Rapid Prototyping industry's defacto standard data transmission. Almost all of today's CAD systems are capable to produce a .STL file for a solid model, such as AutoCAD, ProE, SolidEdge, SolidWorks, Unigraphics, and so on.

The .STL file is used as the original input to create the model that is analyzed by CastView. The STL model generated can be manipulated to rotate, zoom, translate, and even switch between different display modes, such as wire frame, no hidden line, and shaded mode. The components of the die casting tool, such as cooling lines, gates, overflows, etc, are all designed based on the STL model in CastView.

Voxel-based representation is another way to represent solid objects. In voxel-based representation, the solid is decomposed into identical cells, which are called voxels (volume element), arranged in a fixed, regular grid. The most common cell type is the cube. The solid objects can then be encoded by a unique and unambiguous list of occupied cells, which mean that they are inside the solid objects.

Since each voxel is represented by the coordinates of a single point, such as the centroid of that cell, and a specific spatial scanning order is imposed, it is easy to control the behavior of each voxel during reasoning. And since the voxels are identical, the

reasoning algorithms will not be dependent on the geometric complexity of the casting part. This is the greatest benefit of representing the complex casting part by a voxel model.

During the thick and thin section analysis, the STL model is first transformed to the voxel model, and the data processing techniques, such as distance transformation, thinning, etc, are then be used to assist the die castability evaluation. Wall thickness is then mapped back to the voxel model surface after the thinning process.

2.3 OpenGL

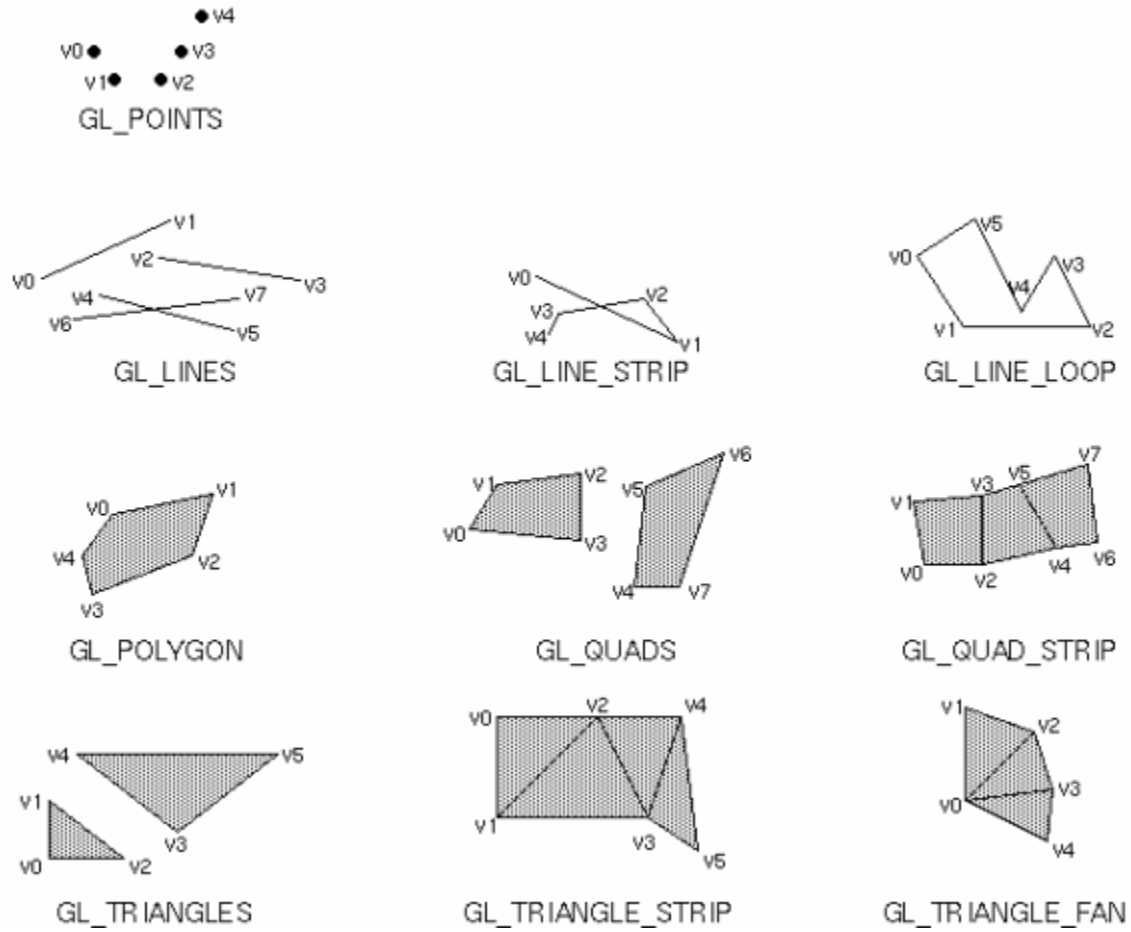
OpenGL is a software interface to graphics hardware. It consists of about 120 different commands that allow the user to specify the objects and operations needed for producing interactive 3-dimensional applications. OpenGL is fully cross-platform and it is very easy to move applications from one platform to another. Today, OpenGL has become the most widely used 3D API in Win9x/NT/2k/XP systems for both professional and consumer graphics applications.

By using OpenGL, CastView software implements the view and manipulation functions for both the STL model and the voxel model, including viewing in different display modes, translation, rotation and zoom.

2.3.1 Displaying Geometric Primitives

Using OpenGL, all kinds of objects are constructed from a small number of geometric primitive items, which are eventually described in terms of their vertices - coordinates that define the points themselves, the endpoints of line segments, or the corners of

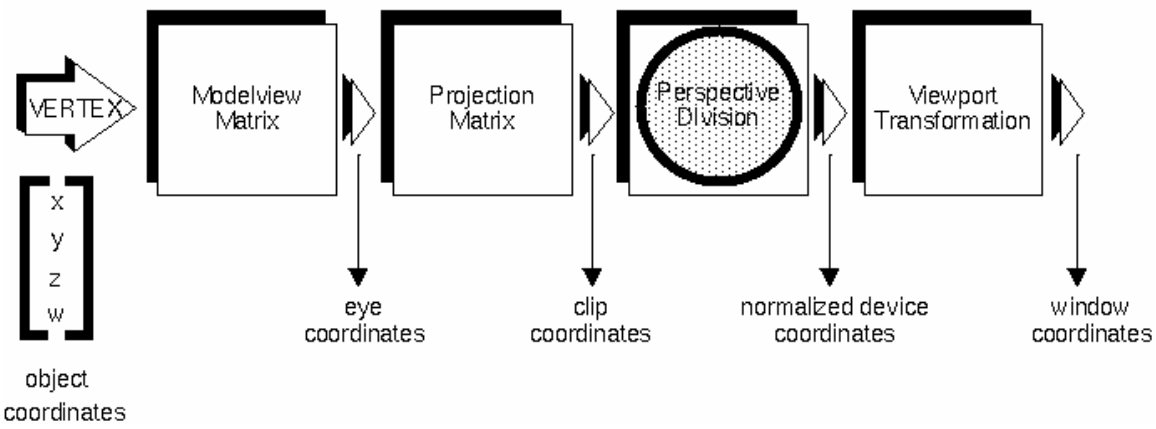
polygons. Figure 2.2 shows examples of all the geometric primitives used in OpenGL. (v0, v1, v2, ... , vn-1) are described using OpenGL commands sequentially that are written between glBegin() and glEnd(), which represents the beginning and the end of drawing one kind of geometric primitives described in Figure 2.2.



Section II-Figure 2.2: Geometric primitive types [20]

2.3.2 Pipeline of the Transformation

The STL model is defined in object coordinate system in CastView. In order to view it on the screen, a series of computer operations are used for each vertex composing the STL model to convert the 3-dimensional object coordinates to screen coordinates on the screen. Figure 2.3 shows the stages of vertex transformation.



Section II-Figure 2.3: Stages of vertex transformation [20]

First, the modeling transformation matrix transforms the vertex from the object coordinate system to the world coordinate system. Then the viewing transformation matrix transforms the vertex from world space to eye space. The modeling and viewing transformations are combined to form the modelview matrix, which is applied to the incoming object coordinates to yield eye coordinates. The modelview matrix involves mainly translation, rotation and scaling, among which the rotation transformation can be expressed using concept of quaternion [22].

A quaternion is defined using four floating point values (x, y, z, w), and extends the concept of 3-dimensional rotation to 4-dimensional rotation. This avoids the problem of "gimbal-lock", which may arise when using the Euler angles for a rotation, and allows for the implementation of smooth and continuous rotation. Quaternions allow for rotating an object through an arbitrary rotation axis and angle.

A quaternion can be calculated from another quaternion, or be converted from a rotation matrix, a rotation axis and angle, spherical rotation angles, or Euler rotation angles; vice versa. We can also get the conjugate, the inverse, and the magnitude of a quaternion from its definition. The operations of normalization, multiplication, linear and cubic interpolation are defined for quaternions as well [11].

Second, the projection matrix that defines a viewing volume is applied to yield clip coordinates. Scenes outside the volume are discarded when the final scene is drawn on the screen. Then the perspective division is performed by dividing the homogenous coordinate values by *w*, to produce normalized device coordinates.

Finally, the viewport transformation applies so that the normalized device coordinates are converted to window coordinates, which finally determine the object color of each pixel on the screen. Since all the transformations are performed on *z* coordinates as well, the *z* values of the window coordinates may be used for the visibility detection.

2.3.3 Display Modes

The objects can be viewed in different display modes – wire frame mode, no hidden line mode and shaded mode, using OpenGL. The wire frame mode enables the user to see

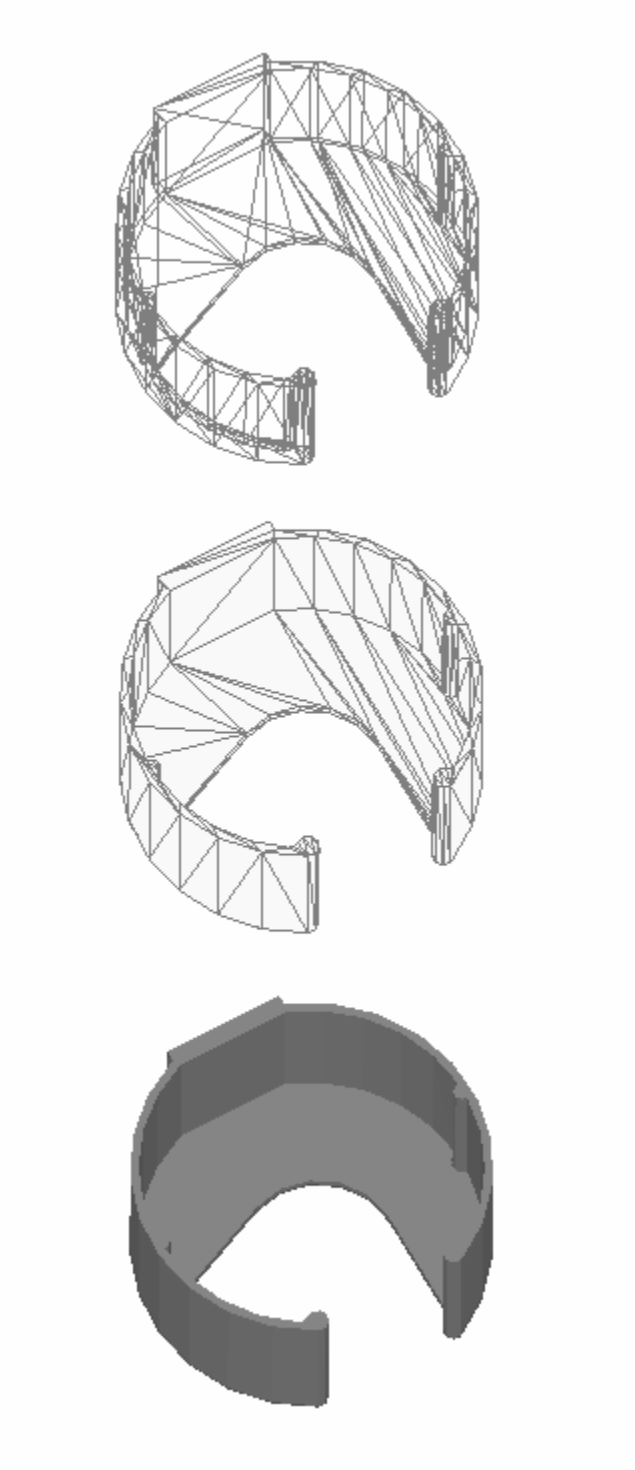
each primitive clearly and select the triangle vertices and edges correctly, while the no hidden line mode and shaded mode makes the objects more realistic. Figure 2.4 shows an object in wire frame mode, no hidden line mode and shaded mode respectively.

2.3.4 Colors and Blending

OpenGL keeps a current color (in RGBA mode) or a current color index (in color-index mode). Unless a more complicated coloring model, such as lighting or texture mapping, is used, each object is drawn using the current color (or color index).

When blending is enabled, the alpha value, which represents the opacity, is used to combine the color value of the fragment being processed with that of the pixel already stored in the frame buffer. Blending occurs after the scene has been rasterized and converted to fragments, but just before the final pixels are drawn in the frame buffer.

Without blending, each new fragment overwrites any existing color values in the frame buffer, as though the fragment is opaque, that is, the alpha value is 1.0. With blending, how much of the existing color value should be combined with the new fragment's value can be controlled. Thus, the alpha blending can be used to create a translucent fragment, one that lets some of the previously stored color value "show through."



Section II-Figure 2.4: An object shown in wire frame, no hidden line, and shaded model

2.4 Summary

This chapter first explained the system structure, which includes four main modules: model building, view module, die configuration module, and cooling line sketching module. Each module has a specific function and can be combined with other modules.

Then the STL model and the voxel model were discussed. The components of the die casting tool are designed by the user based on the STL model, while the analyses, including thick section analysis, thin section analysis, filling analysis, and the wall thickness analysis, are performed based on the voxel model.

CastView software implements the view and manipulation functions for both the STL model and the voxel model using OpenGL, which has been a widely used software interface to graphics hardware. The geometric primitives, the pipeline of transformations, the display modes, colors and blending functions used by OpenGL were talked about in section 2.3.

Since the model building module and the view module have been built in CastView, the die configuration module will be discussed in next chapter, which includes the die opening direction construction, the die orientation construction, the die box and the insert box specification. A machine coordinate system will be constructed during the definition of the die configuration.

3. DIE CONFIGURATION FUNCTIONS

3.1 Defining Die Opening Direction

Die Opening Direction is one of the important specifications of die design in near-net-shape manufacturing. Current research shows that possible die opening directions can be derived directly from the geometric information of the part. Desirable die opening direction allows the part to be ejected from the die successfully. In CastView, the die opening direction points toward the cover side of the die. The direction of the vector determines which side of the parting surface the gate lies on.

Users can define the die opening direction using four methods.

1. Pick the normal of any polygon on the STL model surface to be the die opening direction

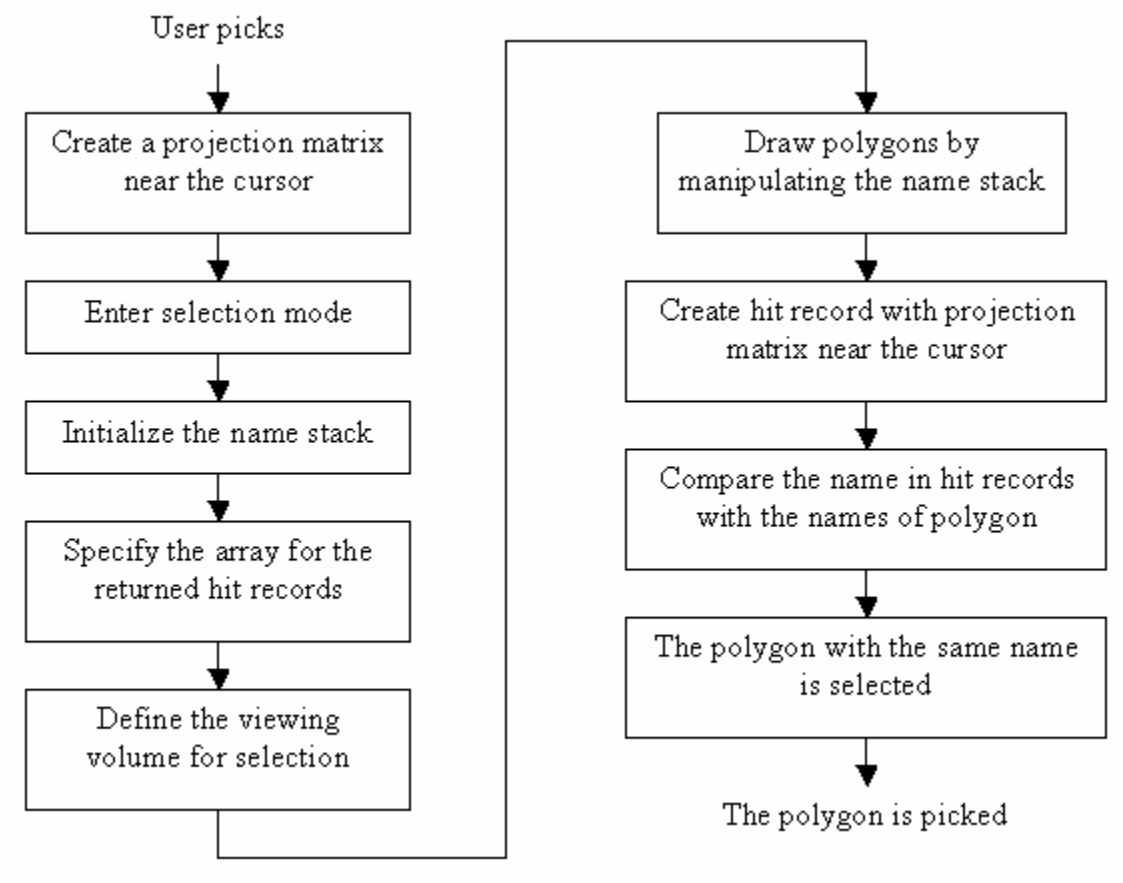
2. Pick any two points on the STL model surface. The die opening direction goes from the second picked point to the first picked point.

3. Pick a polygon edge on the STL model surface. The die opening direction is the edge direction, which goes from the rear vertex to the front vertex.

4. Pick the opposite direction of the current die opening direction by flipping it using right mouse button clicking.

Since the defining of the die opening direction involves picking the geometric entities (points, edges, and triangles) and picking polygon functions are used in the first three

methods, the reasoning pipeline for picking polygons is discussed here. It is achieved by using a reasoning algorithm provided by OpenGL. The input element of the pipeline is the user click on the screen. The output of the pipeline is the polygon index on the polygon list. The pipeline for picking polygons is illustrated in Figure 3.1.



Section II-Figure 3.1: The pipeline of picking a polygon [23]

3.1.1 Defining the Die Opening Direction by the Normal of a Polygon

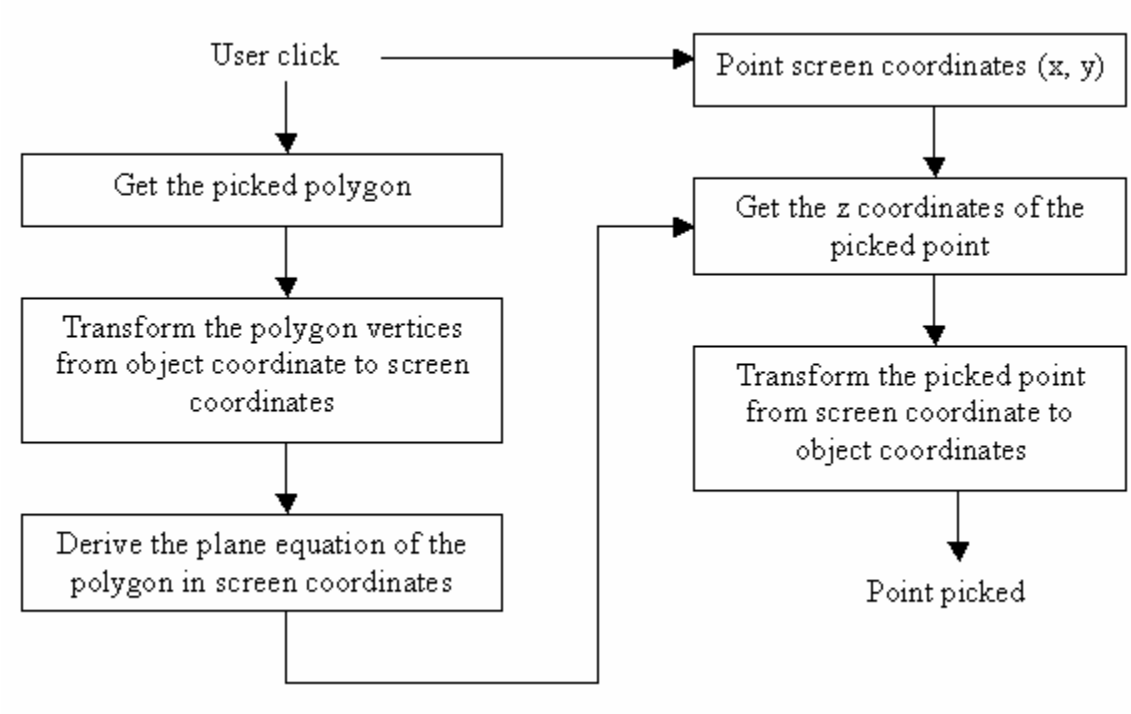
Since the normal of the polygon is built into the STL model when the STL file is imported, the normal can be retrieved right after the polygon is selected through the

method described in previous section. Defining the die opening direction by the normal of a polygon is the simplest way. Users just need to double click on the desired polygon using left mouse button. The latter result of the double click always replaces the previous result.

3.1.2 Defining the Die Opening Direction by Two Points

The die opening direction can be defined by two arbitrary points. In Castview, these points must be on the STL model surface. However, it is sufficient for the user to define an arbitrary die opening direction. Users can pick two points using left mouse button double click. Once the second point is selected, the die open direction is defined as a vector pointing from the second point to the first point.

After the user picks a point on the screen, the corresponding polygon can be easily found by the method described in section 3.1, if there is any. Since we have had the object coordinates of each vertex of the polygon, and we can get the transformation matrix using OpenGL functions, we can calculate the screen coordinates of each vertex of the selected polygon, from which the plane equation $Ax + By + Cz + D = 0$ can be derived again in screen coordinates. As long as the picked point is on this plane, its screen coordinate z can be easily achieved with its screen coordinates (x , y) already known. Then the picked point is transformed from screen coordinates to object coordinates using the inverse transformation matrix mentioned above. Figure 3.2 illustrates how to achieve the object coordinates of the picked points.



Section II-Figure 3.2: Pick a point on the STL model

If the user wants to correct the die opening direction by this method, the only thing required is to select two points again to construct the new die opening direction. The previous process is repeated.

3.1.3 Defining the Die Opening Direction by the Edge of a Polygon

After the user double clicks on the screen, the corresponding polygon is selected using the method discussed in section 3.1. Then the object coordinates of the picked point are calculated using the same method as described in section 3.1.2. With the object coordinates of the picked point and each vertex of the polygon known, the distance between the picked point and the edge line is calculated. The smallest distance is selected and the corresponding edge line is copied as the die opening direction. The die opening

direction points from the rear point to the front point of the edge line, in the saved sequence. The later result will replace the previous result if the user picks a new point.

In order to get the desired edge of the polygon as the die opening direction, the user is recommended to use the wire frame or no hidden line display mode instead of the shaded display mode.

3.1.4 Reversing the Die Opening Direction

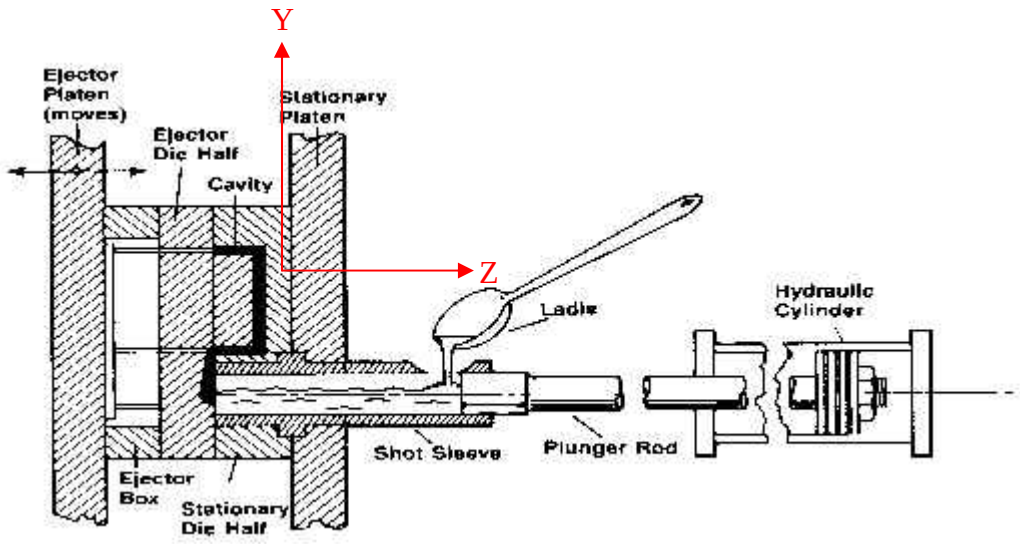
After the die opening direction is defined by one of the first three methods, users can flip it to the opposite direction by right mouse click. This provides the user a convenient way to get an opposite die opening direction quickly.

3.2 Defining the Die Orientation

Only the die opening direction is not enough to determine the placement of the die box and the insert box relative to the casting. The die orientation provides the other information to place the die box and the insert box.

Since the surfaces of the die box and the insert box are perpendicular or parallel to either the die opening direction vector or the die orientation vector, a *machine coordinate system* is defined using the die opening direction vector as the *z machine axis* vector and the die orientation vector as the *x or y machine axis* vector. Then the right hand rule is applied to get the other *machine axis* vector. The coordinates of the casting part center point are kept the same in both the *machine coordinate system* and the *object coordinate system*, which is constructed when the STL model is built. In this way, the surfaces of the die box and the insert box are perpendicular to one of the *machine axes*, and the user can

have a clear view about the relative position of the die box and the insert box and the casting part. Moreover, it is much easier for the user to place the cooling lines using the *machine coordinate system*. The following thermal analysis, which needs to separate part voxels from die voxels, also becomes much easier in the *machine coordinate system*.



Section II-Figure 3.3: Machine coordinate system construction (NADCA website)

Figure 3.3 illustrates how the *machine coordinate system* is constructed. The *z machine axis* is in the same direction as the die opening direction, which points toward the cover die from the ejector die. The *y machine axis* points upwards and the *x machine axis* points inwards, which are defined by the die orientation. The user can define either *x or y machine axis* direction to construct the *machine axis system*. The die orientation vector can be defined using five methods.

1) Pick the normal of any polygon on the STL model surface to be the die orientation vector

2) Pick any two points on the STL model surface. The die orientation vector goes from the second picked point to the first picked point.

3) Pick an edge of a polygon on the STL model. The die orientation vector is the edge direction, which goes from the rear vertex to the front vertex.

4) Pick any point on the *datum plane*, which is perpendicular to the die opening direction and goes through the casting part center point. The die orientation vector points from the casting center to the picked point on the *datum plane*.

5) Pick the opposite direction of the current die orientation vector by flipping it using right mouse button clicking.

3.2.1 Defining the Die Orientation by the Normal of a Polygon

See section 3.1.1.

3.2.2 Defining the Die Orientation by Two Points

See section 3.1.2.

3.2.3 Defining the Die Orientation by the Edge of a Polygon

See section 3.1.3.

3.2.4 Defining the Die Orientation by Point on the Datum Plane

The *Datum Plane* is constructed by the die opening direction, which is the *z machine axis* vector, and the casting part center point. The *datum plane* equation is described as:

$$Ax + By + Cz + D = 0$$

where *(A,B,C)* is the normal vector to the plane, that is, the die opening direction.

$$D = -(Ax_0 + By_0 + Cz_0)$$

where $(x_{0,} y_{0,} z_0)$ is the casting part center point.

After the *datum plane* is established, the user can pick any point on the *datum plane*. The picked point is in 2D screen coordinates and can be transformed to 3D object coordinates using the same method described in section 3.1.2.

The die orientation vector points from the casting center point to the picked point. If the user picks another point on the *datum plane*, the die orientation vector will point from the casting part center point to the newly picked point. The previous result is replaced with the new one.

3.2.5 Reversing the Die Orientation

See section 3.1.4.

3.3 Transformation Matrix to Machine Coordinate System

The *machine coordinate system* is constructed once the die opening direction and the die orientation are defined, as is described in the previous section. Since the information of the STL model is stored in *object coordinates*, the transformation matrix is needed to transform the information from the *object coordinates* to the *machine coordinates* when the calculations are to be done in the *machine coordinate system*.

The OpenGL convention is used here to get the transformation matrix between different coordinate systems, that is, the coordinate system frame is transformed instead of the

object. In CastView, the *machine coordinate system* is transformed to coincide with the *object coordinate system* while the coordinates of the casting part center point are unchanged. This process involves rotation and translation.

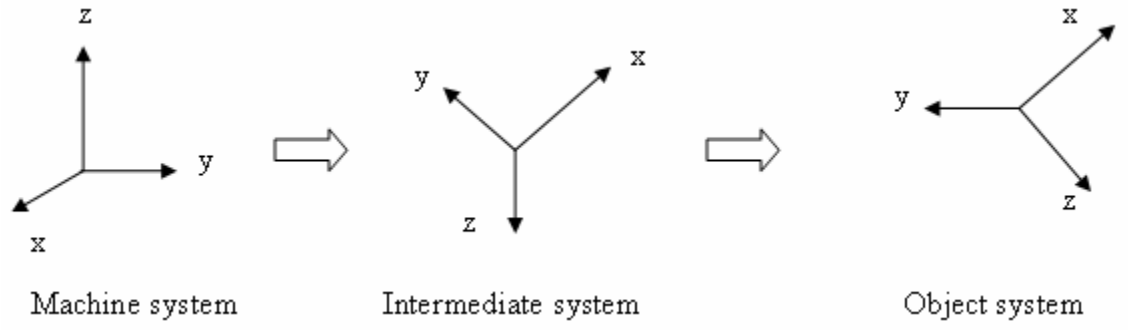
Since the coordinates of the casting part center point are unchanged, the rotation must be around the casting part center point. Noticing that the rotation in OpenGL is around the coordinate system origin, the coordinate system origin must be translated to the casting part center point first. After the rotation, the coordinate system origin is translated back to its original position.

First, the *machine coordinate system* is translated so that *machine coordinate system origin* coincides with the casting part center point. The translation matrix is:

$$M_1 = \begin{bmatrix} 1 & 0 & 0 & x0 \\ 0 & 1 & 0 & y0 \\ 0 & 0 & 1 & z0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where $(x0, y0, z0)$ are the coordinates of the casting part center point.

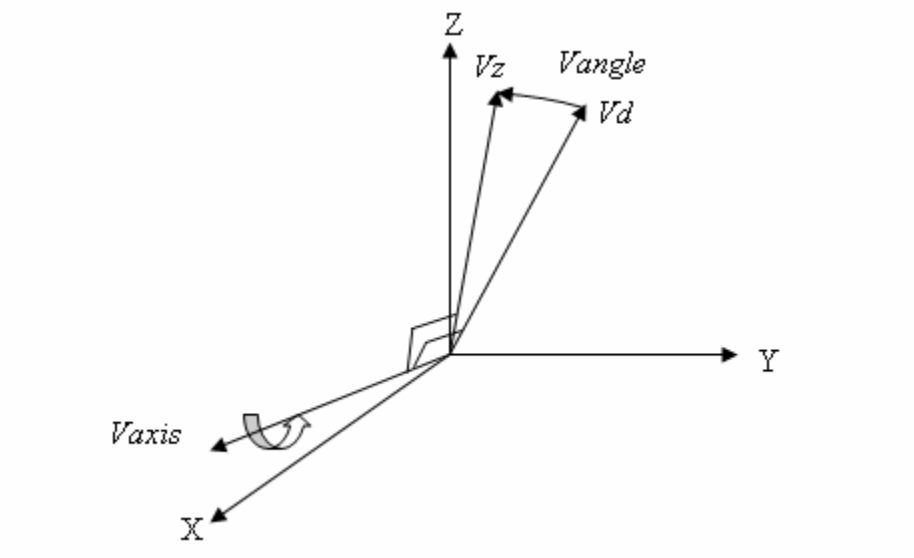
Second, the *machine coordinate system* is rotated twice so that its orientation is the same as that of the *object coordinate* system. This process is illustrated in Figure 3.4.



Section II-Figure 3.4: Transform the machine coordinate system to object coordinate system

The first rotation is made to map the *x machine axis* onto the *x object coordinate axis*. An *intermediate coordinate system*, whose *x intermediate axis* is the same direction as the *object coordinate system*, is generated after the first rotation. The first rotation matrix is *M2*. The second rotation is made to map the *z intermediate axis* onto the *z object coordinate axis*. The second rotation matrix is *M3*.

The rotation matrix can be generated using the concept of *quaternion*, which is described in section 2.3.2, to map one unit vector *Vd* onto another unit vector *Vz*. There are two elements for a quaternion structure: rotation axis and rotation angle, which is shown in Figure 3.5.



Section II-Figure 3.5: Mapping a vector *Vd* onto another vector *Vz*

The rotation axis *Vaxis* is calculated by taking the cross product between the unit vectors, *Vd* and *Vz*. That is,

$$Vaxis = Vd \times Vz$$

The rotation angle *Vangle* is calculated by taking the dot product between *Vd* and *Vz*. That is,

$$Vangle = \cos^{-1}(Vd \bullet Vz)$$

From section 2.3.2, we know that the quaternion *Q(X, Y, Z, W)* can be calculated from (*Vaxis, Vangle)* and a rotation matrix can then be converted from the quaternion *Q*.

What is noticeable is that the cross product returns (0,0,0) if the unit vector *Vd* and *Vz* are collinear. In our problem, this will occur if the *z intermediate axis* is (0,0,1) or (0,0,-1), or the *x machine axis* is (1,0,0) or (-1,0,0). When the *z intermediate axis* is (0,0,1) or the *x*

machine axis is (1,0,0), we may set an identity matrix to the rotation matrix M since they have already aligned with the *z or x object coordinate axis*. If the *z intermediate axis* is (0,0,-1), we simply set the rotation axis as (1,0,0) and the rotation angle as 180^{o}. If the *x machine axis* is (0,0,-1), we set the rotation axis as (0,0,1) and the rotation angle as 180^{o}.

After the rotation, the *machine coordinate system* is translated back to its *machine coordinate system origin*. The translation matrix is:

$$M_4 = \begin{bmatrix} 1 & 0 & 0 & -x0 \\ 0 & 1 & 0 & -y0 \\ 0 & 0 & 1 & -z0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The total transformation matrix is described as:

$$M = M_1 * M_2 * M_3 * M_4$$

If an arbitrary point on the STL model is (x, y, z) before the transformation, the point becomes (x', y', z') after the transformation. Then the relationship between the points is:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x''/w \\ y''/w \\ z''/w \end{pmatrix}, \qquad \begin{pmatrix} x'' \\ y'' \\ z'' \\ w \end{pmatrix} = M * \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

After the transformation, the information of the STL model is under *machine coordinate system*, with the coordinates of casting part center point unchanged.

3.4 Defining the Die Box and the Insert Box

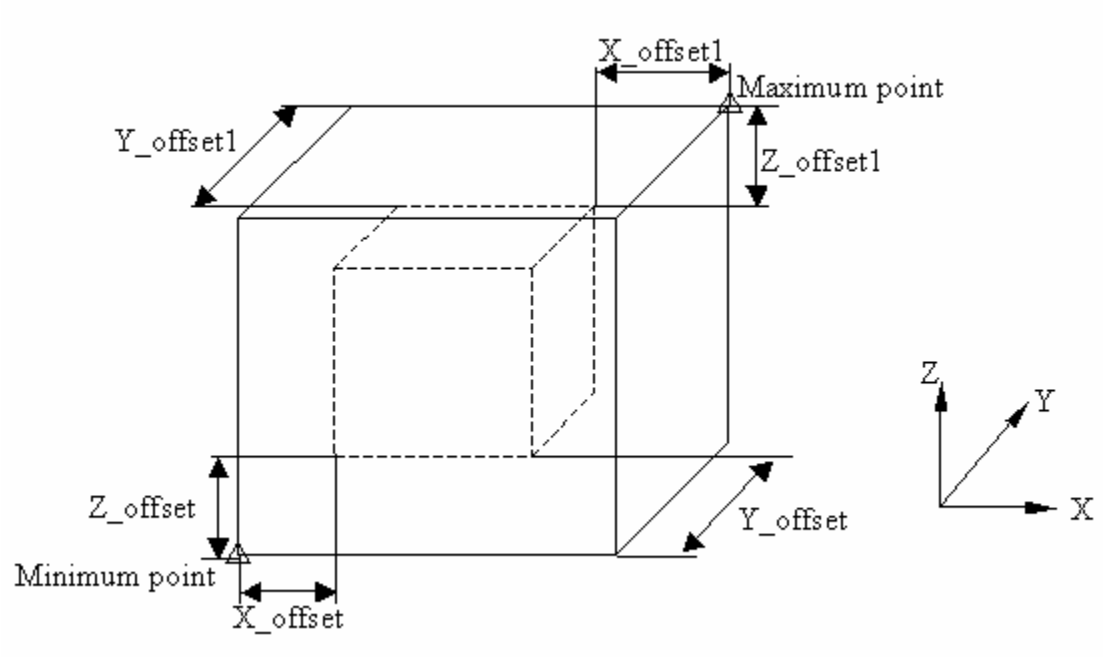
Die box and insert box are the most important elements in the die casting process. The inserts must be held in position by the die when the metal is ejected. Die casting dies consist of at least two sections which meet at the parting surface. In CastView, the die box refers to the die shoe while the insert box is actually the die in which the casting is formed.

In the system, die box and insert box are represented by two boxes. The edges of the boxes are parallel to one of the *machine axes*. Therefore, we can define the box by two points: the minimum point and the maximum point. The minimum point has the smallest coordinates and the maximum has the largest coordinates.

3.4.1 Defining the Insert Box

The size of the insert box is often decided relative to the casting part to be produced. The calculation of the size of the insert box is illustrated in Figure 3.6.

The inner box is the bounding box of the casting part, which is stored in the part information. The outer box is the insert box. The minimum point and the maximum point of the insert box are calculated using the bounding box and the 6 offset values. All the edges of the boxes are parallel to one of the *machine axes*.



Section II-Figure 3.6: Calculate the size of the insert box

3.4.2 Defining the Die Box

The size of the die box is calculated relatively to the insert box. The calculation of the size of the die box is similar to that of the insert box, which is also shown in Figure 3.6. The inner box is the insert box, which has been achieved in the previous section. The outer box is the die box. The minimum point and the maximum point of the die box are calculated using the insert box and the 6 offset values. All the edges of the boxes are parallel to one of the *machine axes*.

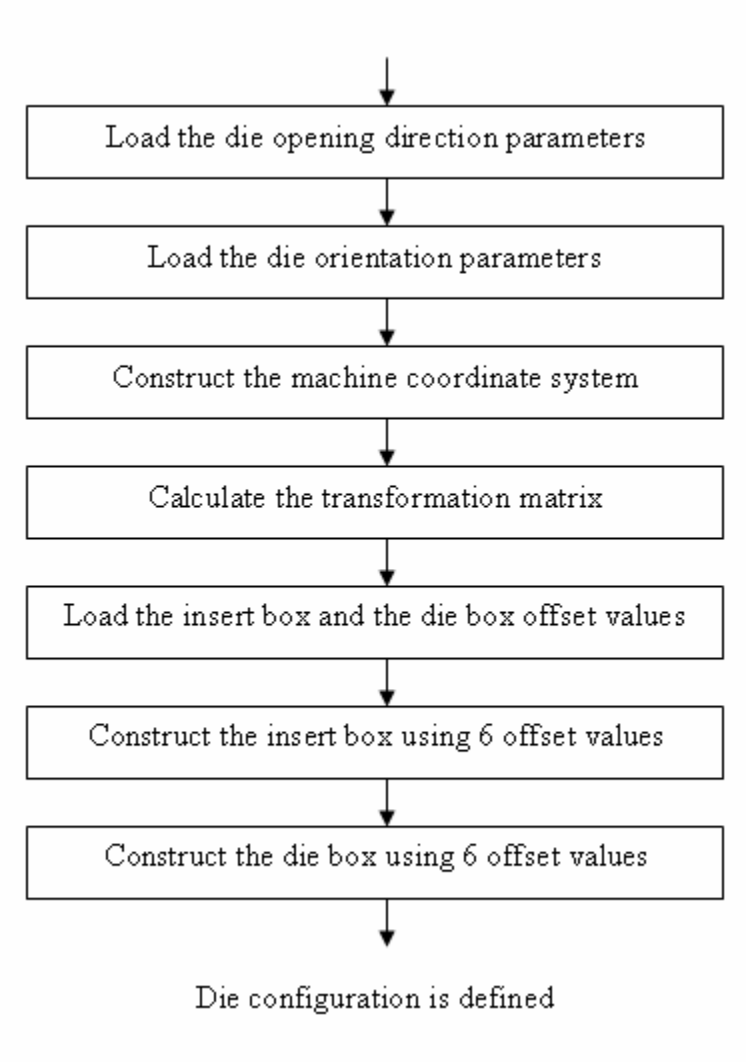
3.5 Modifying Die Configuration

A die configuration can be modified at any time. Users can modify the die opening direction, the die orientation, and the size of the die box and the insert box by repeating the same operations as before.

But since the die configuration is defined step by step based on the sequence of defining the die opening direction first, then the die orientation, and finally the die box and the insert box, the user is required to finish the subsequent definition once they modify the die configuration. For example, if the die orientation is modified, the die box and the insert box must be defined again, while the die opening direction is not necessarily redefined.

3.6 Saving and Loading Die Configuration

Saving the die configuration includes saving the die opening direction, the die orientation, the die box and the insert box. The information is stored in files so that we can load the die configuration to review or modify. After the die configuration is loaded, the result should be the same as the die configuration just defined. Since the die configuration is defined in the order of defining the die opening direction first, then the die orientation, and finally the die box and the insert box, we save the die configuration in the same sequence in order to load the die configuration conveniently. The file format is shown in Appendix A. Figure 3.7 shows the approach of loading a die configuration.



Section II-Figure 3.7: Load die configuration

3.7 Summary

This chapter discussed the die configuration, including the die opening direction, the die orientation, the die box and the insert box. A *machine coordinate system* is constructed after the die opening direction and the die orientation are defined. With the *machine coordinate system*, it is very easy for the user to define the cooling lines, which will be discussed in Chapter 4.

4. COOLING LINE FUNCTIONS

A cooling line is a tube or passage through which fluid is forced to circulate to cool a casting die. The cooling system is one of the subsystems related with the die casting process technology. The location of the cooling lines is a very important factor for an ideal die casting die. Cooling prevents the die from overheating which may cause premature die failure and helps the die to keep a uniform temperature distribution and shrinkage to obtain favorable surface finish and longer die life.

However, cooling channel placement is often designed by guesswork or by past experience. It is very expensive and time consuming to modify the improperly placed cooling channels after the die has been built. In CastView, we enable the user to place the cooling lines before the thermal analysis. Redesign is also made possible interactively. The system provides three methods to build a cooling line: directly input coordinates of the cooling line nodes, orthogonal sketch and free sketch. The system also allows the user to modify a defined cooling line by changing its node coordinates, orthogonal editing or graphical editing.

Since a cooling line is constrained inside the die box, a die box must be defined first before the cooling lines are constructed.

4.1 Constructing New Cooling Lines

A cooling line is composed of a sequence of points which are connected one by one in the sequence they have been picked.

4.1.1 Input Coordinates to Construct Cooling Lines

This is the easiest way to construct a cooling line. In order to let the user to locate the nodes well, the *relative origin (0,0,0)* is defined. Its location is set at the corner of the die box with the minimum coordinates of all the points on the die box in the *machine coordinate system.* The *relative coordinates* of each node, which composes the cooling line, are displayed in a table. The user can add, insert, delete, or change the *relative coordinates* of the node.

When the user confirms to construct the cooling line by clicking "OK" button on the dialog, the *relative coordinates* of the cooling line nodes are then translated to the *machine coordinates*.

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix}$$

where, (x_0, y_0, z_0) is the *machines coordinates* of the *relative origin*, (x, y, z) is the *relative coordinates* of each cooling line node, (x', y', z') is the *machine coordinates* of each cooling line node, which is then transformed to the *object coordinates* using the inverse transformation matrix described in section 3.3 before it is saved in the cooling line structures.

Since the cooling line must fall inside the die box, the system will detect the boundary and provide an alternative value(s) for the boundary value(s) if one or more node coordinates are beyond the die box. When one of the node coordinates is larger than the corresponding maximum value or smaller than the corresponding minimum value, which

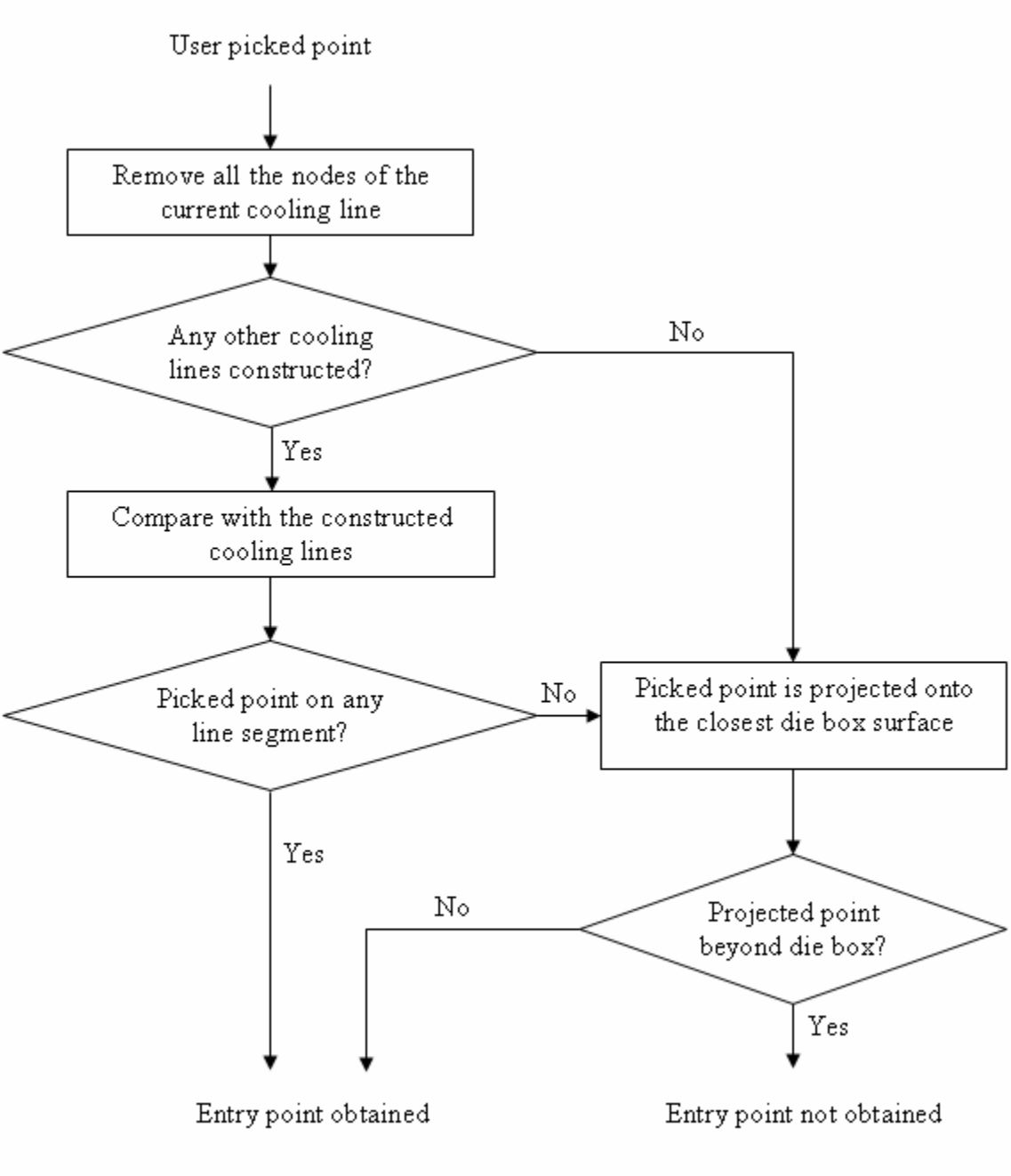
are all in *machine coordinates*, that boundary value is said to be violated. The user can choose to agree with the provided value(s) or to input other valid value(s) by hand.

4.1.2 Orthogonal Sketch

With this method, the cooling line is placed parallel to an axis direction of the *machine coordinate system*. The entry point is first selected, which may lie on the die box surface or on an existing cooling line, by double clicking the left mouse button. Then the cooling line is driven to go forward at a set pace along one *machine axis* by pressing one of the 6 function keys -- right arrow, left arrow, key <Home>, key <End>, up arrow, and down arrow, which represent 6 axis directions of the *machine coordinate system*: *x*+, *x*-, *y*+, *y*-, *z*+, and *z*-.

4.1.2.1 Picking Entry Point

The entry point can be selected at the beginning or in the middle of the cooling line construction. If the entry point is selected in the middle of the cooling line construction, the cooling line is to be constructed from the very beginning and the previous work on this cooling line is cancelled.

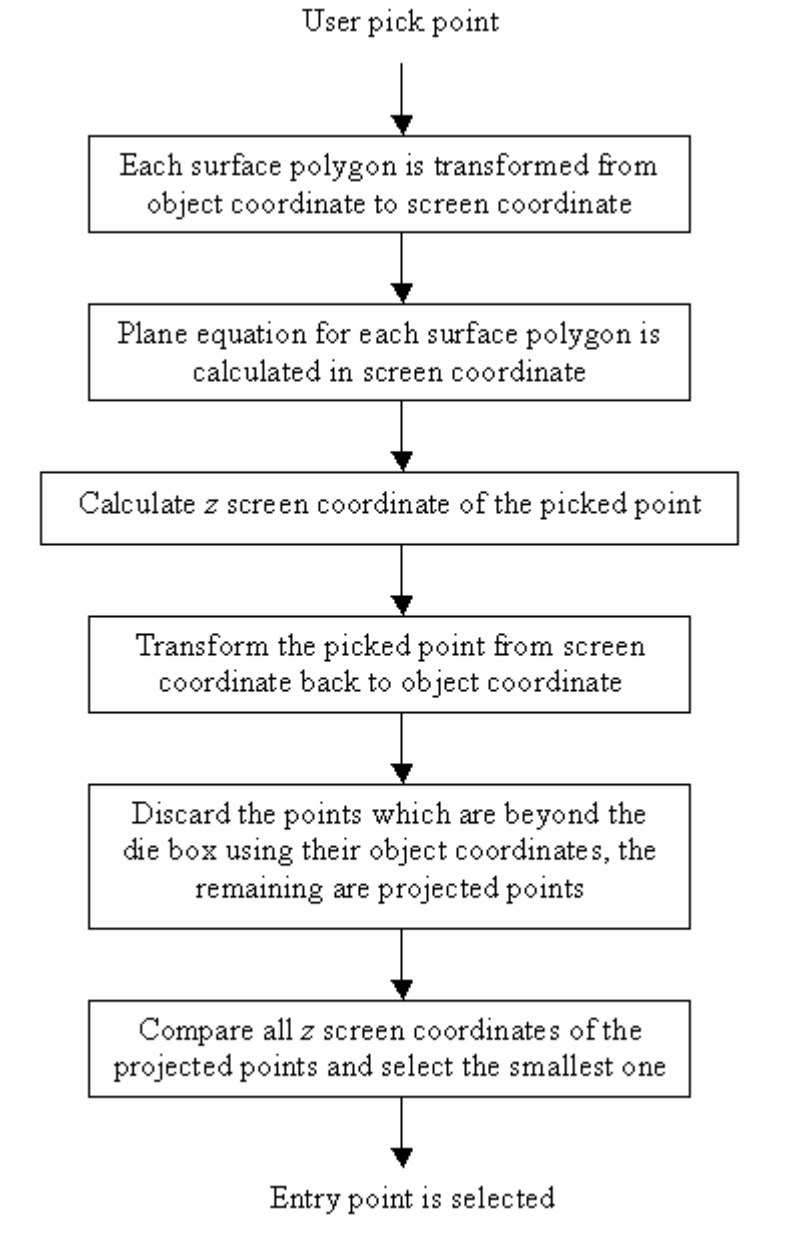


Section II-Figure 4.1: Pick entry point

The screen point is obtained by the user's left mouse button double click. Then it is compared with each of the existing cooling line segments, of which the end points are transformed from *object coordinates* to *2D screen coordinates*. If the picked screen point is on any existing cooling line segment on screen, the entry point is supposed to lie on that cooling line segment also in object space. Otherwise, the picked point is projected onto the closest die box surface to the user by certain transformations to obtain entry point if the projected point exists. Figure 4.1 illustrates the pipeline of this procedure.

If the picked screen point does not happen to be on any existing cooling line segment, it will be projected onto the closest die box surface. First, we calculated the projected point on each of the 6 faces, which is represented by a polygon. The polygon is transformed from *object coordinates* to *3D screen coordinates*. In this 3D screen coordinates, the plane equation of the polygon is established. And the picked point is fitted into this equation to get its *z* coordinate since its *x* and *y* coordinates are known from the pick operation.

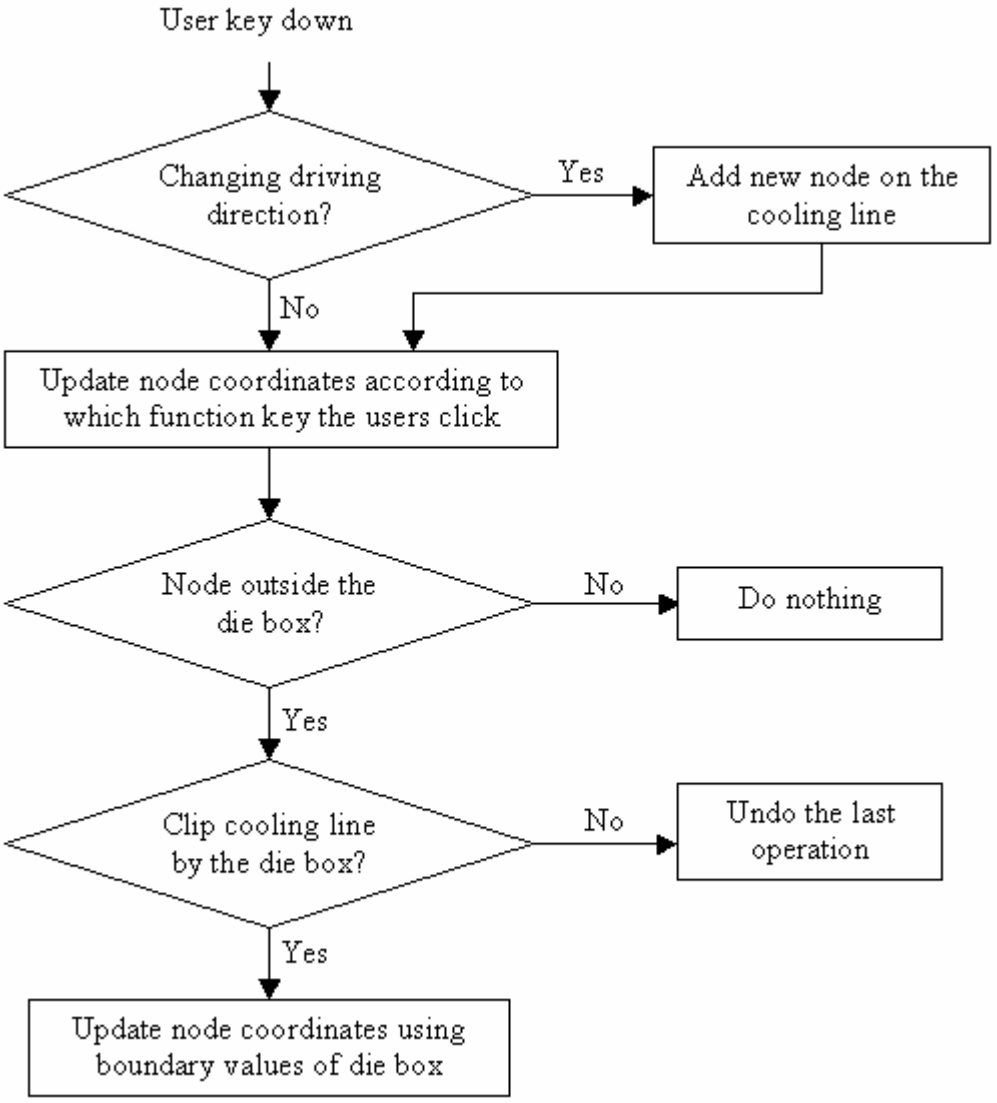
Then the picked point is transformed from the *3D screen coordinates* back to the *object coordinates* to see if this point is beyond the die box. It will be excluded if so. Otherwise, the obtained point is treated as a projected point. If the number of projected points found is 0, the selection of the entry point fails. If not 0, we compare the points to get the closest one to the user using the *z screen coordinate*. Since the *screen coordinate system* we are using sets the *z* coordinate between –1.0 and 1.0, and the smaller the *z* coordinate, the closer the point is to the user, we select the projected point with the smallest *z screen coordinate* as the entry point. The idea is shown in Figure 4.2.



Section II-Figure 4.2: Pick entry point on die box surface

4.1.2.2 Driving the Cooling Line Forward

Once the entry point is selected, the function keys are used to drive the cooling line forward at a set pace. Figure 4.3 shows how this works.



Section II-Figure 4.3: Driving the cooling line forward

When the user clicks one of the 6 function keys, the first thing to do is to judge if the driving direction has changed. If the driving direction changes, a new node will be added to the cooling line, and the coordinates of the new node are updated according to the set pace. Otherwise, only the coordinates of the last node are updated to show the node change. If this click happens right after the entry point is selected, it is treated the same

way as the direction is changed. Reversing the marching direction does not mean a change of the direction. Only the direction change is a turn, the principle of the changing direction is applied.

After the coordinates of the node have been updated, it is necessary to see if its coordinates are beyond the die box. If the node goes outside the die box, a message box is popped up to warn the user and there are two options. If the user agrees to clip the cooling line by the die box boundary, the coordinate of the node will be replaced by the violated boundary value. Otherwise, the operation is undone to do nothing.

The pace at which the node marches on may be set to any value using the edit box. It is the distance the node goes forward at one time's key down. The initial value of the pace is set to be $1/20^{th}$ of the maximum dimensional distance along the axes of the *machine coordinate system*. This feature can be used to precisely control the motion.

4.1.3 Free Sketch

This method requires that sketch planes, on which the cooling line nodes are picked by left mouse button double click, are drawn first. The size of the sketch planes is restricted by the die box since the cooling lines fall inside the die box. The cooling line nodes can then be picked on these sketch planes and connected to form a cooling line.

There are two methods to construct a sketch plane. One is by offset to a polygon on the STL model. The other is by the plane normal and a point on the plane to be constructed. The sketch plane may or may not intersect with the STL model. If it does, the STL model is invisible when the cooling line nodes are to be picked; the intersection line loops are

calculated and displayed along with the sketch plane. Otherwise, the STL model is visible and displayed in a transparent mode. In both situations, the relationship between the sketch plane and the STL model is provided.

4.1.3.1 Sketch Planes Defined by Offset to a Polygon Surface

There are two elements in constructing the sketch plane: the plane normal and a point on the plane. The polygon normal can be achieved by the method described in section 3.1.1, and the picked point can be calculated using the same method used in section 3.1.2. By the offset to the polygon, the plane normal stays the same while the point on the plane is obtained by moving the picked point a distance of the offset along the direction of the plane normal. Thus the two elements for constructing the sketch plane are achieved.

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + t * \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

where, *(x,y,z)* is the picked point on the polygon, *(u,v,w)* is the normalized polygon normal, *t* is the offset value, and *(x',y',z')* is a point on the sketch plane.

The sketch plane equation is thus obtained in the same way as in section 3.2.4; the intersection line loop with the die box is then calculated to form the visible profile of the sketch plane, as will be discussed in section 4.1.3.3. If the sketch plane intersected with the STL model, the intersection line loops are also calculated in order to provide the user the relative position between them, as will be discussed in section 4.1.3.4.

When a large offset value is input such that the sketch plane does not intersect with the die box, a warning information that says “Offset value too large, plane out of die box!” prompts the user to input the offset value again.

4.1.3.2 Sketch Planes Defined by Plane Normal plus a Location Point

The plane normal is defined by one of the three methods: the normal of a polygon, two points on the STL model surface, and an edge of a polygon. Here, the direction of the normal does not matter for the sketch plane since the cooling line nodes lie on the plane.

After the plane normal is obtained, a point which lies on either the STL model or one of the 6 die box faces will be selected. Selection of a point on the STL model has been discussed in section 3.1.2. A point may also lie on one of the 6 die box surfaces, as has been talked about in section 4.1.2.1 when an entry point of the cooling line is needed.

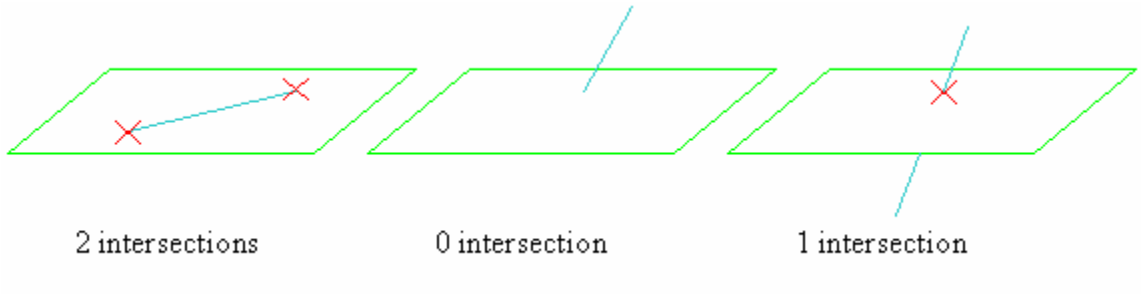
The remaining work for constructing a sketch plane is the same as the method by using the offset to a polygon on the STL model surface.

4.1.3.3 Intersection Line Loops of Sketch Planes with Die Box

After the sketch plane equation is achieved, we need to deal with the visualization. Since a flat plane is boundless and infinitely large, it can’t be shown on the screen unless its size is limited. In CastView, we present the sketch plane with its four vertices. The size of the sketch plane is dependent on the size of the die box.

First, we need to calculate the intersections of the sketch plane with the die box. Considering the die box is composed of 12 edges, we can calculate the intersection of the

sketch plane with each edge of the die box. There are 3 situations for the intersection between a line and a plane. The 3 situations are shown in Figure 4.4.



Section II-Figure 4.4: Intersection(s) between a line and a plane

The calculated intersections are then sorted so that they form a quadrangle. This quadrangle is shown on the screen to tell the users the relative location of the sketch plane with the die box. It also provides some information so that the users can easily locate the cooling lines.

4.1.3.4 Intersection Line Loops of Sketch Planes with STL Model

In CastView, the STL model is made up of numerous triangles, each of which has three edges. When the triangle has 0 or more than 1 intersection lines, it is parallel to or coplanar with the sketch plane and plays no role on the profile of the intersection line loops. So we only consider the triangles that have only 1 intersection line with the sketch plane. When all the intersection lines are calculated, they are connected to form the profile of the intersection line loop. Since the triangles are already sorted in a continuous

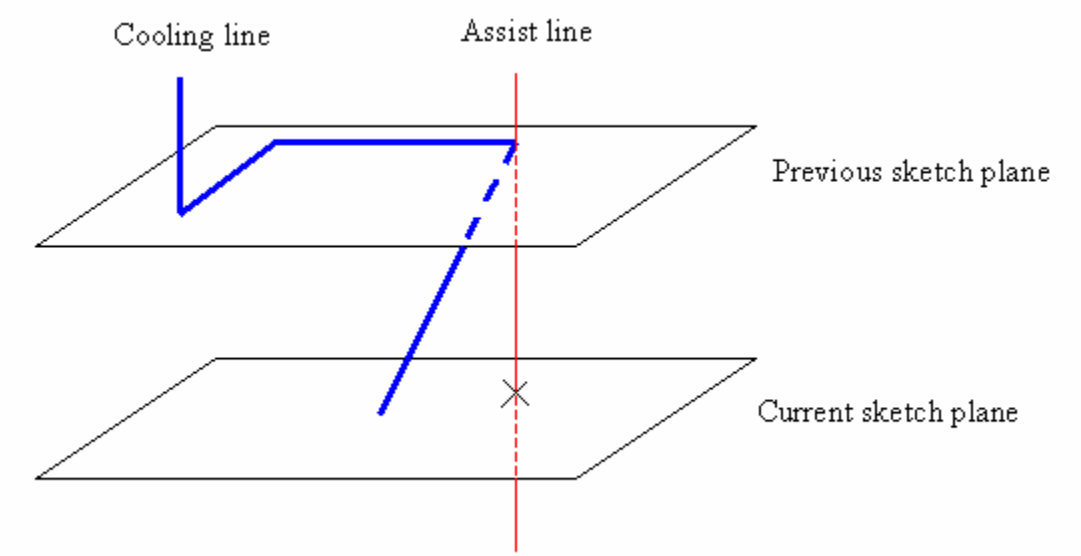
manner, the calculated intersection lines are naturally in the right order and no sort problem is involved here.

Again, the triangles are composed of three edges each. There are also 3 possible intersection situations between each edge and the sketch plane. If there are only 2 intersections totally between the triangle edges and the sketch plane, an intersection line is formed and added to the profile loops of the sketch plane.

4.1.3.5 Picking Points on Sketch Planes

The cooling line nodes can be picked anywhere on the sketch plane since the sketch plane is actually infinitely large. After the node is picked on the screen, its object coordinates can be obtained by calculation, as is discussed in Chapter 3 when two points are selected to construct the die opening direction. The picked screen point is then checked to see whether it happens to be on any existing cooling line segment on screen or not. If it does, the node is supposed to be on that cooling line segment also in the object space.

When the cooling line transits from one sketch plane to another, a line that goes through the last picked node on the previous sketch plane and is perpendicular to the current sketch plane is displayed. The line has an intersection with current sketch plane. When the cooling line node is picked on the current sketch plane, it is also checked to see whether the picked node is near that special intersection or not. If it is, the intersection will be copied to the selected node. This situation is illustrated in Figure 4.5.



Section II-Figure 4.5: Picking nodes on another sketch plane

When a node is outside the die box, the violated die box boundary value will replace the corresponding node coordinate.

4.1.4 Cooling Line Properties

Cooling lines have properties, such as the medium, diameter, flow rate, temperature and the heat transfer coefficient. Each cooling line may have different properties. The user can set the properties for the cooling line himself or use the default setting for the properties. The default setting can be changed, saved and loaded.

The heat transfer coefficient may be calculated from other parameters or be set to a specific value by the user. When the coefficient is defined by the user, the coefficient can be arbitrary. The calculation of the heat transfer coefficient for convection with a moving fluid used by the system is expressed as following.

$$h = a\frac{k}{D}\left(\frac{VD\rho}{\mu}\right)^{0.8}\left(\frac{\mu Cp}{k}\right)^{n}$$

where,

$\left(\frac{VD\rho}{\mu}\right)$ = Reynolds number

$\left(\frac{\mu Cp}{k}\right)$ = Prandtl number

D = diameter of the line

k = the thermal conductivity of fluid

h = the surface heat transfer coefficient

a = constant, here is 0.023, suggested by author Colburn

n = constant, here is 0.333, suggested by author Colburn

V = velocity of fluid

ρ = Density of fluid

μ = Dynamic viscosity of the fluid

Cp = Specific heat of fluid

The parameters k, ρ, μ, and the Prandtl number can be obtained according to different temperatures and different mediums by using a lookup table. Then the Reynolds number is calculated with V and D known. Finally, the heat transfer coefficient is achieved using the above equation.

In order to satisfy the different custom of different users, the parameters are allowed to be input in different units, including English units and SI units, as is shown in table 4.1.

Section II-Table 4-1: Parameters and units

Parameter	**Units**
Diameter	mm, cm, m, in., ft
Flow rate	m/s, ft/h, m^3/h, ft^3/h, ft^3/min, gal/min
Temperature	$°C$, $°F$
Heat transfer coefficient	W/m^2-K, BTU/hr-ft^2-F

4.2 Modifying Cooling Lines

During or after the construction of a cooling line, modification is allowed. There are three methods to modify the cooling lines. One method is by editing the node coordinates of the cooling line, as is achieved by using the same dialog that is used when the node coordinates are input to construct the cooling line. An unwanted node may be removed, the node coordinates may be changed, and a new node may be added or inserted to the cooling line.

Another method is orthogonal method. Using this method, a node or a line segment can be moved or deleted. When the node or line is to be moved, the pace of the movement can be also set to any value by the user. If the node or the line will be moved outside the

die box, a warning message is given to ask whether the user wants to continue or not. If yes, the cooling line will be clipped by the die box boundary; otherwise, the operation is undone. The node or the line to be moved or deleted is rendered in another color to give the user a clear vision and understanding. If the node or the line is to be deleted, a confirmation dialog is popped up for the user to reconsider their choice.

The above two methods can be used to modify a cooling line that is constructed by any method. The last method is the graphical editing. This method is exclusively used for modifying cooling lines that are constructed using sketch planes. The user may drag the desired node or line segment to a new position. A node or a line segment can also be deleted by left mouse double clicking.

An undesired cooling line or sketch plane can be removed from the cooling line list or sketch plane list. And the properties of a cooling line may be modified at any time during or after its construction.

4.3 Saving and Loading Cooling Lines

Since the cooling line data may be retrieved later and it is needed for the thermal analysis, it is saved in a cooling line file. Different cooling line files can be used for the input of the thermal analysis. All the sketch planes and the cooling lines are saved in one cooling line file. For each cooling line, the number of the nodes, the properties of the cooling line, and the coordinates of the nodes are saved.

If there are less than 2 nodes for a cooling line, that cooling line will not be saved since this kind of cooling line is totally meaningless. The cooling line file format is shown in appendix B.

4.4 Summary

This chapter discussed the cooling line functions, including how to add a new cooling line, modify a specified cooling line, save and load cooling lines. Since the cooling lines must fall inside the die box, a die configuration must have been defined when the cooling line functions are used. Both the die configuration and the cooling lines functions are based on the STL model. In next chapter, a voxel model will be used for the wall thickness analysis, which can let the user see the wall thickness distribution globally and serves as an input for the “castability assessment” system.

5. WALL THICKNESS MAPPING

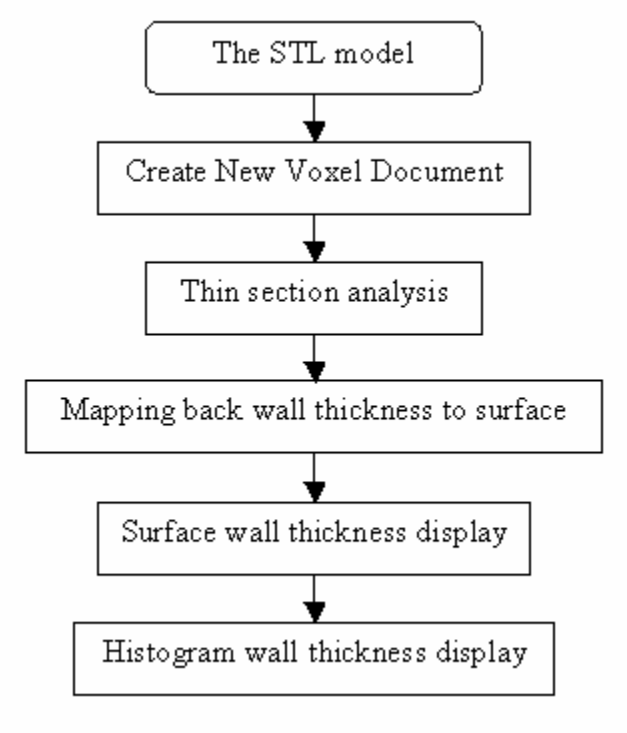
Geometric features of the STL model are important information needed in the castability assessment, which can help the user to measure the overall castability of a design and to evaluate the trade-offs inherent in configuring the die casting die. The geometric variable that has the largest impact on the mechanical performance of the casting part is wall thickness. In CastView, the half wall thickness is presented by thick section analysis and thin section analysis. The thicker or thinner section where the half wall thickness is larger or smaller than the threshold value is displayed, while the half wall thickness also can be shown on a slice.

Since it is highly desirable to design the part at reasonably uniform wall thickness and avoid abrupt changes in wall thickness, it will be favorable to give the designers an overall view of the wall thickness throughout the part. In this task, the wall thickness is mapped onto the voxel model surface and the geometric features of different wall thickness are rendered in different colors.

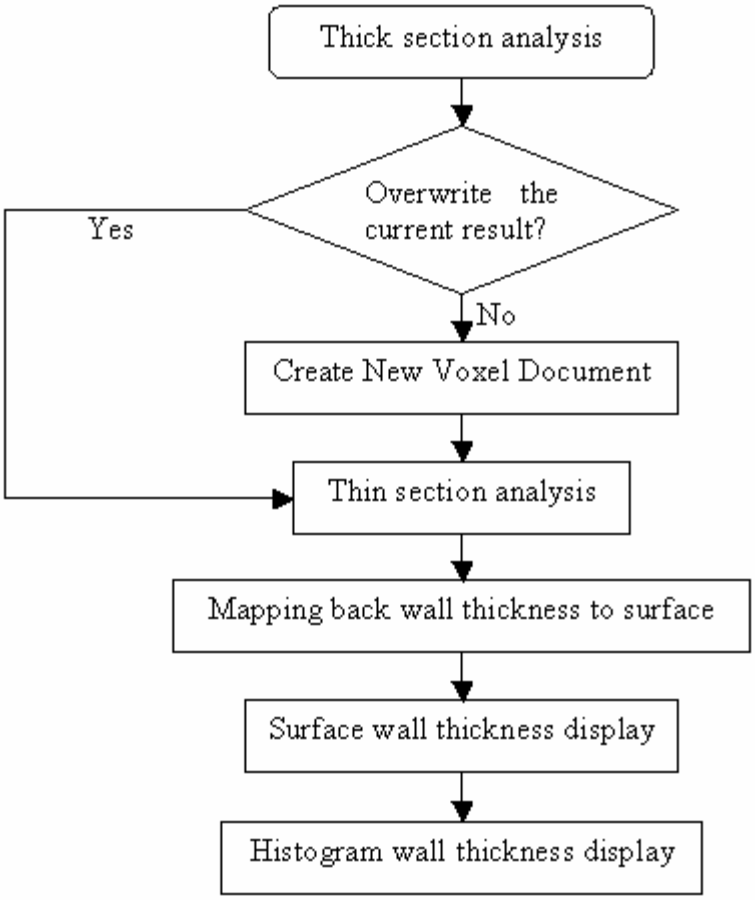
5.1 Pipeline of the Procedure

The mapping of the wall thickness onto the voxel model surface is based on the result of the thin section analysis, during which the skeletons of the geometric features are extracted. Each skeleton has a half wall thickness value for that geometric feature. From the skeletons, the thickness values are expanded out until reaching the part surface so that the wall thickness can be displayed on the voxel model surface. The pipeline is shown in

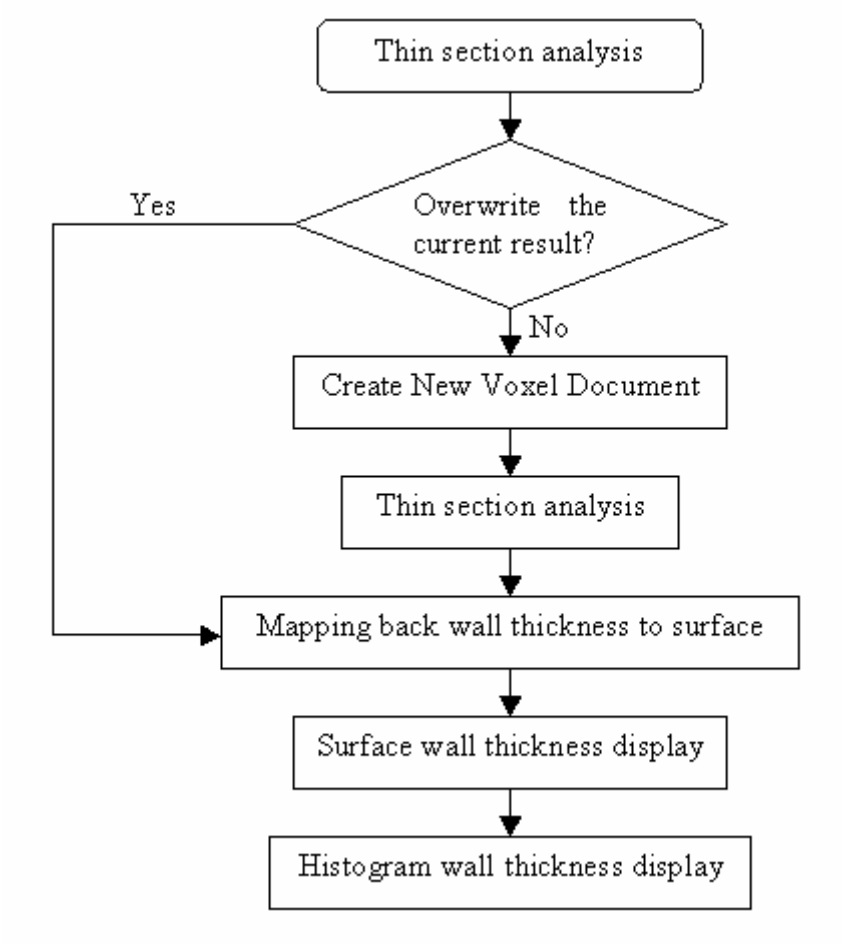
Figure 5.1, 5.2, and 5.3 for mapping the wall thickness back from the STL model, thick section analysis result, and thin section analysis result respectively.



Section II-Figure 5.1: Wall thickness mapping back from STL model



Section II-Figure 5.2: Wall thickness mapping back from thick section analysis result

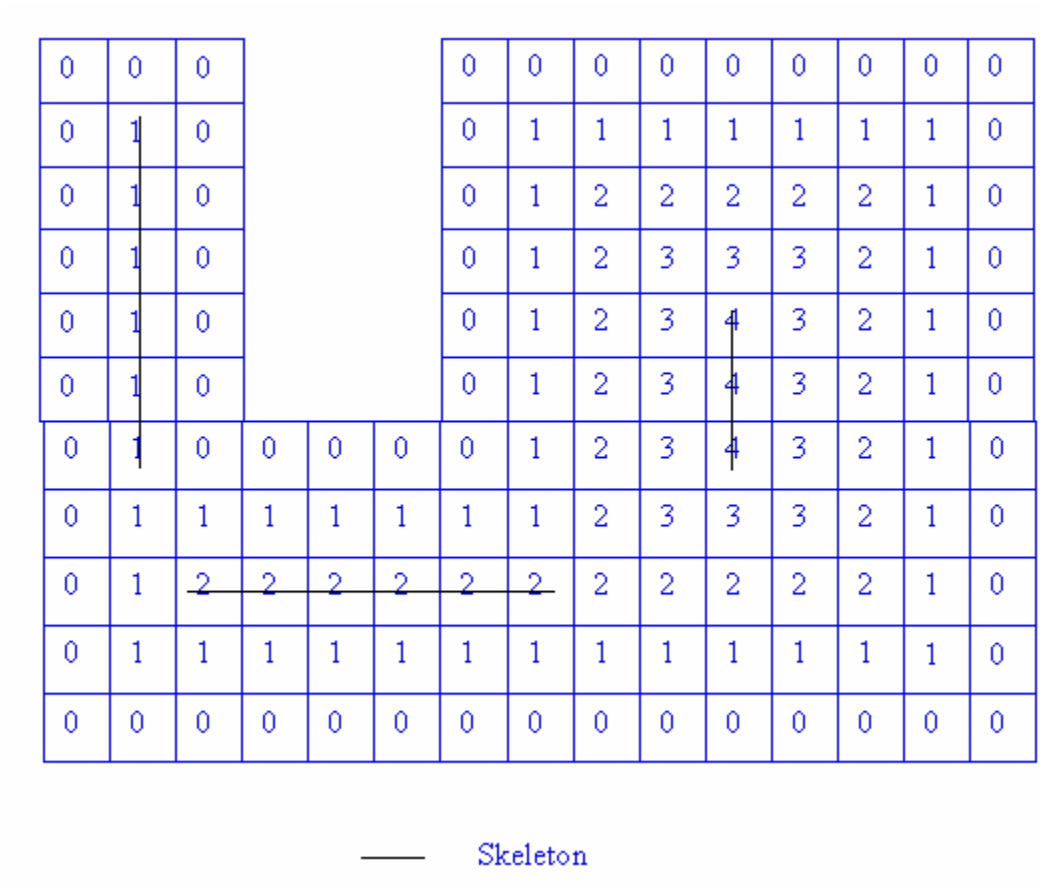


Section II-Figure 5.3: Wall thickness mapping back from thin section analysis result

5.2 Obtaining the Feature Skeletons

The wall thickness mapping is based on the skeleton, which is obtained during the thin section analysis. The skeleton is attained by "onion peeling", which is illustrated in Figure 5.4. The surface voxels are first peeled and numbered with 0, then the next layer to the surface voxels are peeled and numbered with 1, etc. The numbers assigned to the voxels are the number of layers, which are peeled off from the surface until the voxels are exposed. After the skeletons are found, all of the voxels are emptied so that we can fill

them with the right values when the wall thickness is mapped back. Each voxel is filled only once during the wall thickness mapping back process.



Section II-Figure 5.4: Skeleton calculation by "onion peeling"

5.3 Mapping Wall Thickness Back to the Voxel Model Surface

During the thin section analysis, only the skeleton voxels have "thickness values" and only the skeleton thinner than the core value specified is rendered in green by the display when the slider is set to a corresponding value. Everything else will be rendered as gray.

After the wall thickness is mapped onto the voxel model surface, all the wall thickness is displayed in different colors with different values.

Wall thickness mapping is started at the skeleton. First, the neighboring voxels with a value 1 less than the local skeleton are reset to the skeleton's value. At the next step, all the neighbors two less than the original skeleton are reset, etc. With connectivity maintained, the surface voxels are reset values equal to the core values. Thus, different wall thickness is displayed on the screen using rendering mechanism.

5.3.1 Selecting Favorable Algorithm for Wall Thickness Mapping

The mapping back procedure is described in the previous section. But since the skeleton may be noisy from the thin section analysis, the results can be different when it is implemented. The noise, which is voxels not correctly classified as skeletons, usually exists as medium value skeletons near large value ones, based on experience.

There are totally 8 candidate algorithms for the wall thickness mapping back, which can be classified into 2 groups – thickest first and thinnest first, according to whether thickest or thinnest skeleton is expanded first.

For example, if the skeleton values are 4, 2, 1, the 8 algorithms are shown in Table 5.1 and Table 5.2.

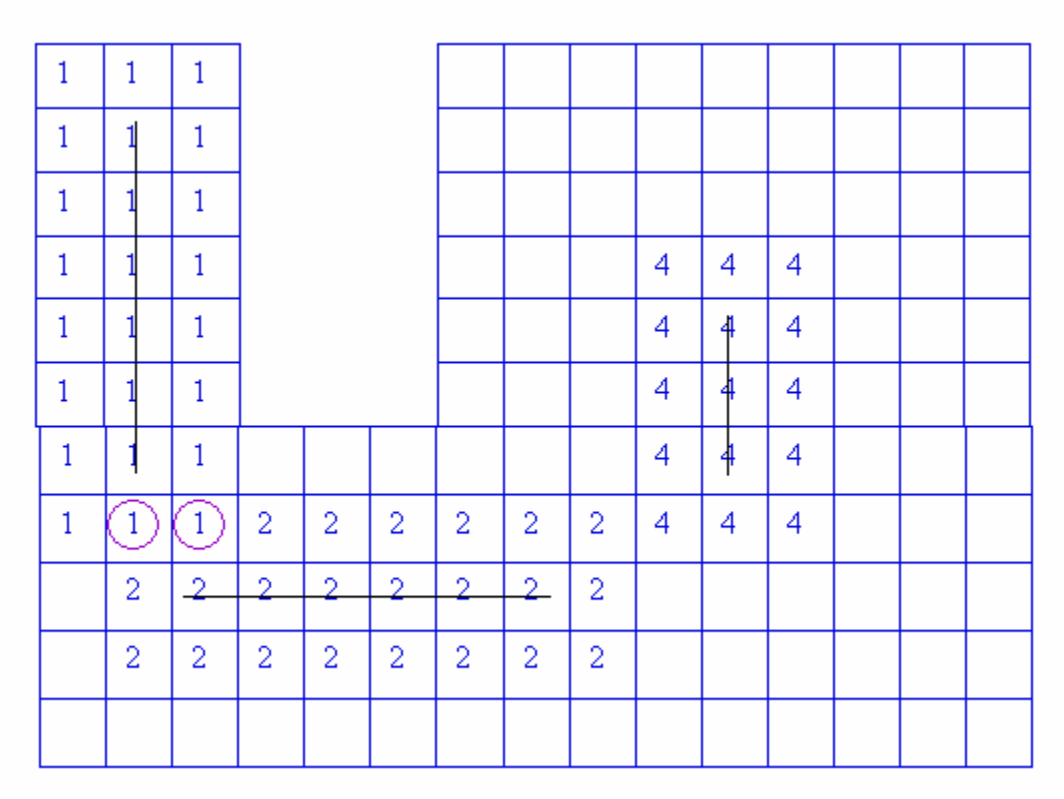
Section II-Table 5-1: Thickest first algorithms

Algorithm #	Sweep sequence of skeletons	No. of sweeps	Sequence problem
1	4; 4; 4/2; 4/2/1	4	Yes
2	4; 4; 4, 2; 4, 2, 1	7	No
3	4; 4; 2, 4; 1, 2, 4	7	No
4	4, 4, 4, 4; 2, 2; 1	7	No

Section II-Table 5-2: Thinnest first algorithms

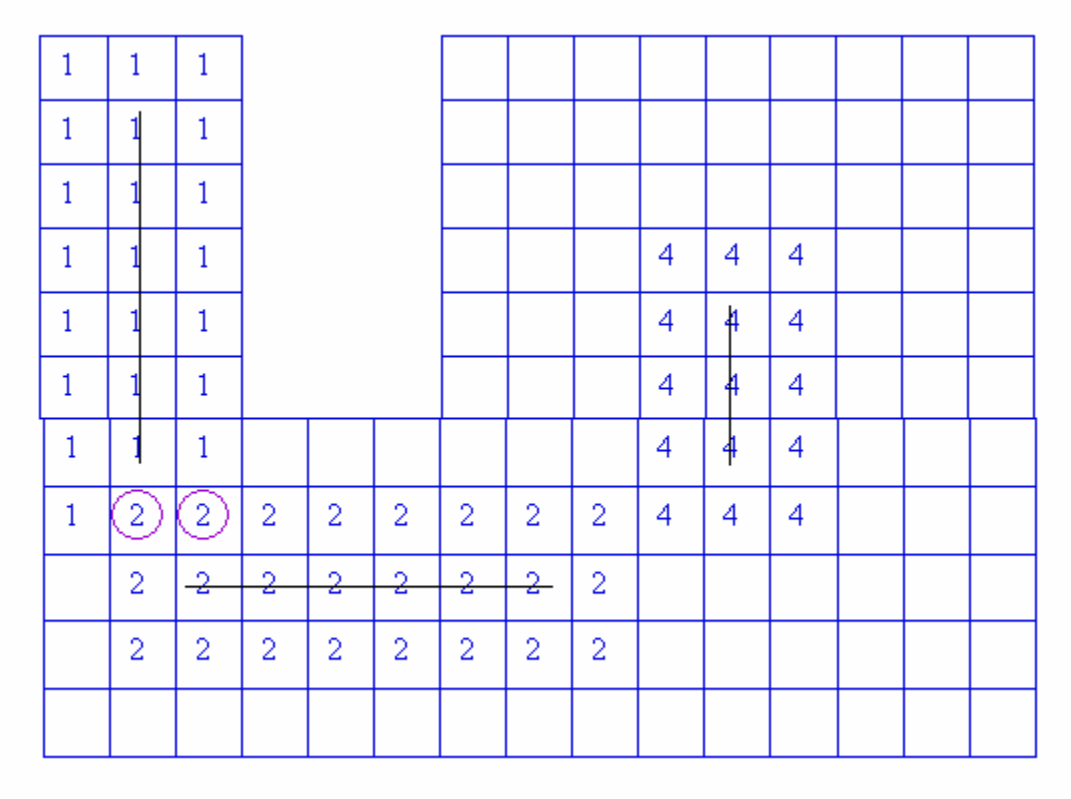
Algorithm #	Sweep sequence of skeletons	No. of sweeps	Sequence problem
5	4/2/1; 4/2; 4; 4	4	Yes
6	4, 2, 1; 4, 2; 4; 4	7	No
7	1, 2, 4; 2, 4; 4; 4	7	No
8	1; 2, 2; 4, 4, 4, 4	7	No

When the expanded skeleton values go through all of the voxels, one sweep is finished. If there are more than one skeleton values expanded through one sweep, as in algorithms #1 and #5, sequence problem may occur, which means that the result is different for the situation when the larger skeleton values are expanded before the smaller ones compared to the reverse situation The sequence problem is illustrated for algorithm #5 in Figure 5.5 and Figure 5.6.



Section II-Figure 5.5: Mapping algorithm #5(1)

In Figure 5.5, we map the skeleton values from left to right column by column, from top to bottom in a column. This means that the 1s are expanded to their neighbors first, so the voxels circled are filled with 1.

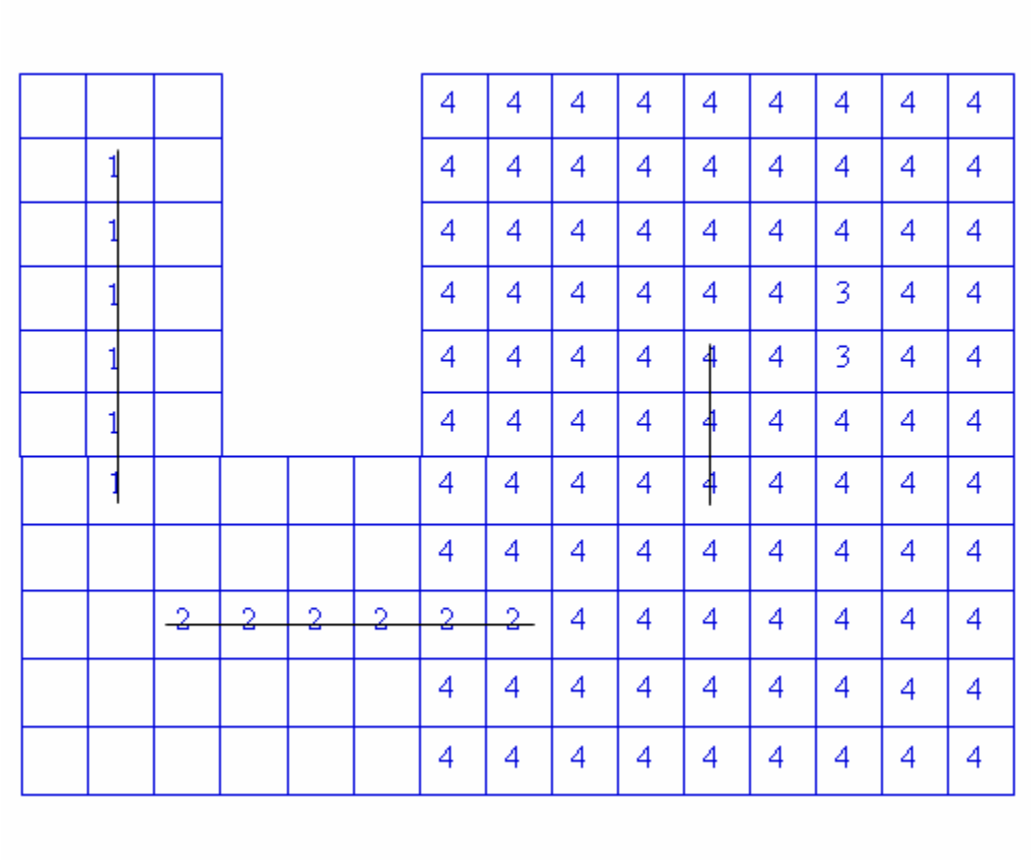


Section II-Figure 5.6: Mapping algorithm #5(2)

In Figure 5.6, the skeleton values are mapped right to left column by column while still from top to bottom in a column. The circled voxels are filled with 2 in this way. So when more than one skeleton values are extended to their neighbors in one sweep, the sequence problem exists, which is the case in algorithms #1 and #5. So the algorithms #1 and #5 are discarded.

Another problem is noise. Since noise exists as medium value skeletons near large value ones, the neighbors of the large value skeletons will be occupied if they are mapped back to the voxel model surface first. When it is time for the medium value skeletons to expand their skeleton values to their neighbors, there will be no unoccupied neighbor voxels so the medium value skeletons can not be expanded. Figure 5.7 shows an example

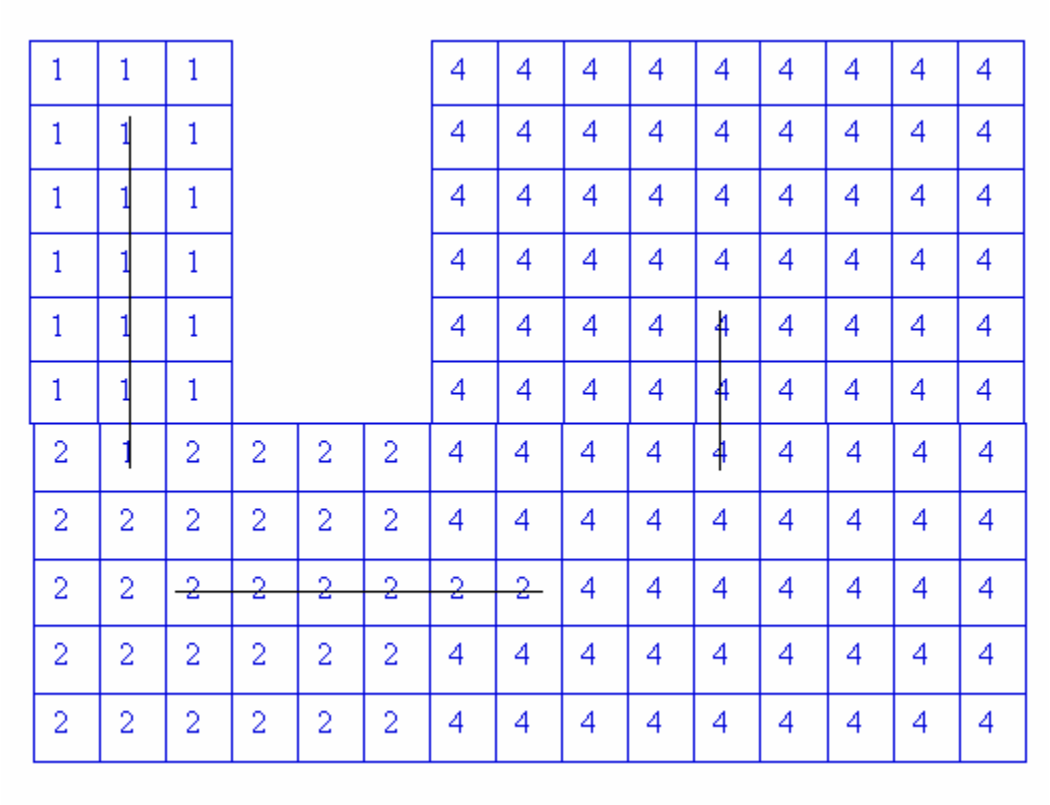
of this case. After skeleton 4 is mapped back for 4 sweeps, skeleton 3, which is actually noise, can not grow out since all the neighbor voxels have been occupied. For the reason that the sequence problem hurt the result badly with high voxel resolution and severe noise, we select algorithm #4 to map back the wall thickness onto surface.



Section II-Figure 5.7: Elimination of noise

5.3.2 Wall Thickness Mapping Result

In algorithm #4, 7 sweeps are made: 4, 4, 4, 4, 2, 2, 1. That is, the skeletons with value 4 are mapped back for 4 sweeps, straightly to the voxel model surface. Then the skeletons with value 2 are mapped back for 2 sweeps. Finally the skeletons with value 1 are mapped for 1 sweep. The final result of the example is shown in Figure 5.8.



Section II-Figure 5.8: Wall thickness mapping back result

Generally, if a skeleton value is *m*, it will take *m* sweeps for this skeleton value to be expanded to the model surface. So if the skeleton values are *n*, *n-i*, *n-j*, *n-k*, …, which are not necessarily continuous, the minimum number of sweeps that may be taken is *n + (n-i) + (n-j) + (n-k) + ….*

After taking the minimum number of sweeps, there may still be some voxels unfilled because there may be some error during the "onion peeling" algorithm. We may fill these voxels by taking another several sweeps using all the skeleton values instead of just one specific skeleton value. Until all the voxels are occupied, the process is done.

5.4 Wall Thickness Quantitative Display

After each voxel of the STL model is filled with one skeleton value, the result is displayed on the voxel model surface. But this is not enough since the quantitative analysis has not been done. A histogram is drawn so that the wall thickness of each geometric feature can be compared quantitatively. The wall thickness variation can be fed to the evaluation procedures for the castability assessment.

5.4.1 Average Wall Thickness

The wall thickness has a large impact on the mechanical performance of the casting part, and maximum mechanical properties can be achieved in die castings when the wall thickness is in the range of .078 to .150 inches for alloys such as aluminum, zinc, and magnesium [13].

The minimum wall thickness that can be cast without the risk of cold shut depends on configuration of the casting, position of the gate, metal flow in the die, and the projected area of the die cast part.

Average wall thickness is usually used for the designers to decide whether the design is feasible for the casting part to be produced by the die casting process or not. The average wall thickness is calculated as the ratio of the volume to the surface area.

$$Average\ Wall\ Thickness = \frac{Volume}{1/2\ Surface\ Area}$$

5.4.2 Wall Thickness Distribution

The average wall thickness is not enough for a precise castability evaluation. Since we've got the wall thickness for each voxel of the STL model, a histogram is used to show the wall thickness distribution, which may be the input of the castability evaluation system. From the histogram, a more accurate average wall thickness and its deviation can also be calculated. The user can even tell if the wall thickness is uniform just from the appearance of the histogram.

5.4.3 Wall Thickness Visual Option

After the thin section analysis, the skeleton values are obtained. These values are actually the distances from the voxel model surface, as can be seen from the algorithm of "onion peeling". According to the generic meaning of the wall thickness, its value should be twice of the value of the distance from the voxel model surface. In CastView, the user can select any mode according to their preference. Wall thickness option is set as the default setting.

5.5 Saving the Wall Thickness Data

After the analysis, the wall thickness data may be input into the evaluation system for the castability evaluation. The wall thickness data is saved in a text file so that it can be read by the evaluation system. The wall thickness data file format is shown in appendix C.

5.6 Summary

This chapter talked about how to display the wall thickness distribution on the voxel model surface. The wall thickness distribution can then be shown on a histogram and the

statistics may be performed based on this histogram. The data can also be input into the castability assessment system for the castability evaluation.

Next chapter will show the graphical user interface for the work described in chapter 3, chapter 4, and chapter 5. The friendly graphical user interface makes the application much easier to use and understand.

6. GRAPHICAL USER INTERFACE

CastView is developed using Visual C++, MFC, and OpenGL. The graphical user interface is designed in Visual C++ and MFC.

6.1 Toolbars

Toolbars are important components for a perfect Windows application, though they are not necessarily a must.

The buttons in a toolbar generate commands through program handler functions when they are clicked. There may be three types of buttons on a toolbar according to their different behavior: pushbuttons, check boxes, and radio buttons. The buttons on the toolbar containing die configuration and cooling lines functions are all pushbuttons.

A toolbar may have tool tips when the user moves the mouse over its buttons. The tool tip offers a hint of the button's purpose. The description of the button functions may also be displayed on the status bar, which is a control bar with one or more panes.

Figure 6.1 shows the toolbar containing the die configuration functions. Figure 6.2 shows the toolbar with the cooling line functions.

Section II-Figure 6.1: Toolbar containing die configuration functions (the first 3 buttons)

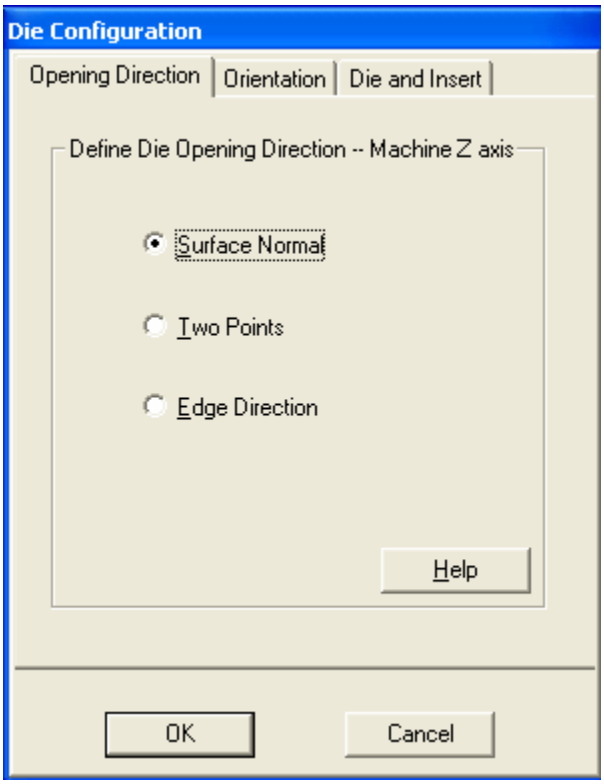
Section II-Figure 6.2: Toolbar with cooling line functions

6.2 Die Configuration Property Sheet

A property sheet that is also called a tab dialog box is generally used to modify the attributes of some external object, which is the STL model in this system. A property sheet is mainly composed of three parts: the containing dialog box, one or more property pages shown one at a time, and a tab at the top of each page that allows the user to select that page by clicking it. A property sheet may be very useful for situations where you have a number of similar groups of settings or options to change, which may be arranged into several property pages.

In order to exchange data between the property sheet and the external object it is modifying when the property sheet is open, the property sheet must be a modeless one so that the user does not have to close the property sheet before manipulating the external object using other parts of the application.

Since the die configuration including the definition of die opening direction, die orientation, and the die box and the insert box, the user interface is designed using a modeless property sheet that contains three pages, each of which is used to manage one kind of setting composing the die configuration. The property sheet for the die configuration is shown in Figure 6.3.

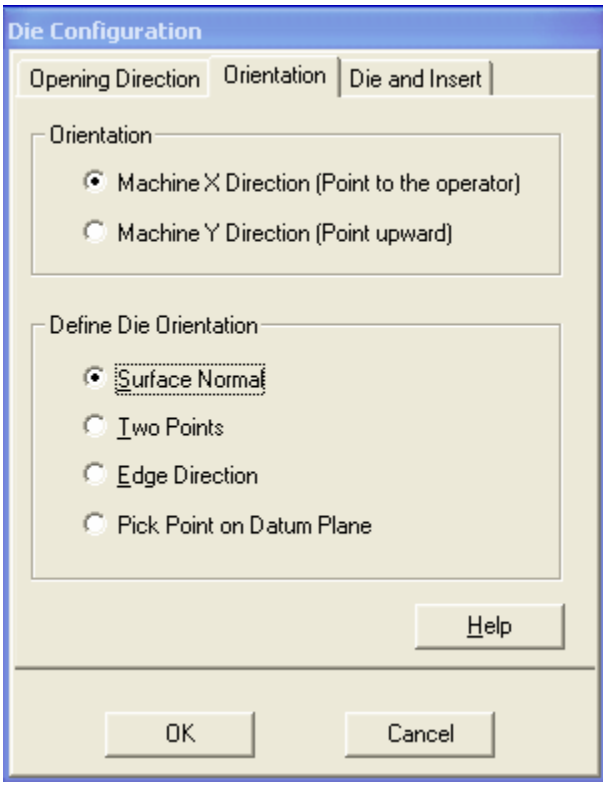


Section II-Figure 6.3: Die configuration property sheet

6.2.1 Die Opening Direction Property Page

The die opening direction must be defined first before the definition of the die orientation and the die box and the insert box. The user can select any of the three methods to proceed to define the die opening direction by left mouse button double click on the STL model and may reverse it by right mouse button click. The die opening direction property page is shown in Figure 6.3.

6.2.2 Die Orientation Property Page



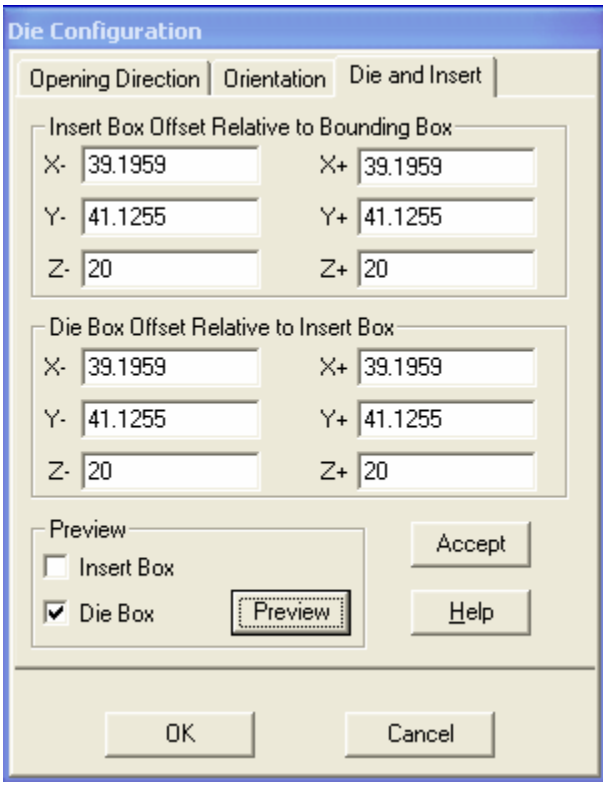
Section II-Figure 6.4: Die orientation property page

After the die opening direction is defined, the *z machine axis* is decided. In order to construct the *machine coordinate system*, the die orientation is defined afterward as the *x or machine axis*. Then the right hand rule is applied to get the other *machine axis*. The die orientation property page is shown in Figure 6.4.

6.2.3 Die Box and Insert Box Property Page

After the *machine coordinate system* is constructed, it is time to define the die box and the insert box. The offset values are input by the user. The die box and the insert box may

be previewed so that the undesirable results can be avoided. Figure 6.5 shows the die box and insert box property page.



Section II-Figure 6.5: Die box and insert box property page

If the die configuration property sheet is opened after the die box and the insert box have been defined, the offset values used to define them are shown in the edit boxes. If the die box and the insert box have not been defined when the die configuration property sheet shows up, the default values, which are 1/4 of the bounding box dimensions in the *machine coordinate system*, are shown in the edit boxes. The bounding box is the minimum box that could contain the casting part inside. The edges of the bounding box are parallel to one of the *machine axes*,

If the die construction has not been finished when the user push button “OK”, the operation will fail and a warning message will be given.

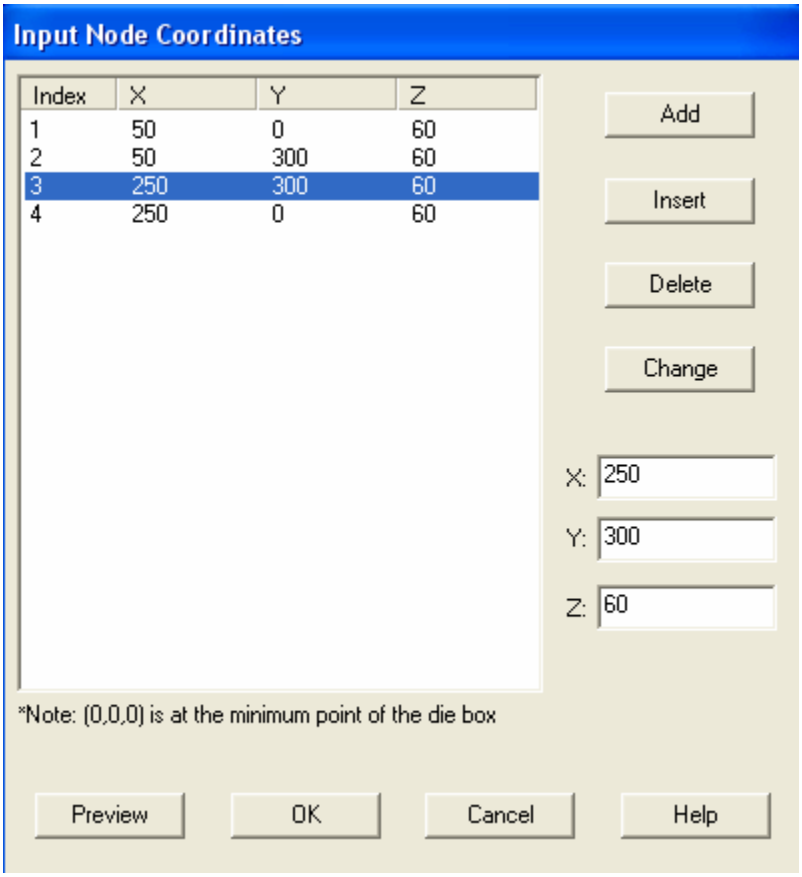
6.3 Cooling Line Sketcher

After the die configuration is defined, the cooling line dialogs may be popped up to draw new cooling lines, change cooling line properties, edit or modify existing cooling lines.

6.3.1 Sketching Cooling Line Dialogs

6.3.1.1 Sketching a Cooling Line by Inputting Coordinates

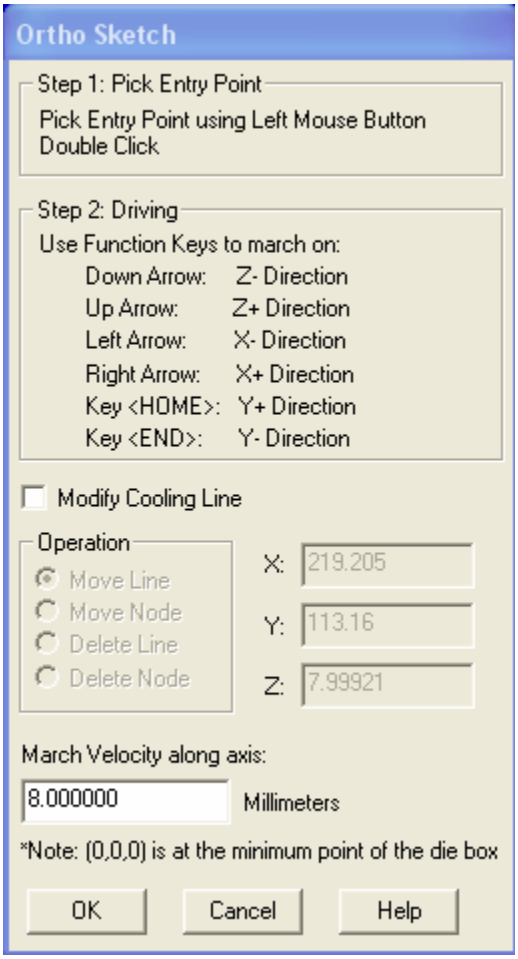
There are three methods to sketch a cooling line. Figure 6.6 illustrates how a cooling line is constructed by inputting the coordinates for each of its nodes. A table is used to display the coordinates of all of the nodes that form the cooling line. The “Add” button is pushed to add a new node that has the coordinates in the edit boxes to the end of the cooling line. “Insert” button is used when the new node is to be put right before the highlighted item. The highlighted item will be removed if the “Delete” button is pushed down. The coordinates of the current item can also be changed to the values in the edit boxes by clicking the “Change” button.



Section II-Figure 6.6: Sketching a cooling line by inputting node coordinates

6.3.1.2 Sketching a Cooling Line by Orthogonal Sketch

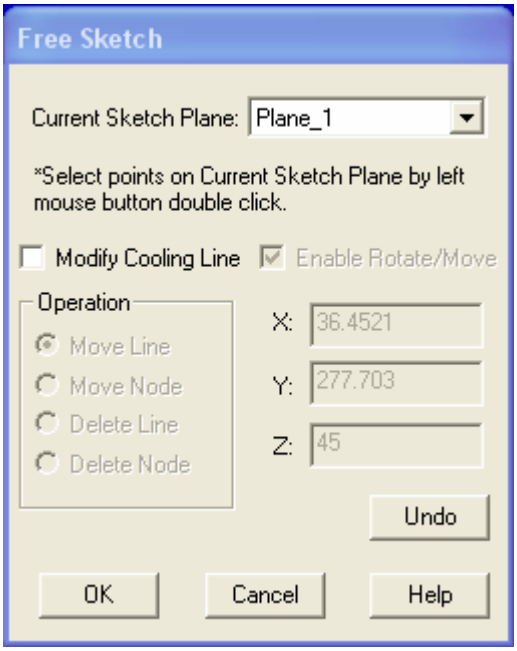
Figure 6.7 shows how to sketch a cooling line using the orthogonal method. The operations are described on the dialog for each step. The distance that the cooling line will go with each key press is displayed and may be changed in the edit box.



Section II-Figure 6.7: Sketching a cooling line by orthogonal sketch

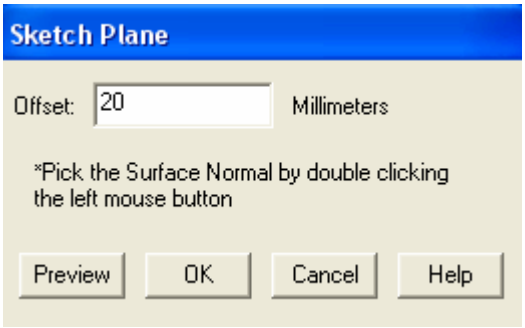
6.3.1.3 Sketching a Cooling Line by Free Sketch

Free sketching a cooling line is shown in Figure 6.8. With a sketch plane selected and displayed, the cooling line nodes are then picked one by one by double clicking the left mouse button. All the nodes must lie on the sketch plane and fall inside the die box.



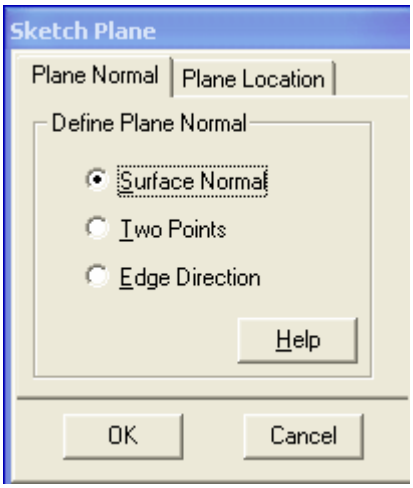
Section II-Figure 6.8: Sketching a cooling line by free sketch

A sketch plane used in the free sketching can be constructed by one of the two methods: offset from a polygon, and the plane normal plus a point method. If the first method is chosen, the input offset dialog shown in Figure 6.9 will then be popped up to urge the user to select a polygon on the STL model and input an offset value. The new sketch plane may be previewed before it is ok.

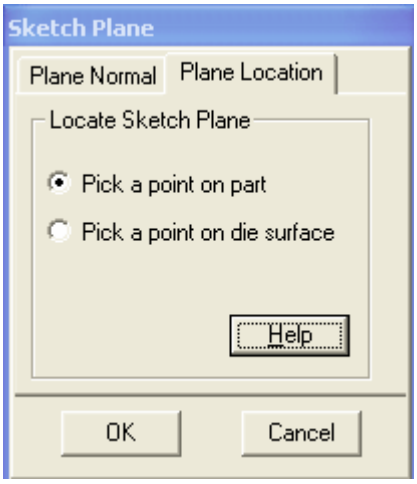


Section II-Figure 6.9: Inputting offset value to construct new sketch plane

If the new sketch plane is to be established using the plane normal and a point, another dialog is shown to direct the user to construct the sketch plane step by step. The dialog is as following in Figure 6.10 and Figure 6.11. There are three ways to define the plane normal, as is similar to the definition of the die opening direction. Two methods are selectable to locate the sketch plane. The location point may be on the STL model or on one of the die box surfaces.



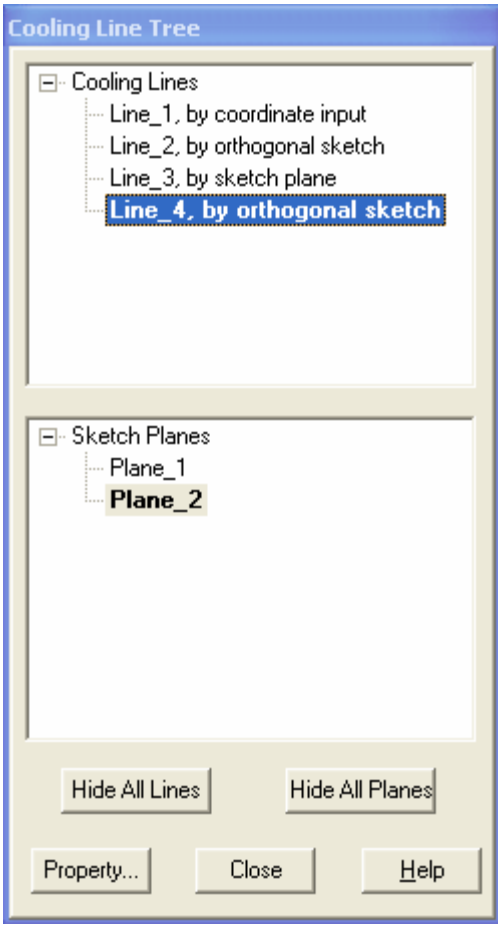
Section II-Figure 6.10: Plane normal plus a point to construct new sketch plane: step 1



Section II-Figure 6.11: Plane normal plus a point to construct new sketch plane: step2

6.3.2 Cooling Line Tree

A cooling line tree dialog will pop up allowing the userto view the cooling lines/sketch planes previously constructed. The user can view a specific existing or all the cooling lines/sketch planes. When a specific cooling line is selected, the user may modify this cooling line or its properties. The cooling line tree dialog is shown in Figure 6.12.

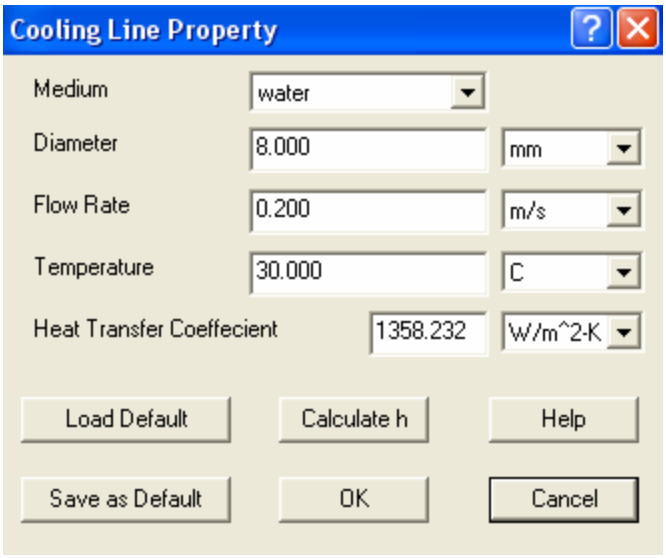


Section II-Figure 6.12: Cooling Line Tree

6.3.3 Setting Cooling Line Properties

Cooling line properties are all set in the dialog shown in Figure 6.13. The values are set and the units are selected before the heat transfer coefficient can be calculated. Users can

set another value to the heat transfer coefficient instead of letting the system calculate it. Values can be saved to a file as default settings, which can be loaded whenever needed.



Section II-Figure 6.13: Cooling line properties Dialog

6.3.4 Modifying Cooling Line Dialogs

Modification can be done during or after the construction of a cooling line. There are three methods to modify a selected cooling line. One method is by editing its node coordinates. Another is by orthogonal method, and the other method is graphical editing.

When a selected cooling line is to be modified by editing its node coordinates, the dialog in Figure 6.6 shows up and the user can edit the node coordinates as much as they want.

If the orthogonal method is chosen to modify the selected cooling line, the same dialog in Figure 6.7 shows up and allows the user to choose the operation that they want to do to modify that specific cooling line.

First, the node or line segment that is to be modified or to be removed is picked by left mouse button double clicking on that node or line. Then if it is to be removed, a message box is popped up for the user to confirm the operation. If the node or line is about to be moved, the function keys are used to move it along the axes of the object coordinate system at a certain pace, specified in the edit box. The user may change the pace if they want. For visual purposes, the node to be modified or removed is surrounded by a red rectangle on screen and the line to be modified or removed is rendered in a color other than that of a usual cooling line.

If the user wants to modify a cooling line by graphical editing, the same dialog in Figure 6.8 pops up for the modification. A node or line segment can be removed by left mouse button double clicking or modified by pressing the left mouse button and dragging it to a new position.

6.4 Summary

This chapter has provided a friendly graphical user interface for the die configuration and the cooling line functions. Audible or visual effects are used to give the user some feedback of the successful operation. For example, when the die opening direction is selected, a beep sound is made; when a line segment is selected for modification, it is rendered in a different color from the usual cooling line color.

Several examples will be given in next chapter.

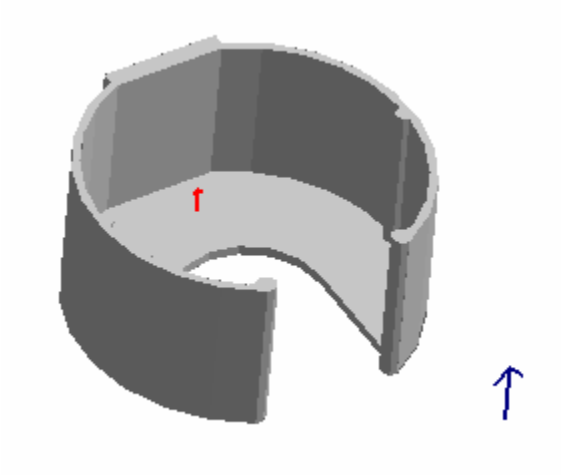
7. IMPLEMENTATION AND EXAMPLES

A lot of examples have been used to test the functions of the system. All the tests are performed on a Pentium 800MHz PC, which runs Windows 2000 and has 512MB RAM. We use the casting part of the bowl as an example of the die configuration and the cooling line sketcher. The axes of the coordinate system *x, y, z* are rendered in red, green and blue respectively.

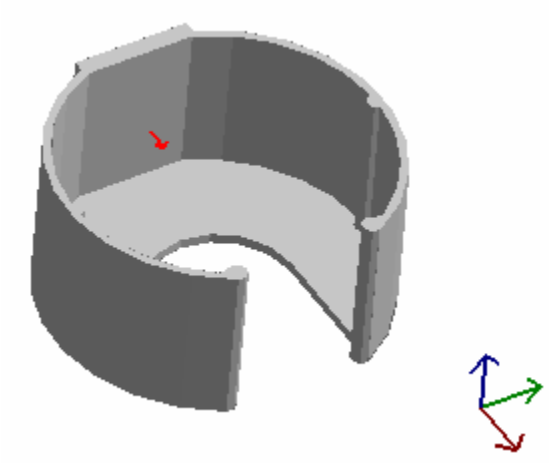
7.1 Die Configuration

Figure 7.1 is the STL model after the die opening direction is picked. The die opening direction serves as the *z machine axis*. The *z machine axis* is shown in blue. The die opening direction is shown as an arrow in red.

Figure 7.2 is the STL model after the die orientation is picked. The die orientation vector can be the *x or y machine axis* depending on the user's selection. In this example, the die orientation vector is the *x machine* axis. The *axes* of the *machine coordinate system* are shown in red, green, and blue respectively. The die orientation is shown as an arrow in red.

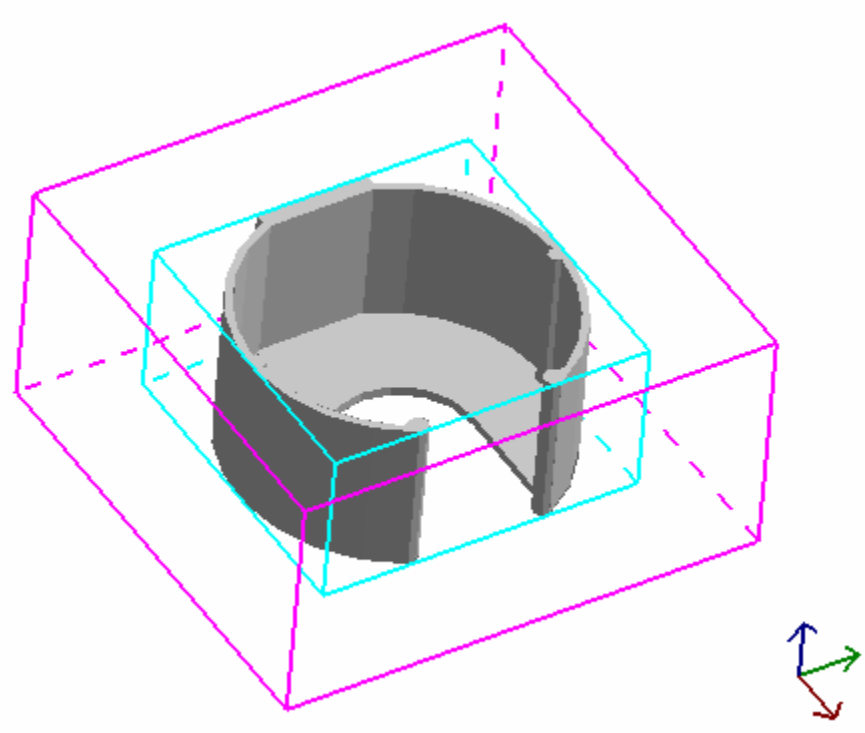


Section II-Figure 7.1: Pick die opening direction

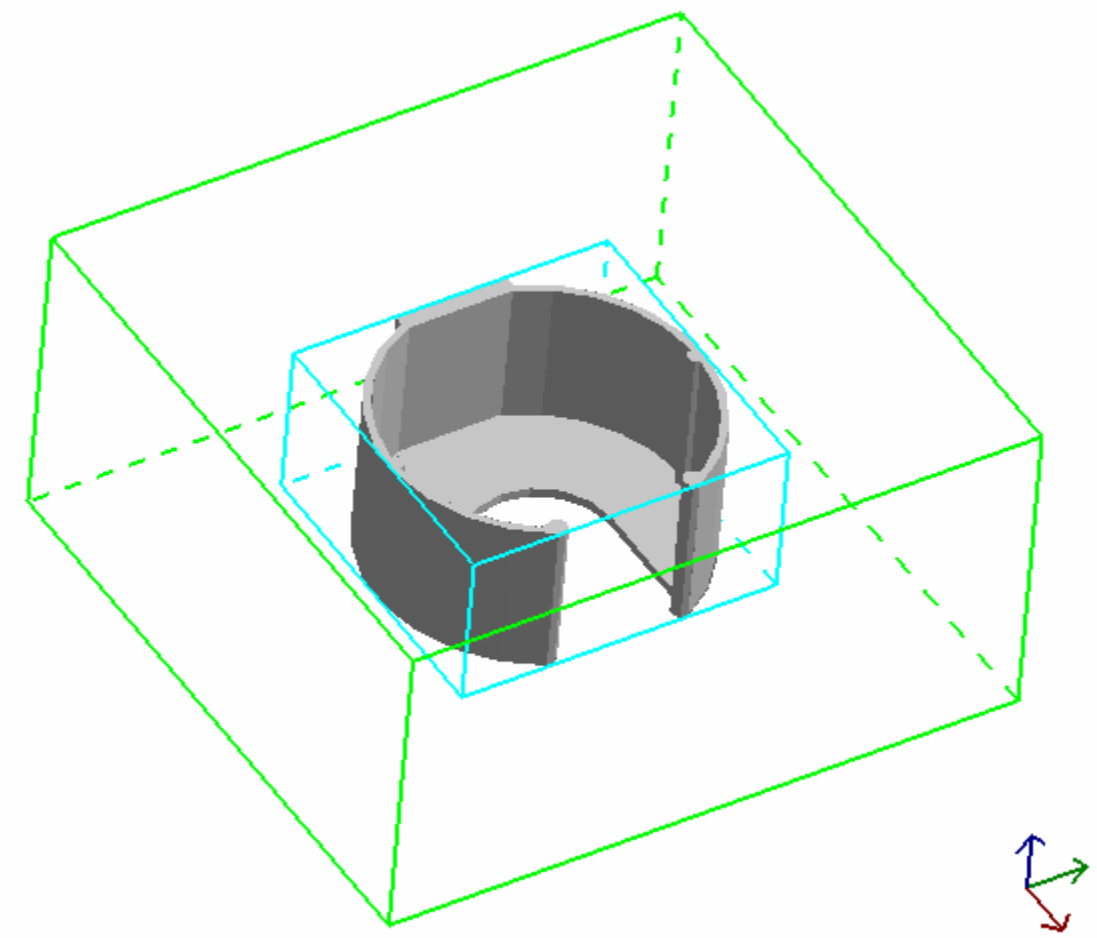


Section II-Figure 7.2: Pick die orientation

Figure 7.3 is the insert box preview and Figure 7.4 is the die box preview. The bounding box is also shown to help the user to decide the offset values. From the figures, the edges of the insert box and the die box are parallel to one of the axis of the *machine coordinate system*. The insert box is in purple, and the die box is in green.



Section II-Figure 7.3: Insert box preview



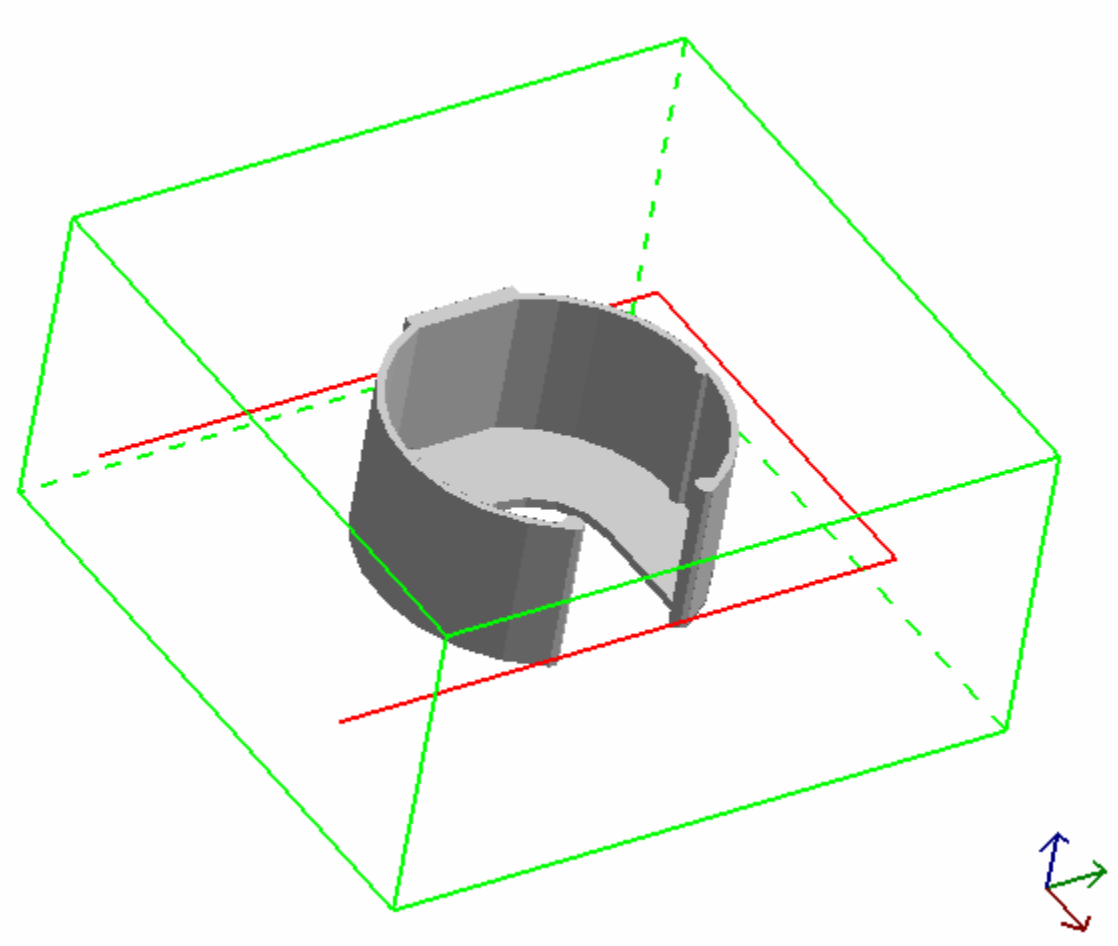
Section II-Figure 7.4: Die box Preview

7.2 Sketching Cooling Line

There are three methods to sketch a cooling line: input node coordinates, orthogonal sketch, and free sketch.

7.2.1 Sketching Cooling Line by Inputting Node Coordinates

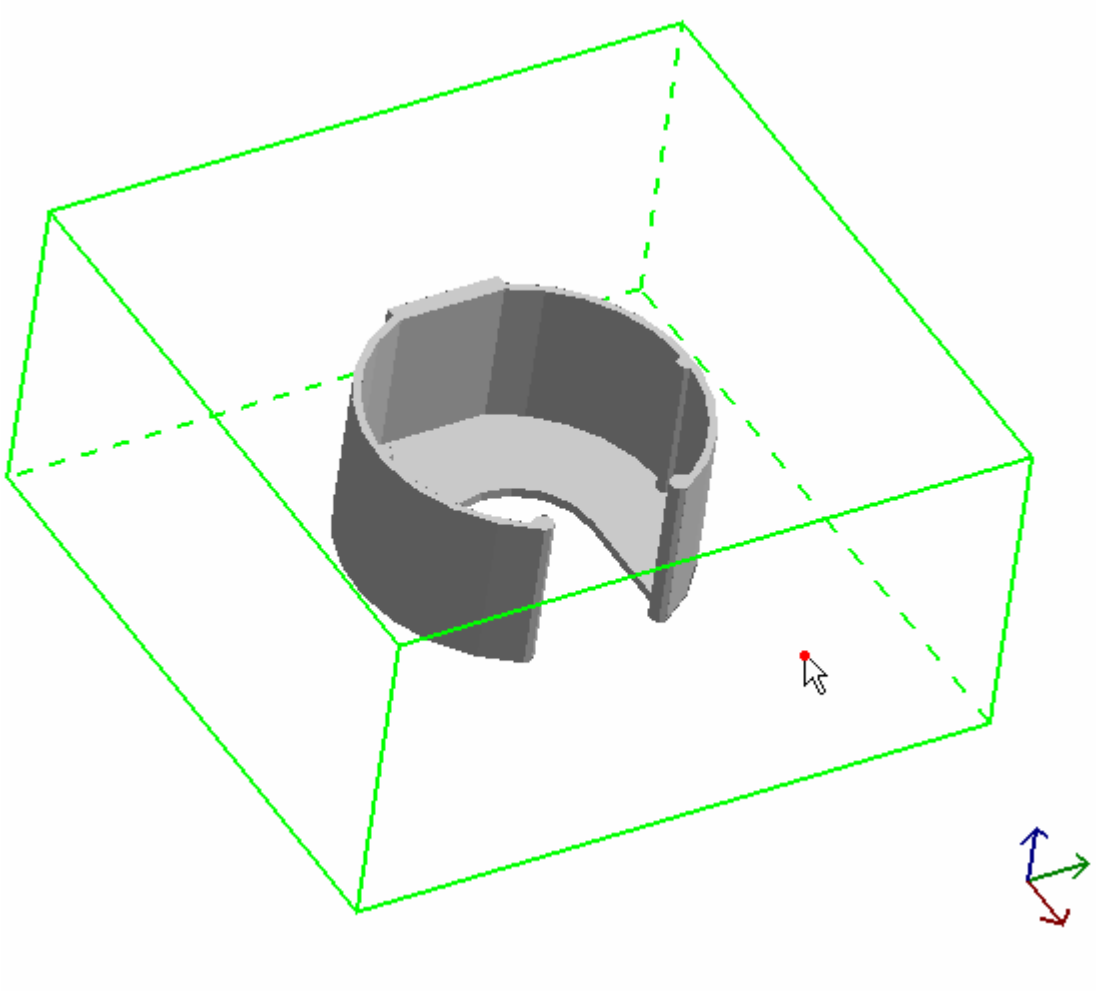
Figure 7.5 shows the cooling line drawn by inputting node coordinates. There are four nodes on this cooling line.



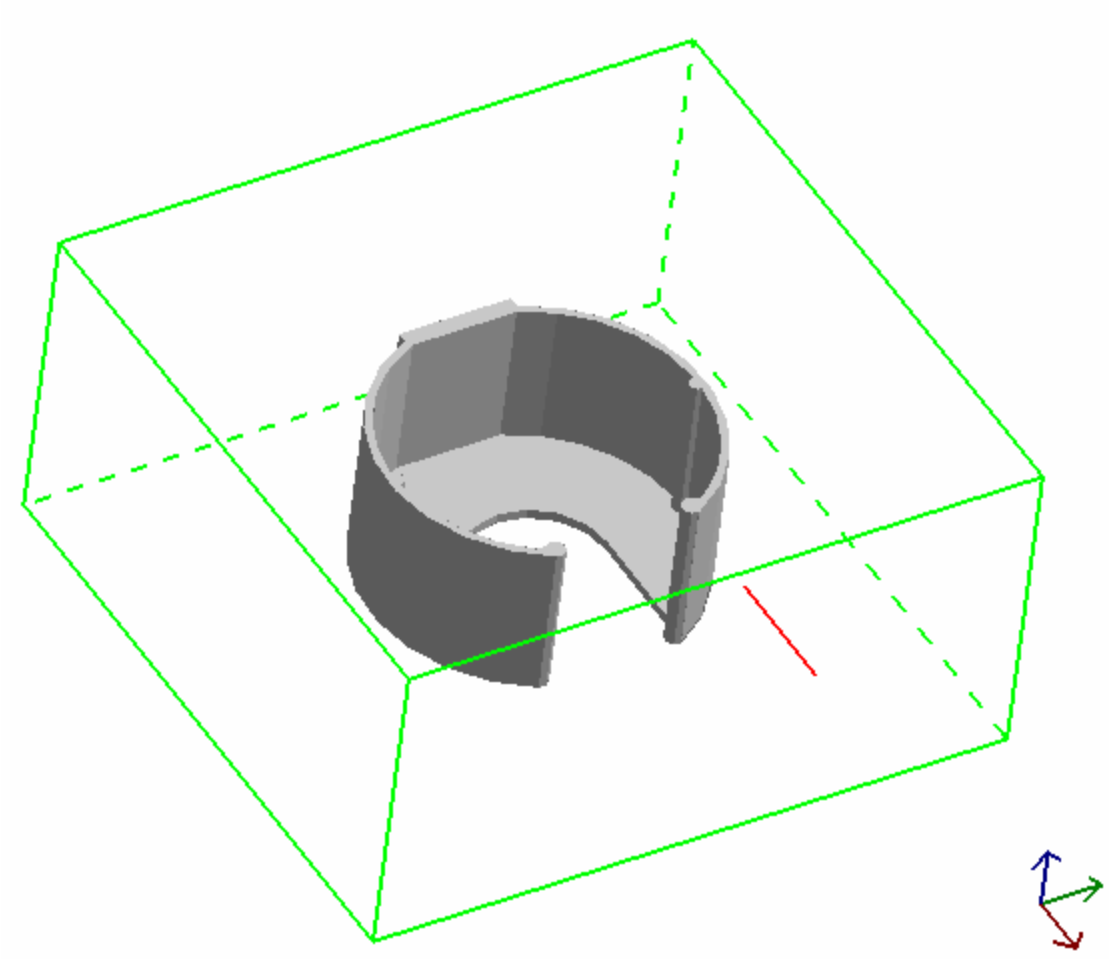
Section II-Figure 7.5: Sketching cooling line by inputting node coordinates

7.2.2 Sketching Cooling Line by Orthogonal Sketch

The entry point is picked by left mouse button double click, as is illustrated in Figure 7.6. The entry point may be on an existing cooling line segment if it happens to be so, or may lie on one of the die box surfaces. Then the cooling line is driven on until it ends using the function keys, as is shown in Figure 7.7.



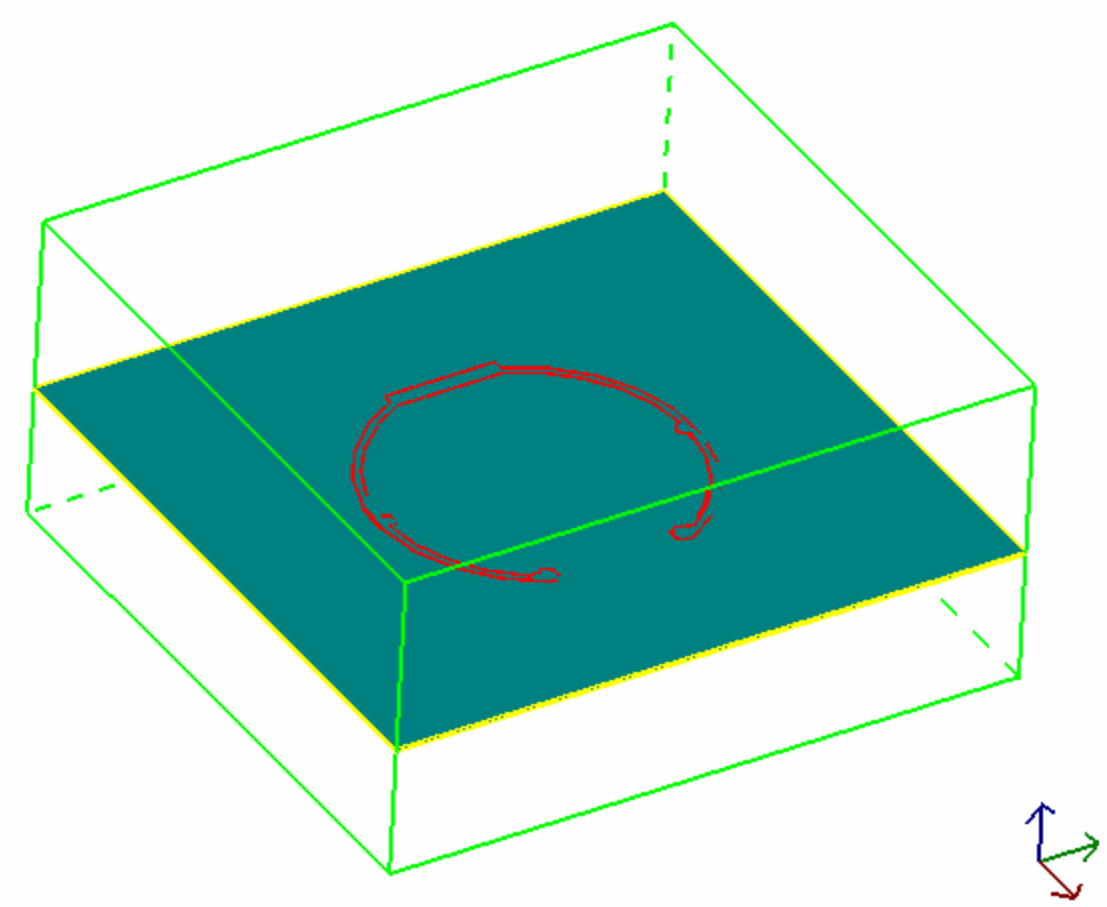
Section II-Figure 7.6: Pick entry point



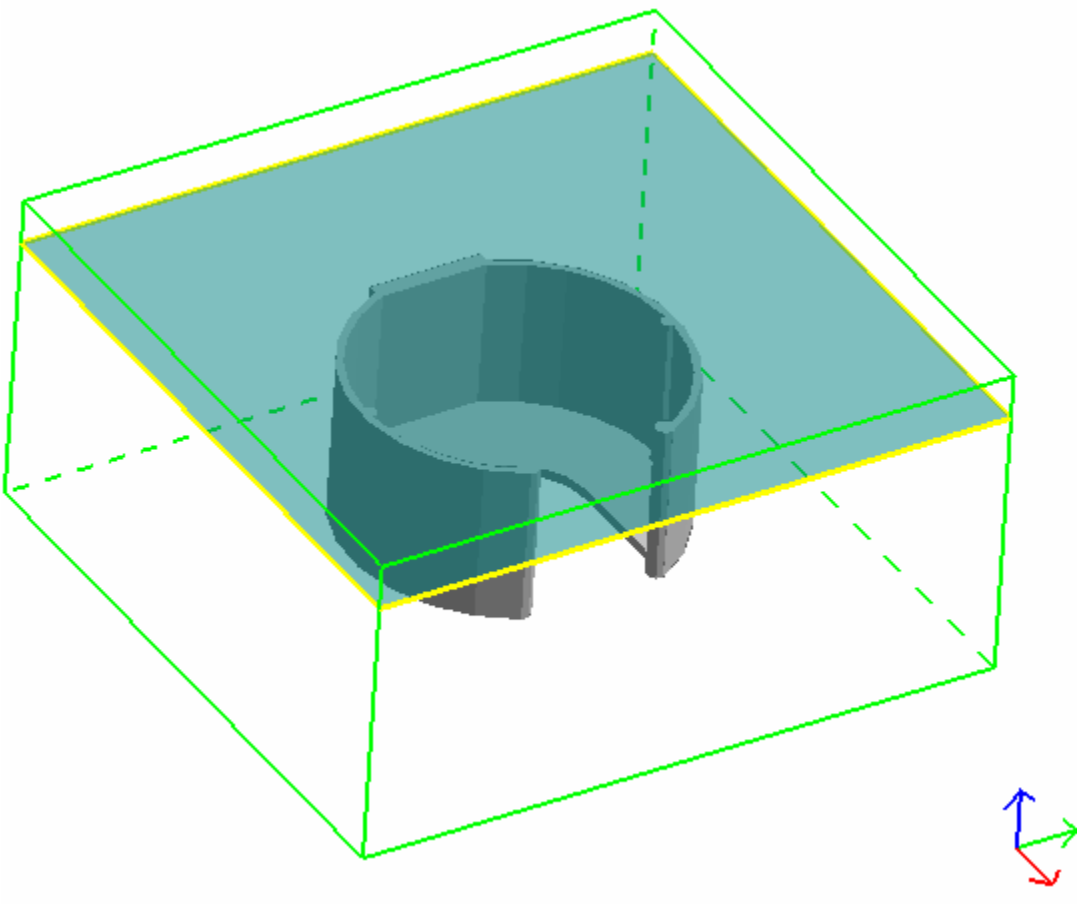
Section II-Figure 7.7: Driving cooling line

7.2.3 Sketching Cooling Line by Free Sketch

Since the cooling line nodes lie on sketch planes, the sketch plane is established first. Figure 7.8 illustrates a sketch plane that intersects with the STL model, and its intersection line loops with the die box and the STL model are also shown in red. Figure 7.9 shows a sketch plane that does not intersects with the STL model, and it only has an intersection line loop with the die box. In the latter case, we don't want the user to lose the relative position of the sketch plane to the STL model, so it is rendered transparently. All sketch planes are rendered in dark green.

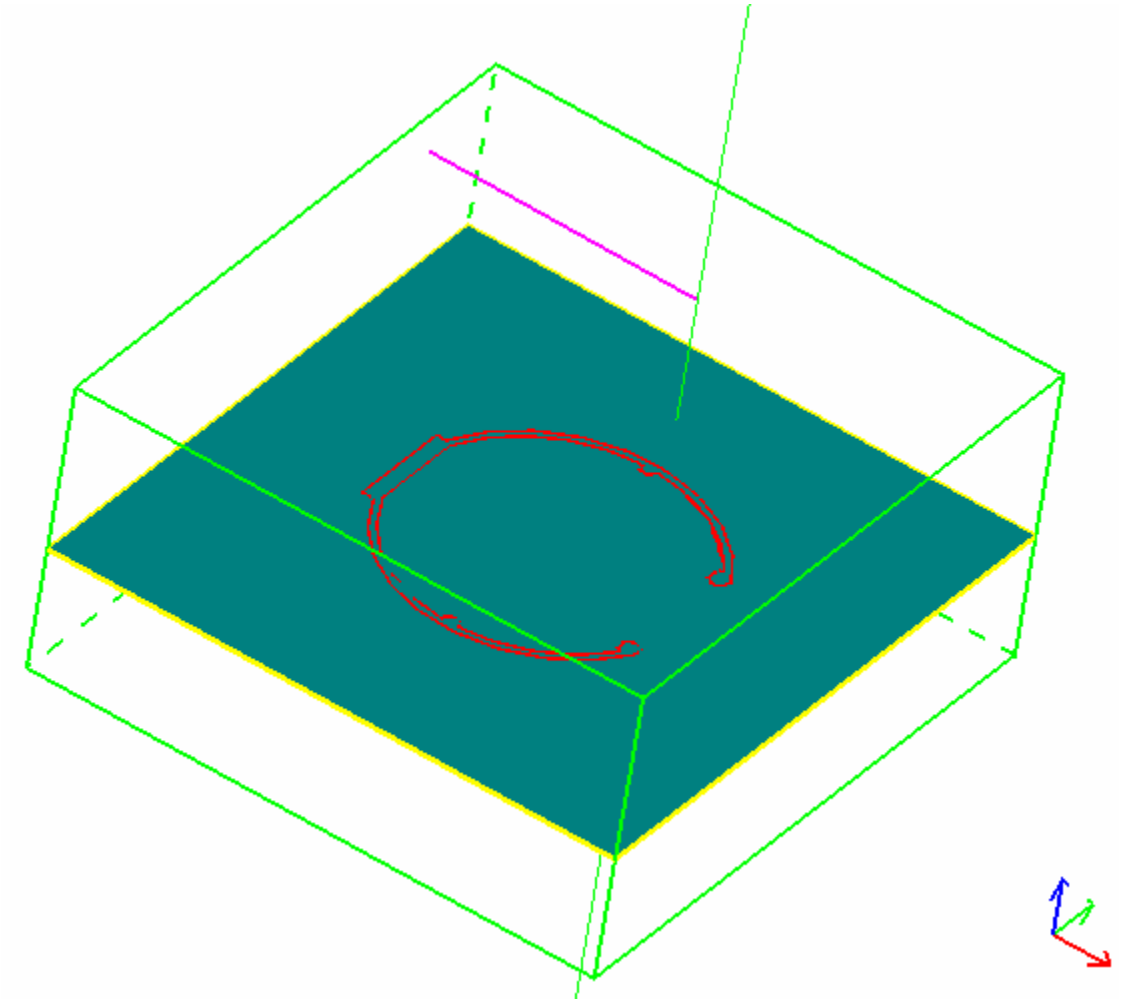


Section II-Figure 7.8: A sketch plane intersecting with STL model



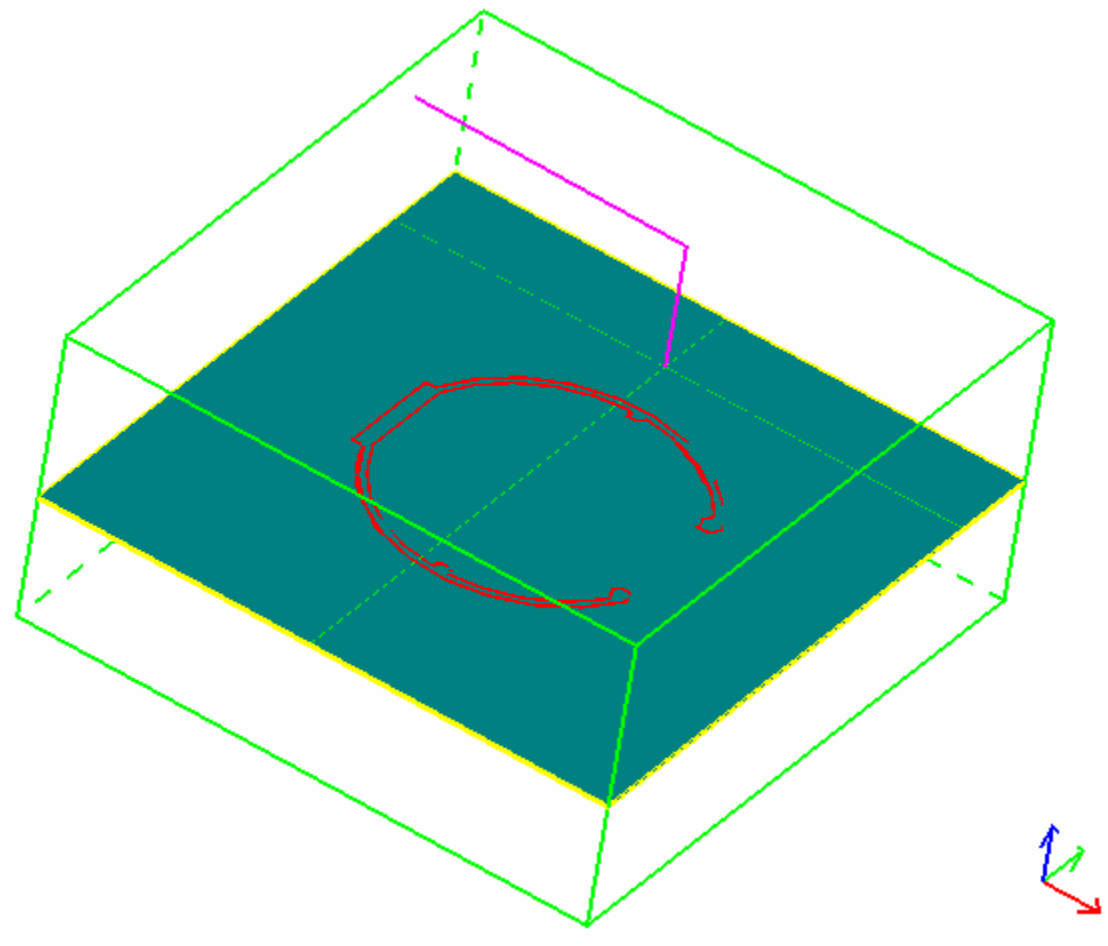
Section II-Figure 7.9: A sketch plane not intersecting with STL model

After the sketch planes are selected, the cooling line nodes are picked on the sketch planes. Figure 7.10 illustrates the cooling line of which the first few nodes are picked on the sketch plane shown in Figure 7.9 and the following nodes are to be selected on the sketch plane shown in Figure 7.8. Figure 7.11 shows that the cooling line continues to be drawn on the sketch plane shown in Figure 7.8.



Section II-Figure 7.10: Changing sketch planes while drawing cooling line

When a cooling line node is picked on a sketch plane, two orthogonal lines, each of which is parallel to one of the edges of the sketch plane, are shown to assist the user to draw cooling lines parallel to the axes of the *machine coordinate system*, as is the most frequently happening case in die casting cooling lines.

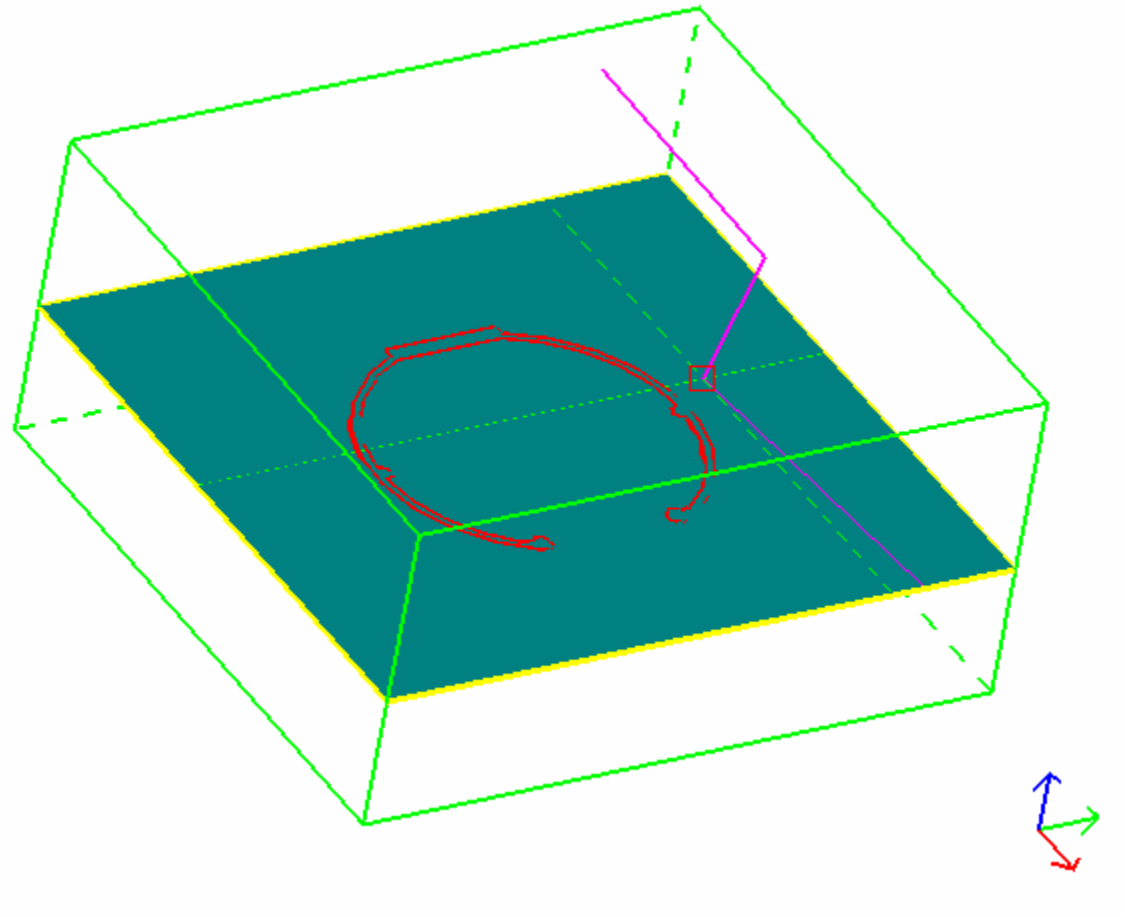


Section II-Figure 7.11: Continue drawing cooling line on new sketch planes

7.2.4 Modifying an Existing Cooling Line

A specific cooling line may be modified using one of the three methods: modifying the node coordinates, orthogonal editing or graphical editing. The first two methods are used in the same way as a new cooling line is drawn.

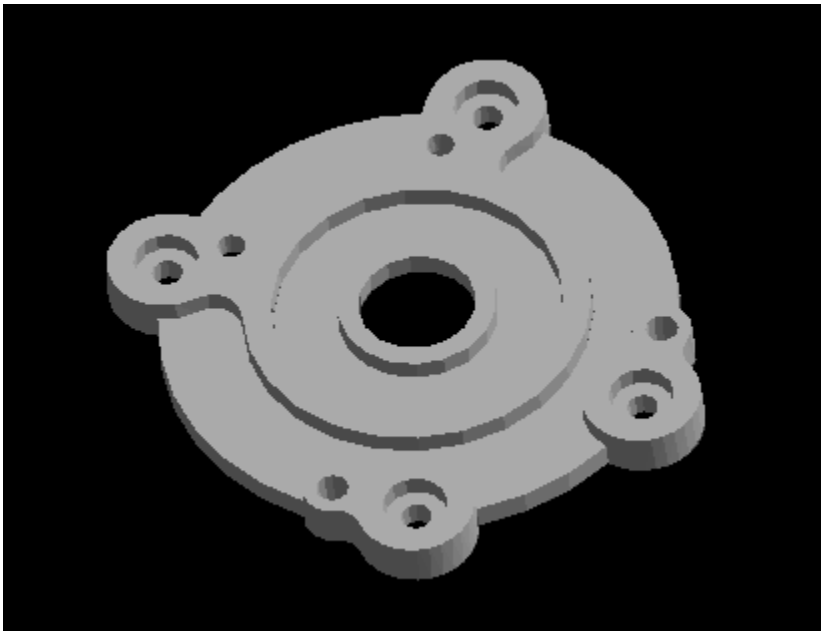
When a node or line segment of a cooling line is to be moved by graphical editing, the user presses the left mouse button and drags the node or line segment to the new desirable position. Figure 7.12 shows a node to be moved on the sketch plane related with it.



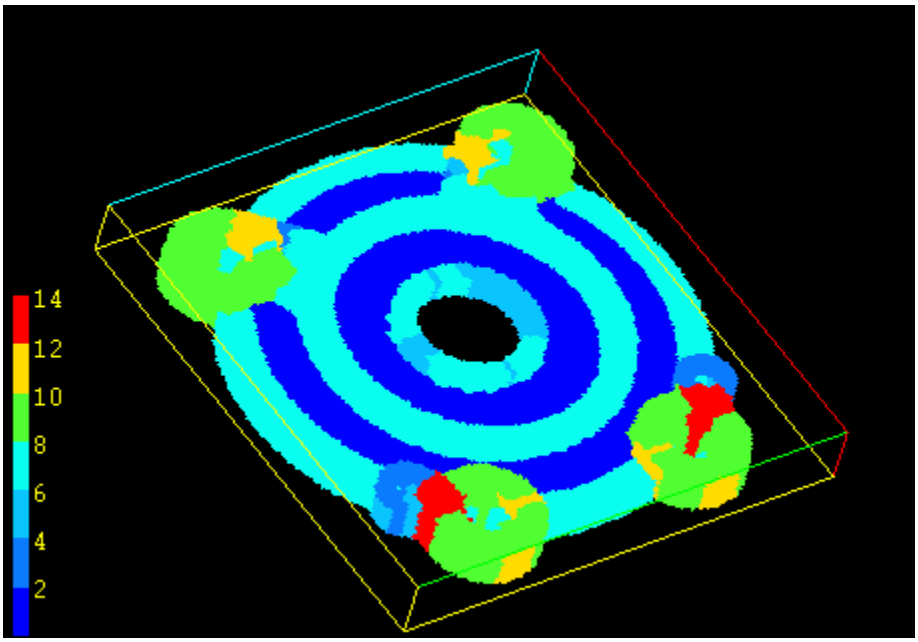
Section II-Figure 7.12: Move a node on the sketch plane

7.3 Wall Thickness Mapping Back

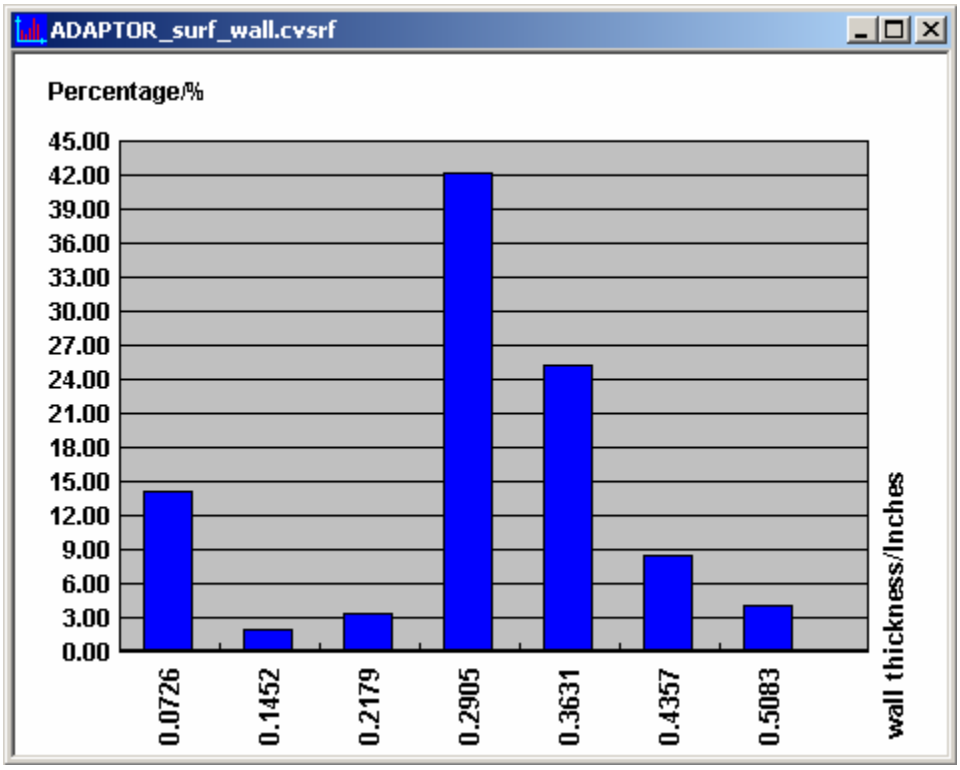
The wall thickness is mapped back onto the Voxel model surface after the thin section analysis. Here we use the casting part of the adaptor as an example. The imported STL model is shown in Figure 7.13, and the mapping back result is illustrated in Figure 7.14. The voxels are classified according to their wall thickness, and the statistical result is shown in Figure 7.15.



Section II-Figure 7.13: Adaptor – STL model



Section II-Figure 7.14: Wall thickness mapped back to voxel model surface



Section II-Figure 7.15: Wall thickness quantitative display

8. CONCLUSIONS

It has been well understood that design and manufacturing must work together in order to achieve quality products with a short lead-time and a low cost. The area of Design for Manufacturing (DFM) has been subject to intense investigation in recent years. Manufacturability evaluation tools have been developed in various domains to identify and resolve the design problems that might arise during the manufacturing process while still in the design stage.

CastView software is an easy-to-use design visualization tool that presents die casting-specific information in 3-D graphical form. This software provides a “quick and dirty” analysis of die casting part designs, which is related with thermal and flow problems, based only on the part geometry and requires no special experience with computer simulation or computer-aided engineering. The analysis is qualitative and generally takes less than one hour regardless the geometry complexity.

The cooling system affects the metal flow and the thermal distribution significantly. This work makes the design and the redesign of the cooling system available before the analysis is processed. Many functions are designed and implemented in the current system, and provide an easy way to define and modify the die configuration and the cooling lines. There may be many schemes for the design of the die configuration and the cooling system. Each scheme may be saved in a file and serve as a candidate so that the optimal one can be chosen after comparing the results of the analyses based on each scheme. The Graphical User Interface makes the user’s work a lot simpler. For example,

the cooling line segment is rendered in a different color from other segments in order to indicate this segment is about to be modified. The Graphical User Interface also provides the user a guidance of how to begin and complete a desired operation.

Since the wall thickness is used to predict the thermal problems and the filling patterns during thick and thin analysis, it is also mapped back onto the voxel model surface to help the user view the casting part globally. This process can also be applied to die surface to predict thermal problems sometimes called steel condition problems. The mapping back results may then serve as an input to the castability evaluation system.

APPENDIX A

DIE CONFIGURATION FILE FORMAT

Created Time: Tue Dec 03 14:14:20 2002 //file created time

Die Opening Direction: 0 1 0 //die opening direction (x,y,z)

X Direction: 0 0 1 //x machine axis vector(x,y,z)

Insert Box Offsets:

40 40 20

//insert box offsets (x-,y-,z-)

40 40 20

//insert box offsets (x+,y+,z+)

Die Box Offsets:

40 40 20

//die box offsets (x-,y-,z-)

40 40 20

//die box offsets (x+,y+,z+)

Die Configuration Finished.

APPENDIX B

COOLING LINE FILE FORMAT

/* The number of sketch planes */

2

/* Sketch plane index, parameters of plane equation Ax+By+Cz+D=0 */

1 0 1 0 -5

2 0 1 0 -60.4828

/* The number of cooling lines */

3

/* File format for each Cooling Line:

Cooling Line index, creation flag, and the number of nodes for this cooling line. Cooling Line properties: fluid medium, line diameter, flow rate, fluid temperature, and heat transfer coefficient. Node coordinates (x,y,z) and related sketch plane index if any.

Creation flag: 1 means to establish this cooling line by inputting node coordinates; 2 means to establish this cooling line orthogonally; 3 means to establish this cooling line using sketch planes */

1 1 2 // line index, creation flag, number of nodes

water //medium flowing through the cooling line

8 mm //diameter of the cooling line

0.2 m/s //flow rate of the cooling fluid

30 C //temperature of the cooling fluid

1358.23 W/m^2-K //heat transfer coefficient of the fluid

-162.287 20 -112.553 //1st node coordinates (x,y,z)

162.215 20 -112.553 //2nd node coordinates (x,y,z)

2 2 2 // line index, creation flag, number of nodes

oil //medium flowing through the cooling line

1 cm //diameter of the cooling line

0.2 m/s //flow rate of the cooling fluid

100 F //temperature of the cooling fluid

203.034 W/m^2-K //heat transfer coefficient of the fluid

1.1126 120 22.7826 //1st node coordinates (x,y,z)

1.11289 -40 22.7826 //2nd node coordinates (x,y,z)

3 3 4 // line index, creation flag, number of nodes

others //medium flowing through the cooling line

8 mm //diameter of the cooling line

0.3 m/s //flow rate of the cooling fluid

20 C //temperature of the cooling fluid

1000 W/m^2-K //heat transfer coefficient of the fluid

162.215 60.4844 108.509 2 //1st node coordinates and its sketch plane index

6.74853 60.484 108.706 2 //2nd node coordinates and its sketch plane index

6.74853 5 108.706 1 //3rd node coordinates and its sketch plane index

-162.287 5.00273 109.228 1 //4th node coordinates and its sketch plane index

APPENDIX C

WALL THICKNESS FILE FORMAT

Castview Wall Thickness file: bowl_mm_surf_wall.cvsrf //file name

Data Created Time: Mon Nov 18 02:19:27 2002 //file create time

//wall thickness flag: 1 means the histogram will show distance from surface, 2 means the //histogram will show the wall thickness

Wall Thickness flag: 2

Voxel Layers: 5 //the maximum skeleton value

Voxel Size: 0.82251 //voxel size

Wall thickness unit: Millimeters //wall thickness unit

Total Number of Voxels: 479044 //total number of voxels

//wall thickness value, number of voxels, and its percentage over total number of voxels

Wall Thickness	Voxel Numbers	Overall Percentage
1.645020	10182	2.125483%
3.290040	218884	45.691836%

4.935060	120306	25.113768%
6.580081	19043	3.975209%
8.225101	110629	23.093703%

---------- OVER ----------

REFERENCES (Section II)

[1] Yagel, R., Lu, S.C., Rebello, A.B., and Miller, R.A., "Volume-based reasoning and visualization of diecastability", *Proceedings of Visualization '95*, Atlanta, GA, pp.359-362

[2] Shah, Jami J., and Wright, Paul K., "Developing Theoretical Foundations of DfM", *Proceedings of the 2000 Design For Manufacturing Conference*, DETC2000/DFM-14015, Baltimore, MD

[3] Bralla, James G., "*Handbook of Product Design for Manufacturing*", McGraw-Hill Book Company, 1986

[4] American Society for Metals, "*Casting Design Handbook*", Metals Park, Ohio, 1962

[5] New Jersey Zinc Company, "*Practical Considerations in Die Casting Design*", New York, 1955

[6] Doehler, H. H., "*Die Casting*", New York, McGraw-Hill, 1951

[7] Herman, E. A., "*Die Casting Die Designing*", The Society of Die Casting Engineering Inc., 1987

[8] Herb, Charles Oliver, "Die-casting: the die-casting process and its application in modern manufacture, die-casting machines, design of different types of dies, composition and properties of die-casting alloys, and the die-casting of zinc, aluminum, brass, and other non-ferrous alloys", 1952

[9] Kurth, Georg Reinhard and Gadh, Rajit, "Virtual prototyping for die-design: Determination of die-open directions for near-net-shape manufactured parts with extruded or rotational features", *Computer Integrated Manufacturing Systems*, Feb. 1997, Vol. 10, No. 1, pp.69-81

[10] Beiter, Kurt A., Cardinal, James M., and Ishii, Kos, "Design for injection molding: balancing mechanical requirements, manufacturing costs, and material selection", *ASME Computer Integrated Concurrent Design Conference*, Sept., 1995, Boston, MA

[11] 3D graphics using the OpenGL API website, http://www.cs.ualberta.ca/~andreas/math/matrfaq_latest.html

[12] http://madmax.me.berkeley.edu/~DFM/casting/casting.html#7

[13] http://ctiparts.com/new_page_1.htm

[14] http://www.etm1.com/diecast.htm

[15] http://www.atnf.csiro.au/computing/software/visualisation/node5.html

[16] Jones, Mark W., "The Production of Volume Data from Triangular Meshes Using Voxelisation", *Computer Graphics Forum*, Volume 15, 1996, Number 5 pp. 311-318

[17] Hearn, D. and Baker, M. P., "*Computer Graphics*", Prentice Hall, Inc., 1997

[18] Foley, James, Dam, Adries van, Feiner, Steven, and Hughes, John, "*Computer Graphics: Principles and Practice*", Addison-Weley Publ. Co., 1996

[19] Hill, F.S., and Hall, Prentice, "*Computer Graphics Using OpenGL"*, 2000

[20] Neider, J., Davis, T., and Woo M., "*OpenGL Programming Guide*", Release 1, 1993

[21] "Learning Visual C++", http://www.vckbase.com/study/

[22] http://www.cs.berkeley.edu/~laura/cs184/quat/quaternion.html

[23] Sreeramoju, Praveen, "Graphical user interface for gate functions", M.S. Thesis, The Ohio State University, 1999

[24] Qiong, Ouyang, "An algorithm for generating a Voxel-based model from a B-rep model", M.S. Thesis, The Ohio State University, 1994